COVERS AND LOGARITHMIC SIGNATURES OF FINITE GROUPS IN CRYPTOGRAPHY

Von der Fakultät für Ingenieurwissenschaften,
Abteilung Informatik und Angewandte Kognitionswissenschaft
der Universität Duisburg-Essen
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften (Dr.-Ing.)

genehmigte Dissertation

von

Pavol Svaba
aus Bratislava, Slowakische Republik

1. Gutachter: Prof. Dr. Tran van Trung
2. Gutachter: Prof. Dr.-Ing. A. J. Han Vinck
Tag der mündlichen Prüfung: 22. Juni 2011

Zusammenfassung

Nachdem Diffie und Hellman [DH76] die Idee von getrennten Schlüsseln für Verschlüsselungsverfahren präsentierten, wurde die asymmetrische Kryptographie zunehmend weiter entwickelt. Viele Public Key Kryptosysteme wurden vorgeschlagen, aber nur wenige wurden letztlich nicht gebrochen. Die meisten von ihnen, die noch heute verwendet werden, basieren auf den bekannten Schwierigkeiten von bestimmten mathematischen Problemen in sehr großen endlichen zyklischen Gruppen.

In den späten 1970ern begann S. Magliveras den Nutzen spezieller Faktorisierungen auf endlichen nicht-abelschen Gruppen, bekannt als logarithmische Signaturen, in der Kryptographie zu erforschen [MOS84, Mag86, MM89a, MM89b]. Später folgten weitere wegweisende Arbeiten von Magliveras, Stinson und Tran van Trung [MST02], die sowohl das Kryptosystem MST_1, welches auf logarithmischen Signaturen basiert, als auch MST_2, das auf einer anderen Art von Gruppen-Überdeckungen – den sogenannten $[\mathsf{s}, \mathsf{r}]$-Gittern – arbeitet, bekannt machten. Bisher sind allerdings noch keine praktische Realisierungen von MST_1 oder MST_2 bekannt. Kürzlich wurde ein neues Public Key Kryptosystem namens MST_3 [LMTW09] entwickelt, das auf der Grundlage von logarithmischen Signaturen und zufälligen Überdeckungen von endlichen nicht-abelschen Gruppen arbeitet. Für eine mögliche Realisierung der generischen Version dieses Systems wurden die Suzuki-2-Gruppen vorgeschlagen.

Das Hauptziel dieser Arbeit liegt darin zu zeigen, dass MST_3 auf Suzuki-2-Gruppen realisiert werden kann. Diese Frage können wir im positiven Sinne beantworten. Es gab einige Änderungen in der Umsetzung der Realisierung des Systems. Das erste Problem besteht darin, effizient zufällige Überdeckungen für große Gruppen mit guten kryptographischen Eigenschaften zu erzeugen. In dem wir den Bezug zum klassischen Belegungsproblem ("the occupancy problem") herstellen, können wir eine Schranke für die Wahrscheinlichkeit, dass eine zufällige Ansammlung von Gruppenelementen eine Überdeckung bilden, bestimmen. Eine Konsequenz daraus ist, dass wir das Problem, zufällige Überdeckungen für beliebige große Gruppen zu erzeugen, lösen können. Weiterhin stellen wir einige Resultate spezieller Computerexperimente bezüglich Überdeckungen und gleichmäßigen Überdeckungen zu verschiedenen Gruppen vor.

Dank ihrer einfachen Struktur erlauben uns die Suzuki-2-Gruppen die Sicherheit des Systems genau zu studieren und es effizient zu implementieren. In der ersten Realisierung wird eine spezielle Klasse von kanonisch logarithmischen Signaturen zu elementar-abelschen 2-Gruppen als Basis für die Schlüsselgenerierung verwendet. Diese sind leicht zu konstruieren und erlauben eine sehr effiziente Faktorisierung. Wir betrachten einen Angriff, der zeigt, dass kanonische Signaturen nicht benutzt werden können um eine sichere Umsetzung von MST_3 mit Suzuki-2-Gruppen zu realisieren. Motiviert durch die Attacke auf die erste Realisierung konnten wir eine neue Variante mit signifikanten Verbesserungen vorstellen, welche die Sicherheit des Systems deutlich stärken. Zu diesem Zweck verwendeten wir für das Setup des Systems eine Funktion zur Maskierung des privaten Schlüssels. Ferner führten wir eine Klasse von fusionierten transversalen logarithmischen Signaturen für die Realisierung des Verfahrens ein. Diese erlauben eine effiziente Faktorisierung mit Hilfe einer "Trapdoor" Information. Wir stellen eine genaue Studie der Sicherheit des Systems vor, in dem wir heuristische und algebraische Methoden verwenden. Zunächst bestimmen wir die untere Schranke der Komplexität bezüglich der Gruppengröße von möglich vorstellbaren direkten Attacken, um den privaten Schlüssel zu erhalten. Diese Schranken geben einen Hinweis auf die Stärke des Systems. Weiterhin entwickeln wir eine mächtige Methode für eine Chosen-Plaintext-Attacke, und zeigen, dass nicht-fusionierte transversale logarithmische Signaturen nicht verwendet werden können. Zudem zeigen wir, dass die vorgeschlagene Klassen von fusionierten transversalen Signaturen dieser Attacke widerstehen, und nach unserem Wissen, sie damit eine sichere Realisierung des Systems ermöglichen. Wir beschreiben und diskutieren die Implementierung des Systems im Detail und ziehen dabei Daten über die Effizienz, die wir als Resultate von einem Experiment erhielten, mit ein.

Abgesehen von dem zentralen Forschungsobjekt werden wir noch einen neuen Ansatz für die Konstruktion pseudo-zufälliger Zahlengeneratoren (PRNG) vorstellen, welcher auf zufälligen Überdeckungen von endlichen Gruppen basiert. PRNGs basierend auf zufälligen Überdeckungen, auch MSTg genannt, zeigten sich bisher zu einer bestimmten Klasse von Gruppen als höchst effizient und produzierten qualitativ hochwertige zufällige Bit-Sequenzen. Eine sehr komplexe Folge von aufwendigen Zufälligkeits-Tests zeigte durch Nutzung der NIST "Statistical Test Suite" und "Diehard Battery of Test" die starken Eigenschaften der neuen Methodik. Noch wichtiger ist allerdings, dass wir Beweise erbringen können, dass diese Klasse von Generatoren adäquat für kryptographische Anwendungen sind. Schließlich fügen wir noch Daten über die Effizienz der Generatoren an und schlagen eine Methode zur praktischen Anwendung vor.

Acknowledgments

I would like to thank my mother for her infinite love. I dedicate this thesis to her.

My deepest thanks to my adviser Professor Tran van Trung for his kind guidance, and all the help and support. It has been a real honour and pleasure to work with and learn from such an exceptional person. I could not have wished for a better mentor.

Thanks to Professor Han van Vinck for his kindness and consistent support through all the years.

Thanks to Professor Spyros Magliveras for his friendship and help, as well as for his work which has been greatly inspiring.

Thanks to Professor Otokar Grošek for opening my eyes to the world of cryptography.

Thanks to the Institute of Experimental Mathematics of University Duisburg-Essen, and the Faculty of Electrical Engineering and Information Technology of the Slovak University of Technology.

Contents

List of Figures

List of Tables

List of Algorithms

List of Used Symbols

Symbol	**Description**
$Z(\mathcal{G})$	Centrum of the group $\mathcal{G}$.
$Aut(\mathcal{G})$	Automorphism group of $\mathcal{G}$.
$GL(m,p)$	General linear group over $\mathbb{F}_p$.
$\mathfrak{S}_n$	Symmetric group of degree n.
$\mathfrak{A}_n$	Alternating group of degree n.
$A(m,\theta)$	Suzuki 2-group (see Section 2.6).
$\mathcal{E}(\mathcal{G})$	The set of all exact transversal logarithmic signatures of $\mathcal{G}$.
$\Lambda(\mathcal{G})$	The set of all transversal logarithmic signatures of $\mathcal{G}$.
$a \parallel b$	Concatenation of two vectors a and b.
$f(x) = \mathcal{O}(g(x))$	$\exists c, x_0 \quad \forall x > x_0 \quad f(x) \leqslant c \cdot g(x)$
$f(x) = \Omega(g(x))$	$\exists c, x_0 \quad \forall x > x_0 \quad f(x) \geqslant c \cdot g(x)$
$f(x) = \Theta(g(x))$	$f(x) = \mathcal{O}(g(x))$ while simultaneously $f(x) = \Omega(g(x))$

Chapter 1

Introduction

"Cryptology is a fascinating discipline at the intersection of computer science, mathematics and electrical engineering."

Bart Preneel

1.1 Public-key Cryptography

Until the 1970s, cryptography was primarily found in diplomatic, military and government applications. Symmetric cryptography was exclusively used to build a secure communication channel between communicating parties. In order to establish such a channel, the users had to share secret information in advance; a private key. In 1976, W. Diffie and M. E. Hellman published their famous paper *New Directions in Cryptography* [DH76] with the revolutionary concept of public key cryptography and provided a solution to the long standing problem of key exchange and pointed the way to digital signatures. Their vision was to employ trapdoor functions to encrypt and digitally sign electronic documents. Informally speaking, a trapdoor function is a function that is easy to compute but hard to invert unless one knows and has access to some specific trapdoor information. Diffie and Hellman's work culminated in a key agreement protocol; the Diffie-Hellman key exchange protocol which allows two parties who share no prior secret to establish a shared secret key over a public channel.

After Diffie and Hellman published their discovery, asymmetric cryptography has became increasingly developing. Many public key cryptosystems have been proposed, but only a few of such systems remain unbroken. Most of them used nowadays are based on the perceived intractability of certain mathematical problems in very large finite cyclic groups in certain particular representations. Prominent hard problems are: a) the problem of factoring large integers; b) the Discrete Logarithm Problem (DLP) in particular representations of large cyclic groups; c) finding a short basis for a given integral lattice $\mathcal{L}$ of large dimension. Unfortunately, in view of P. Shor's quantum algorithms for integer factoring and solving the DLP [Sho97], the known public-key systems will be insecure when quantum computers become practical. A recent report edited by P. Nguyen [Ngu02] identifies these and other problems facing the field of information security in the future. Even though no quantum computers are available yet, when dealing with longterm security of cryptosystems, these "putative" computers should be taken into account. An interesting direction of research, where no structural attack based on the use of quantum computers is known, is the use of computational problems in non-abelian groups.

Already in the late 70's, several authors had started to study the possibility of using group theoretical problems for cryptography. A number of public-key cryptosystems based on combinatorial group theory have been proposed. The first of which was probably the outline of Wagner and Magyarik [WM85]. In particular, owing to Magliveras et al., there are various proposals for cryptographic schemes which make use of special structures, called *logarithmic signatures*.

1.2 Group Factorizations in Cryptography

Logarithmic signatures are a kind of factorization of finite groups. Originally motivated by Sims' bases and strong generators, in the late 70's, S. Magliveras started to study the logarithmic signatures for permutation groups and their usage in cryptography and designed the first potential private key cryptosystem PGM [Mag86]. Statistical and algebraic properties of PGM has been studied [MM89a, MM92], and its usage as a random number generator [MOS84]. Some investigation has been done about logarithmic signatures and their transformations [MM89a, Cus00], or method of applying the logarithmic signatures for generating random elements in a group [Mem89, MM89b]. Later, Magliveras, Stinson, and Tran van Trung have done some preliminary work in designing two public key cryptosystems, MST_1 based on logarithmic signatures, and MST_2 using another type of group coverings, called $[\mathfrak{s}, \mathfrak{r}]$-*meshes* [MST02]. Until now, however, no practical realizations are known for PGM, MST_1 or MST_2.

Recently, a new type of public key cryptosystem, called MST_3 [LMTW09], has been developed on the basis of logarithmic signatures and *random covers* of finite non-abelian groups. For a possible realization of the generic version of this system, the Suzuki 2-groups have been suggested.

1.3 Objectives

The primary objective of this thesis is to verify that MST_3 cryptosystem can be realized using the Suzuki 2-groups. Thereby demonstrating the practical usefulness and effectiveness of schemes based on logarithmic signatures and covers for finite non-abelian groups.

There are several challenges in designing the cryptosystems based on random covers. The first problem is to efficiently generate these structures for large groups and with good cryptographic properties.

An equally important objective of this work is constructing factorizable logarithmic signatures. They provide a trapdoor for the cryptosystem and therefore must not be accessible for the attackers. An effective method for factorization with respect to them is also essential.

A necessary objective for any cryptosystem is the analysis of its security. We would like to develop various attacks exploiting the features of the scheme as well as the structure of the underlying group.

Besides the usage in cryptosystem MST_3, this work also aims at other applications of random covers in cryptography, in particular to designing pseudorandom number generators based on random covers of finite groups.

1.4 Contributions

Our main contribution lies in showing that logarithmic signatures and covers for finite groups can be used in practical cryptography.

First of all, we concentrate our attention on covers and methods of how to efficiently generate covers for large finite groups. We reveal a relation between this problem and the classical occupancy problem, and use it to prove a new bound for the probability for which randomly generated collection of elements for a given group forms a cover. As a consequence, we can solve the problem of generating random covers for arbitrarily large finite groups. The experiments with small alternating groups provide useful hints for generating uniform random covers. Using results from well studied occupancy problem, we are also able to determine the average number of representations for each element covered by a randomly generated cover. This implies a method of how to protect covers against factorization attacks, or provides a tool to prove security of the systems which use random covers (e.g. see Chapter 7).

Secondly, we investigate the realization of public key cryptosystem MST_3 with Suzuki 2-groups. Due to their simple structure, the Suzuki 2-groups enable us to study the security of the system and also provide an efficient implementation. We present a study of its security, proving a new general lower bound for the work effort required to determine the secret key in terms of the size of the underlying group. By exploiting the properties of the group operation in the Suzuki 2-groups, as well as a special structure of canonical logarithmic signatures for elementary abelian groups, we develop and apply an attack, showing that canonical signatures are unfit to use in this realization.

Motivated by the successful attack on the first realization, when canonical signatures are used, we re-design the original scheme introducing a secret homomorphism used to mask a secret logarithmic signature with a random cover. We propose new set-up with randomized encryption without introducing additional ciphertext expansion. We investigate the security of the new variant of MST_3, establishing lower bound for the work effort required to recover an equivalent private key. We provide a study of factorizable logarithmic signatures for elementary abelian 2-groups and transformations used to generate them. In particular, the operation of fusing blocks is underscored as it becomes necessary for the generation of secure private keys for this realization. By developing a powerful plaintext attack, called Matrix-permutation attack, we show that non-fused transversal logarithmic signatures should not be used for a secure realization of new MST_3. For the proposed fused transversal signatures we derive the complexity of the Matrix-permutation attack by computing the workload required to break the system.

We discuss practical issues related to the implementation of MST_3 with Suzuki 2-groups. We present data of key storage and speed performance from experimental implementation of the scheme.

Lastly, we introduce a new application of random covers in cryptography. We develop a very effective method of using random covers for large finite groups to designing pseudorandom number generators, called MSTg. We concentrate on realizations with elementary abelian 2-groups, which allow highly efficient implementation at minimum cost. To show the evidence of excellent properties of proposed generators, we provide extensive statistical experiments using the standard tools for testing randomness of outputs from pseudorandom number generators. We provide argumentation showing that MSTg's are suitable for cryptographic use and discuss the issues related to their use in practice.

1.5 Organization

The remainder of the thesis is organized as follows.

Chapter 2 provides the necessary background of the structures and transformations used to design cryptosystems based on finite groups. We briefly introduce the notation, definitions and some basic facts about logarithmic signatures and covers of finite groups and their induced mappings. This chapter also gives a short survey of known cryptosystems based on these structures. A description of Suzuki 2-groups proposed for the realization of the cryptosystem MST_3 can be found at the end of the chapter.

In Chapter 3 we investigate a method of generating random covers for large finite groups. Showing the connection between this issue and the classical occupancy problem, we are able to determine a bound for the probability for which random collection of group elements compose a cover. We investigate the use of a greedy algorithm for generating covers with high degree of uniformity and present experimental results for small alternating groups.

Chapter 4 focuses on the cryptosystem MST_3. The description of its generic version and the first realization using Suzuki 2-groups can be found here. In this realization, the canonical logarithmic signatures are used as a basis for the key generation. We analyze the security of the system, using the knowledge about the group operation as well as the properties of canonical logarithmic signatures.

In Chapter 5 we present an approach to re-designing MST_3 for use with the Suzuki 2-groups. The method makes use of the characteristics of the group operation as well as the structure of the Suzuki 2-groups. The chapter offers a detailed study of the security of the new scheme.

Chapter 6 shows practical aspects of the re-designed MST_3 and presents the data of performance including the attack complexity for the various parameter sets from an experimental implementation.

Chapter 7 is devoted to a new approach in designing a pseudorandom number generators based on random covers for large finite groups. We suggest a model of the generator and a method of using it in practice. We investigate the statistical properties of the output sequence from the realization of the generator. Data from extensive statistical tests as well as performance data are presented. We discuss its security and cryptographic usage.

Chapter 8 concludes the thesis by summarizing our main contributions and showing the possibilities of future research.

Chapter 2

Preliminaries

In this section, we briefly present notation, definitions and some basic facts about logarithmic signatures, covers for finite groups and their induced mappings. For more details the reader is refered to [Mag86, Mem89, MM89a, MM92, Cus00, Mag02]. We assume that the reader is familiar with the elementary concepts in group theory. The group theoretic notation used is standard and may be found in [Hup67] or in any textbook of group theory. In this thesis, all groups and sets considered are finite, unless otherwise specified.

2.1 Logarithmic Signatures and Covers

Let $\mathcal{G}$ be a finite permutation group of degree[1] n. A *logarithmic signature* for $\mathcal{G}$ is an ordered collection $\alpha = \{A_i : i = 1, \ldots, s\}$ of subsets $A_i = \{a_{i,j} : j = 1, \ldots, r_i\} \subseteq \mathcal{G}$ such that each element $g \in \mathcal{G}$ can be expressed uniquely as a product of the form

$$g = a_1 \cdot a_2 \cdots a_{s-1} \cdot a_s$$

with $a_i \in A_i$.

[1] Apart from the obvious meaning for permutation groups, we define the *degree* of an abstract finite group $\mathcal{G}$ to be the integer n such that $|\mathcal{G}| = \lfloor n \log n \rfloor$.

An Example

Let $\mathcal{G}$ be the Alternating group $\mathfrak{A}_5$ of order 60. Then $\alpha = \{A_1, A_2, A_3\} = \big\{ \{\mathrm{id}, (1\ 2\ 3), (1\ 3\ 2)\},\ \{(1\ 2\ 3), (2\ 4\ 3), (2\ 3\ 4), (1\ 4)(2\ 3)\},\ \{(1\ 4\ 3), (1\ 2\ 5\ 4\ 3), (1\ 2\ 4\ 5\ 3), (2\ 3\ 5), (1\ 5)(2\ 4)\}\big\}$ is a logarithmic signature for $\mathfrak{A}_5$, also represented in its table form, which we will prefer here, as

Figure 2.1: Logarithmic signature for $\mathfrak{A}_5$.

	α
	$a_{1,1} = \mathrm{id}$
A_1	$a_{1,2} = (1\ 2\ 3)$
	$a_{1,3} = (1\ 3\ 2)$
	$a_{2,1} = (1\ 2\ 3)$
	$a_{2,2} = (2\ 4\ 3)$
A_2	$a_{2,3} = (2\ 3\ 4)$
	$a_{2,4} = (1\ 4)(2\ 3)$
	$a_{3,1} = (1\ 4\ 3)$
	$a_{3,2} = (1\ 2\ 5\ 4\ 3)$
A_3	$a_{3,3} = (1\ 2\ 4\ 5\ 3)$
	$a_{3,4} = (2\ 3\ 5)$
	$a_{3,5} = (1\ 5)(2\ 4)$

With respect to α, every element of $\mathfrak{A}_5$ can be expressed uniquely, for example

$$\begin{array}{ll}
\mathrm{id} = & a_{1,2} \cdot a_{2,4} \cdot a_{3,1} \\
(1\ 2)(3\ 4) = & a_{1,1} \cdot a_{2,1} \cdot a_{3,1} \\
(1\ 4\ 5\ 2\ 3) = & a_{1,2} \cdot a_{2,2} \cdot a_{3,4} \\
(1\ 2\ 5) = & a_{1,3} \cdot a_{2,3} \cdot a_{3,5} \\
\text{etc.} &
\end{array}$$

If we take $A'_2 = A_2 \cup \{a_{2,5} = (1\ 2\ 3\ 4\ 5)\},\ A'_1 = A_1,\ A'_3 = A_3$ and we set $\alpha' = \{A'_1, A'_2, A'_3\}$, then we get the following table

Figure 2.2: Cover for $\mathfrak{A}_5$.

	α'
A'_1	$a_{1,1} = \mathrm{id}$
	$a_{1,2} = (1\ 2\ 3)$
	$a_{1,3} = (1\ 3\ 2)$
A'_2	$a_{2,1} = (1\ 2\ 3)$
	$a_{2,2} = (2\ 4\ 3)$
	$a_{2,3} = (2\ 3\ 4)$
	$a_{2,4} = (1\ 4)(2\ 3)$
	$a_{2,5} = (1\ 2\ 3\ 4\ 5)$
A'_3	$a_{3,1} = (1\ 4\ 3)$
	$a_{3,2} = (1\ 2\ 5\ 4\ 3)$
	$a_{3,3} = (1\ 2\ 4\ 5\ 3)$
	$a_{3,4} = (2\ 3\ 5)$
	$a_{3,5} = (1\ 5)(2\ 4)$

With respect to α', some elements of $\mathfrak{A}_5$ can be expressed in more than one possible way, for example

$$\begin{aligned}
(1\ 5\ 2) &= a_{1,1} \cdot a_{2,5} \cdot a_{3,2} = a_{1,2} \cdot a_{2,1} \cdot a_{3,4} \\
(1\ 4)(2\ 3) &= a_{1,1} \cdot a_{2,5} \cdot a_{3,5} = a_{1,1} \cdot a_{2,2} \cdot a_{3,1} \\
(2\ 3\ 5) &= a_{1,2} \cdot a_{2,5} \cdot a_{3,2} = a_{1,3} \cdot a_{2,1} \cdot a_{3,4} \\
(1\ 3\ 4) &= a_{1,2} \cdot a_{2,5} \cdot a_{3,5} = a_{1,2} \cdot a_{2,2} \cdot a_{3,1} \\
&\text{etc.}
\end{aligned}$$

Then α' is called a *cover* for $\mathfrak{A}_5$.

In general, let $\mathcal{G}$ be a finite abstract group, we define the *width* of $\mathcal{G}$ to be the positive integer $w = \lceil \log |\mathcal{G}| \rceil$. Denote by $\mathcal{G}^{[\mathbb{Z}]}$ the collection of all finite sequences of elements in $\mathcal{G}$ and view the elements of $\mathcal{G}^{[\mathbb{Z}]}$ as single-row matrices with entries in $\mathcal{G}$. Let $X = [x_1, x_2, \ldots, x_r]$ and $Y = [y_1, y_2, \ldots, y_s]$ be two elements in $\mathcal{G}^{[\mathbb{Z}]}$. We define

$$X \cdot Y = [x_1y_1, x_1y_2, \ldots, x_1y_s, x_2y_1, x_2y_2, \ldots, x_2y_s, \ldots, x_ry_1, x_ry_2, \ldots, x_ry_s]$$

Instead of $X \cdot Y$ we will also write $X \otimes Y$ as ordinary tensor product of matrices, or for short we will write XY. If $X = [x_1, \ldots, x_r] \in \mathcal{G}^{[\mathbb{Z}]}$, we denote by $\overline{X}$ the element $\sum_{i=1}^{r} x_i$ in the group ring $\mathbb{Z}\mathcal{G}$.

Definition 2.1.1 *Suppose that* $\alpha = [A_1, A_2, \ldots, A_s]$ *is a sequence of* $A_i \in \mathcal{G}^{[\mathbb{Z}]}$*, such that* $\sum_{i=1}^{s} |A_i|$ *is bounded by a polynomial in* $\log |\mathcal{G}|$*. Let*

$$\overline{A_1} \cdot \overline{A_2} \cdots \overline{A_s} = \sum_{g \in \mathcal{G}} \lambda_g g \ , \qquad \lambda_g \in \mathbb{Z}$$

Let $\mathcal{S}$ *be a subset of* $\mathcal{G}$*, then we say that* α *is*

(i) *a* **cover** *for* $\mathcal{G}$ *(or* $\mathcal{S}$*), if* $\lambda_g > 0$ *for all* $g \in \mathcal{G}$ *(*$g \in \mathcal{S}$*).*

(iii) *a* **pseudo logarithmic signature** *for* $\mathcal{G}$ *(or* $\mathcal{S}$*), if* $\prod_{i=1}^{s} |A_i| = |\mathcal{G}|$.

(iii) *a* **logarithmic signature** *for* $\mathcal{G}$ *(or* $\mathcal{S}$*), if* $\lambda_g = 1$ *for every* $g \in \mathcal{G}$ *(*$g \in \mathcal{S}$*).*

Thus, a *cover* $\alpha = [A_1, \ldots, A_s]$ for a subset $\mathcal{S}$ of a finite group $\mathcal{G}$ can be viewed as an ordered collection of subsets A_i of $\mathcal{G}$ with $|A_i| = r_i$ such that each element $g \in \mathcal{S}$ can be expressed in at least one way as a product of the form

$$g = a_1 \cdot a_2 \cdots a_{s-1} \cdot a_s \tag{2.1.1}$$

for $a_i \in A_i$. If every $g \in \mathcal{S}$ can be expressed in exactly one way by Equation 2.1.1, then α is called a *logarithmic signature* for $\mathcal{S}$. Thus, logarithmic signatures are a special class of covers. The concept of logarithmic signatures for permutation groups have been introduced by S. Magliveras [MOS84, Mag86, MM89a].

In this thesis, a cover (or logarithmic signature) for a subset of $\mathcal{G}$ is sometimes called a cover (or logarithmic signature) for $\mathcal{G}$ on ambiguity.

In a special case, let s, r be positive integers, a cover $\alpha = [A_1, \ldots, A_s]$ is called $[s, r]$-*mesh* if

(i) $A_i \in \mathcal{G}^{[\mathbb{Z}]}$ and $|A_i| = r$ for each $i \in \{1, \ldots, s\}$

(ii) in $\sum_{g \in \mathcal{G}} \lambda_g g = \overline{A_1} \cdots \overline{A_s}$, the distribution of $\{\lambda_g : g \in \mathcal{G}\}$ is approximately uniform.

The uniformity of a mesh is measured by applying the standard statistical uniformity measures to the distribution $\{\lambda_g : g \in \mathcal{G}\}$, or to the probability distribution $\{P_g : g \in \mathcal{G}\}$, where $P_g = \lambda_g / r^s$ (see [MST02]).

The A_i are called the *blocks*, the vector $(r_1, \ldots, r_s)$ with $r_i = |A_i|$ the *type* of α, and the sum $\ell = \sum_{i=1}^{s} r_i$ is called the *length* of α. We say that α is *nontrivial* if $s \geqslant 2$ and $r_i \geqslant 2$ for $1 \leqslant i \leqslant s$; otherwise α is said to be *trivial*.

Cover α is called ***factorizable*** (or ***tame***) if the factorization in Equation 2.1.1 can be achieved in time polynomial in the width w of $\mathcal{G}$, it is called *non-factorizable* (or *wild*) if it is not tame.

2.2 Cover Mappings

We now define a mapping induced by a cover (or logarithmic signature).

Definition 2.2.1 *Let* $\alpha = [A_1, A_2, \dots, A_s]$ *be a cover (or logarithmic signature) of type* $(r_1, r_2, \dots, r_s)$ *for* $\mathcal{G}$ *with* $A_i = [a_{i,0}, a_{i,1}, \dots, a_{i,r_i-1}]$ *and let* $m = \prod_{i=1}^{s} r_i$. *Let* $m_1 = 1$ *and* $m_i = \prod_{j=1}^{i-1} r_j$ *for* $i = 2, \dots, s$. *Let* τ *denote the canonical bijection from* $\mathbb{Z}_{r_1} \oplus \mathbb{Z}_{r_2} \oplus \cdots \oplus \mathbb{Z}_{r_s}$ *on* $\mathbb{Z}_m$*; i.e.*

$$\tau : \quad \mathbb{Z}_{r_1} \oplus \mathbb{Z}_{r_2} \oplus \cdots \oplus \mathbb{Z}_{r_s} \to \mathbb{Z}_m$$

$$\tau(j_1, j_2, \dots, j_s) := \sum_{i=1}^{s} j_i . m_i .$$

Using τ *we now define the surjective* **mapping** $\breve{\alpha}$ **induced by** α.

$$\begin{aligned} \breve{\alpha} \quad &: \quad \mathbb{Z}_m \to \mathcal{G} \\ \breve{\alpha}(x) \quad &:= \quad a_{1,j_1} \cdot a_{2,j_2} \cdots a_{s,j_s}, \end{aligned} \tag{2.2.1}$$

where $(j_1, j_2, \dots, j_s) = \tau^{-1}(x)$.

Since τ and τ^{-1} are efficiently computable, the mapping $\breve{\alpha}(x)$ is efficiently computable.

Conversely, given a cover α and an element $y \in \mathcal{G}$, to determine any element $x \in \breve{\alpha}^{-1}(y)$ it is necessary to obtain *any one* of the possible factorizations of type (2.2.1) for y and determine indices $j_1, j_2, \dots, j_s$ such that $y = a_{1,j_1} \cdot a_{2,j_2} \cdots a_{s,j_s}$. This is possible if and only if α is tame. Once a vector $(j_1, j_2, \dots, j_s)$ has been determined, $\breve{\alpha}^{-1}(y) = \tau(j_1, j_2, \dots, j_s)$ can be computed efficiently.

An Example

Let α be a random cover of type $(4, 4, 4, 4)$ for $\mathfrak{A}_5$, it induces a mapping $\breve{\alpha} : \mathbb{Z}_{256} \longrightarrow \mathfrak{A}_5$ with $\tau : \mathbb{Z}_4 \oplus \mathbb{Z}_4 \oplus \mathbb{Z}_4 \oplus \mathbb{Z}_4 \longrightarrow \mathbb{Z}_{256}$. For a chosen $x \in \mathbb{Z}_{256}$, we get $\tau^{-1}(x) = (x_1, x_2, x_3, x_4)$, $x_i \in \mathbb{Z}_4$. For example, let $x = 121$, $\tau^{-1}(121) = (1, 2, 3, 1)$

Figure 2.3: A mapping induced by cover α.

	α
	$a_{1,0} = (2\ 5\ 3)$
►	$\mathbf{a_{1,1} = (2\ 5\ 4)}$
	$a_{1,2} = (1\ 2\ 5\ 3\ 4)$
	$a_{1,3} = (1\ 4\ 2\ 5\ 3)$
	$a_{2,0} = (1\ 2)(3\ 5)$
	$a_{2,1} = (1\ 3\ 4)$
►	$\mathbf{a_{2,2} = (1\ 3\ 2\ 5\ 4)}$
	$a_{2,3} = (1\ 5\ 2\ 4\ 3)$
	$a_{3,0} = (2\ 5\ 3)$
	$a_{3,1} = (1\ 3\ 5\ 4\ 2)$
	$a_{3,2} = (1\ 4\ 5\ 3\ 2)$
►	$\mathbf{a_{3,3} = (1\ 4\ 5)}$
	$a_{4,0} = (1\ 2\ 3\ 5\ 4)$
►	$\mathbf{a_{4,1} = (1\ 3\ 4\ 2\ 5)}$
	$a_{4,2} = (1\ 5\ 2\ 3\ 4)$
	$a_{4,3} = (1\ 5\ 3\ 2\ 4)$

$\breve{\alpha}(121) = (2\ 5\ 4) \cdot (1\ 3\ 2\ 5\ 4) \cdot (1\ 4\ 5) \cdot (1\ 3\ 4\ 2\ 5) = (1\ 4\ 3\ 5\ 2)$

The reverse operation would be to find any $z \in \mathbb{Z}_{256}$ with $\tau^{-1}(z) = (z_1, z_2, z_3, z_4)$, $z_i \in \mathbb{Z}_4$, such that for the given element $g \in \mathfrak{A}_5$, we get $\breve{\alpha}(z) = a_{1,z_1} \cdot a_{1,z_2} \cdot a_{1,z_3} \cdot a_{1,z_4} = g$.

In general, the problem of finding a factorization with respect to a randomly generated cover (called *random cover*) is presumedly intractable. There is strong evidence to support the hardness of this problem. For example, let $\mathcal{G}$ be a cyclic group and g be a generator of $\mathcal{G}$. Let $\alpha = [A_1, A_2, \ldots, A_s]$ be any cover for $\mathcal{G}$, for which the elements of A_i are written as powers of g. Then the factorization with respect to α amounts to solving the Discrete Logarithm Problem (DLP) in $\mathcal{G}$.

Therefore, we conjecture the following cryptographic hypothesis (see also [Mag02,MST02]).

Cryptographic Hypothesis 1 *Let $\alpha = [A_1, A_2, \ldots, A_s]$ be a random cover for a "large" subset $\mathcal{S}$ of a group $\mathcal{G}$, then finding a factorization in Equation 2.2.1 is an intractable problem. In other words, the mapping $\breve{\alpha} : \mathsf{Z}_m \to \mathcal{S}$ induced by α with $m = \prod_{i=1}^{s} |A_i|$ is a one-way function.*

This hypothesis is a crucial point that makes covers useful for group based cryptography.

The problem of how to generate random covers for finite groups of large order has been solved and will be discussed in Chapter 3. A probabilistic method shows that generation of random covers for groups of large order can be done with high efficiency and at minimum cost.

Definition 2.2.2 *Two covers (or logarithmic signatures) α, β are said to be* **equivalent** *if* $\breve{\alpha} = \breve{\beta}$.

2.3 Transformations of covers

In this section, we look at different types of transformations that can apply to covers. Here, we consider only those which are used in the next sections. We are particularly interested in transformations that produce inequivalent covers.

Assume that $\alpha = [A_1, A_2, \ldots, A_s] := (a_{i,j})$ is a cover of type $(r_1, r_2, \ldots, r_s)$ for $\mathcal{G}$.

2.3.1 Two-sided transform

Let $t_0, t_1, \ldots, t_s \in \mathcal{G}$, and consider $\beta = [B_1, B_2, \ldots, B_s]$ with $B_i = t_{i-1}^{-1} A_i t_i$. We say that β is a *two sided transform* of α by $t_0, t_1, \ldots, t_s$.

In the special case, where $t_0 = t_s = 1_{\mathcal{G}}$, β is called a *sandwich* of α (see also [MM89a, MM92, Mag02]). Note that β is a cover for $\mathcal{G}$. (If α is a logarithmic signature, so is β.)

If $t_0 = t_1 = \cdots = t_{s-1} = 1_{\mathcal{G}}$ and $t_s = t$, β is called *a right translation* of α by t.

Theorem 2.3.1 (Magliveras [Mag02]) *Let α and β be two covers (or logarithmic signatures) of the same type $(r_1, \ldots, r_s)$ for a group $\mathcal{G}$. Then α and β are equivalent if and only if they are sandwiches of each other.*

A cover (or logarithmic signature) $\beta := (b_{i,j})$ for a group $\mathcal{G}$ is called *normalized* if $b_{i,1} = 1_{\mathcal{G}}$ for each $i = 1, \ldots, s-1$. Clearly, the normalized cover β can be constructed as a sandwich of cover $\alpha = (a_{i,j})$ for $\mathcal{G}$ with $t_0 = t_s = 1_{\mathcal{G}}$, $t_1 = a_{1,1}^{-1}$, $t_2 = a_{2,1}^{-1}.a_{1,1}^{-1}$, $\ldots$, $t_{s-1} = \prod_{i=1}^{s-1} a_{s-i,1}^{-1}$. This can be done by the following algorithm (see also [MM89a]).

Algorithm 1 Construction of normalized cover β equivalent to cover α

Input: cover $\alpha := (a_{i,j})$ of type $(r_1, \ldots, r_s)$ for $\mathcal{G}$

Output: normalized cover $\beta := (b_{i,j})$ equivalent to α

```
 1: $t_0 \leftarrow 1_{\mathcal{G}}$
 2: $t_s \leftarrow 1_{\mathcal{G}}$
 3: for $i \leftarrow 1$ to $s - 1$ do
 4:     $t_i \leftarrow a_{i,1}^{-1} \cdot t_{i-1}$
 5:     for $j \leftarrow 1$ to $r_i$ do
 6:         $b_{i,j} \leftarrow t_{i-1}^{-1} \cdot a_{i,j} \cdot t_i$
 7:     end for
 8: end for
 9: for $j \leftarrow 1$ to $r_s$ do
10:     $b_{s,j} \leftarrow t_{s-1}^{-1} \cdot a_{s,j} \cdot t_s$
11: end for
```

As normalized β is a sandwich of α, the covers are equivalent.

2.3.2 Block shuffle

Let ξ be a permutation in $\mathfrak{S}_s$. Then cover $\beta = [B_1, \ldots, B_s]$ created by applying ξ on block indices of α is called *block permutation (block shuffle)*, i.e. $B_i = A_{\xi(i)}$ for all $i = 1, \ldots, s$. If $\mathcal{G}$ is an abelian and α is a logarithmic signature, so is β. In other words, to preserve logaritmic signature we are able to use the block shuffling operation only if the underlying group is abelian.

2.3.3 Element shuffle

Let π_k be a permutation in $\mathfrak{S}_{r_k}$. The cover $\beta = [B_1, \ldots, B_k, \ldots, B_s] := (b_{i,j})$ created by shuffling the elements in the block k with π_k is called *element permutation (element shuffle)*, i.e.

(i) $B_i = A_i \quad \text{for } i \neq k$

(ii) $b_{i,j} = a_{i,\pi_k(j)} \quad \text{for } j = 1, \ldots, r_i$

If α is a logarithmic signature, so is β.

2.3.4 Fusion of blocks

We create a new cover (logarithmic signature) $\beta = [B_1, \dots, B_{s-1}]$ from $\alpha = [A_1, \dots, A_s]$ by *fusing blocks* k and $k+1$ into a single block of length $r_k \cdot r_{k+1}$ as follows

(i) $B_i = A_i \quad$ for $1 \leqslant i \leqslant k-1$

(ii) $B_k = A_k \otimes A_{k+1}$

(iii) $B_i = A_{i+1} \quad$ for $k+1 \leqslant i \leqslant s-1$

It follows that $\breve{\alpha} = \breve{\beta}$, therefore α and β are equivalent. Note that if the group $\mathcal{G}$ is non-abelian, the fusion operation can be applied to two consecutive blocks. We also see that two equivalent covers need not to have the same type.

An Example

Let $\alpha = [A_1, A_2, A_3, A_4]$ be a logaritmic signature of type $(3, 4, 5, 2)$ for $\mathfrak{S}_5$. We create fused $\alpha' = [A'_1, A'_2, A'_3]$ of type $(3, 4, 10)$ by setting $A'_1 = A_1$ and $A'_2 = A_2$, and A'_3 being constructed by the fusion of blocks A_3 and A_4.

Figure 2.4: Fusion of blocks A_3 and A_4 of logarithmic signature for $\mathfrak{S}_5$.

	α
	id
A_1	(1 3 2)
	(1 2 3)
	id
A_2	(1 4 3)
	(2 3 4)
	(1 4)(2 3)
	(1 3)(2 4)
	(1 5 4 2 3)
A_3	(1 4)(3 5)
	(1 5 2 4 3)
	(1 5)(3 4)
	(2 3 5)
A_4	(1 3 2 5)

$\longrightarrow$

	α'
	id
A'_1	(1 3 2)
	(1 2 3)
	id
A'_2	(1 4 3)
	(2 3 4)
	(1 4)(2 3)
	(1 5 2 4 3)
A'_3	(1 2 5 4 3)
	(1 4)(2 3)
	(1 2 4 5 3)
	(1 2 3 4 5)
	(1 2 4 5)
	(4 5)
	(1 4 3)(2 5)
	(2 4)
	(2 5 3 4)

2.4 Transversal Logarithmic Signatures

Let $\gamma : 1_{\mathcal{G}} = \mathcal{G}_0 < \mathcal{G}_1 < \cdots < \mathcal{G}_s = \mathcal{G}$ be a chain of subgroups of $\mathcal{G}$, and let A_i be an ordered, complete set of right coset representatives of $\mathcal{G}_i$ modulo $\mathcal{G}_{i-1}$. It is clear that $[A_1, \ldots, A_s]$ forms a logarithmic signature for $\mathcal{G}$, called *exact r-transversal* with respect to γ. The sequences constructed analogously by means of left coset representatives are called *exact ℓ-transversals*. Similarly, we can construct *exact mixed transversals*, for which each block is either a complete sequence of left or right coset representatives of a quotient in the chain. Denote by $\mathcal{E}(\mathcal{G})$ the set of all exact ℓ-, r-, or mixed transversal logarithmic signatures for $\mathcal{G}$.

Definition 2.4.1 *A logarithmic signature α for a finite group $\mathcal{G}$ is called*

(i) **transversal***, if α is equivalent to a logarithmic signature of the same type in $\mathcal{E}(\mathcal{G})$;*

(ii) **non-transversal***, if it is not transversal;*

(iii) **totally non-transversal***, if none of its blocks is a coset of a non-trivial subgroup of $\mathcal{G}$.*

Transversal logarithmic signatures are an important example of tame logarithmic signatures. Figure 2.4 shows a transversal logarithmic signature α for $\mathfrak{S}_5$. The complete classification of logarithmic signatures can be found in [Cus00].

Let $\tau : \mathbb{Z}_{r_1} \oplus \cdots \oplus \mathbb{Z}_{r_s} \to \mathbb{Z}_m$ be a bijection defined as in Definition 2.2.1. Let $\gamma : 1_{\mathcal{G}} = \mathcal{G}_0 < \mathcal{G}_1 < \cdots < \mathcal{G}_s = \mathcal{G}$ be a chain of subgroups and $\alpha = [A_1, \ldots, A_s]$ a transversal logarithmic signature of type $(r_1, \ldots, r_s)$ constructed with respect to it. There exists a polynomial time algorithm for factorizing any element $g \in \mathcal{G}$ with respect to transversal logarithmic signature α.

Algorithm 2 Factorization with respect to transversal logarithmic signature

Input: $\alpha := (a_{i,j})$ transversal with respect to γ, function τ as defined above, $g \in \mathcal{G}$

Output: $x = \tau(j_1, j_2, \ldots, j_s) \in \mathbb{Z}_m$ such that $g = a_{1,j_1} a_{2,j_2} \cdots a_{s,j_s}$, $j_i \in \{1, \ldots, r_i\}$

```
1: $b \leftarrow g$
2: for $i \leftarrow s$ downto $1$ do
3:     Find $\ell \in \{1, \ldots, r_i\}$ such that $b \cdot a_{i,\ell}^{-1} \in \mathcal{G}_{i-1}$
4:     $j_i \leftarrow \ell$
5:     $b \leftarrow b \cdot a_{i,\ell}^{-1}$
6: end for
7: return $x \leftarrow \tau(j_1, \ldots, j_s)$
```

Starting from the block s, we search an element $a_{s,\ell} \in A_s$ such that $b \cdot a_{s,\ell}^{-1} \in \mathcal{G}_{s-1}$ where b is initialized as g. The index ℓ then equals to the correct pointer j_s. We multiply b by a_{s,j_s}^{-1} and continue with the block $s-1$. Note that in each cycle i, blocks $[A_1, \ldots, A_{i-1}]$ form a transversal logarithmic signature for subgroup $\mathcal{G}_{i-1}$ and that element b belongs to the coset $G_{i-1} \cdot a_{i,j_i}$, i.e. $b \cdot a_{i,\ell}^{-1} \in G_{i-1} \Leftrightarrow \ell = j_i$. As each block A_i consists of all (right) coset representatives of $\mathcal{G}_{i-1}$ in $\mathcal{G}_i$, such ℓ must exist. There are at most r_i choices for ℓ, and determining membership in a permutation group can be tested in time polynomial in the degree and the number of generators [FHL80].

Proposition 2.4.1 *Let α be a transversal logarithmic signature for an abelian group $\mathcal{G}$, and β is created by a block permutation of α. Then β is tame.*

Sketch of Proof: Let ξ be a permutation used to shuffle the blocks of α. Clearly, knowing ξ we only need to permute the order in which the pointers are recovered in Algorithm 2, i.e. going from $\xi(s)$ to $\xi(1)$, in each cycle $\xi(i)$ we search for $a_{\xi(i),\ell} \in A_{\xi(i)}$ such that $b \cdot a_{\xi(i),\ell}^{-1} \in \mathcal{G}_{i-1}$. Then $\ell = j_{\xi(i)}$. □

Here we present an algorithm which can determine in polynomial time whether a logarithmic signature α for a permutation group $\mathcal{G}$ is transversal or not (see also [MM89a]).

Algorithm 3 Algorithm to determine if the logaritmic signature is transversal

External: Algorithm 1 for construction of equivalent normalized cover

Input: Logaritmic signature $\alpha = [A_1, \ldots, A_s]$ for $\mathcal{G}$ of type $(r_1, \ldots, r_s)$

Output: **true** if α is transversal, **false** elsewhere

```
 1: if $\prod_{i=1}^{s} r_i \neq |\mathcal{G}|$ then
 2:     return false
 3: end if
 4: Use Algorithm 1 to construct normalized $\beta = [B_1, \ldots, B_s]$ equivalent to $\alpha$.
 5: $N \leftarrow 1$
 6: for $i \leftarrow 1$ to $s$ do
 7:     $N \leftarrow N \cdot r_i$
 8:     if $| < B_1, \ldots, B_i > | \neq N$ then
 9:         return false
10:     end if
11: end for
12: return true
```

First we verify that $\prod_{i=1}^{s} r_i = |\mathcal{G}|$, i.e. α is a pseudo logarithmic signature. Afterwards Algorithm 1 is used to create normalized β equivalent to α. For each $i = 1, \ldots, s$, we then check the order of the group generated by $< B_1, \ldots, B_i >$. Starting from the block B_1, we first check that $B_1 = \mathcal{G}_1$, i.e. B_1 is a subgroup. This takes polynomial time by using [FHL80] (or simply by testing the closure). Now, for each $i = 2, \ldots, s$ we build "strong generators" for $< B_1, \ldots, B_i >$ using $B_1 \cup \cdots \cup B_i$ and verify that the order of $< B_1, \ldots, B_i >$ is equal to $N = \prod_{j=1}^{i} r_j$. Both s and the time taken in the i^{th} step are bounded by a polynomial in the degree of $\mathcal{G}$.

Figure 2.5: A non-transversal logarithmic signature for $\mathfrak{A}_5$.

id
(2 3 4)
(2 4 3)
(1 2)(3 4)
(1 2 3 4 5)
(1 2 4)
(1 2 5 4 3)
(1 3 2)
(1 3 4)
(1 3 2 4 5)
(1 3)(2 5)
(1 4 2)
(1 4 5)
(1 4)(2 3)
(1 4 2 5 3)
id
(2 4)(3 5)
(2 5 4)
(2 5)(3 4)

Remark 2.4.1 Consider the following problem.

Question: Let β be a transversal logarithmic signature for an abelian group $\mathcal{G}$. Is the signature β' created by the fusion of blocks of β transversal?

Let $\mathcal{G}$ be abelian and let $\beta = [B_1, B_2 \ldots, B_s]$ be a transversal logarithmic signature constructed with respect to a chain of subgroups $\gamma : 1_{\mathcal{G}} = \mathcal{G}_0 < \mathcal{G}_1 < \mathcal{G}_2 < \cdots < \mathcal{G}_s$. W.l.o.g., assume that block B_1 consists of the complete set of right coset representatives of $\mathcal{G}_0$ in $\mathcal{G}_1$.

Then B_1 is a subgroup of $\mathcal{G}$. We construct a signature β' by the fusion of block B_1 with another block, different from B_2. In β' no block is a subgroup of $\mathcal{G}$, so it is non-transversal.

An Example

Let $\mathcal{G}$ be an abelian group of order 60 defined by the following polycyclic presentation

$$\mathcal{G} := \mathsf{Pc}\langle\, x_1, x_2, x_3, x_4 \mid x_1^2 = x_2,\; x_2^2 = x_3^3 = x_4^5 = 1 \,\rangle.$$

Let $\alpha' = [A_1', A_2', A_3']$ be a transversal logarithmic signature for $\mathcal{G}$ of type $(4, 3, 5)$. Fusion of A_1' and A_2' creates transversal signature β' of type $(12, 5)$. If we fuse A_1' and A_3', the resulting signature γ' of type $(20, 3)$ is non-transversal.

Figure 2.6: Fusion of blocks of transversal logarithmic signature for an abelian group $\mathcal{G}$.

β'

x_1
x_1x_3
$x_1x_2x_3^2$
x_1x_2
$x_1x_2x_3$
$x_1x_3^2$
x_2
x_2x_3
x_3^2
1
x_3
$x_2x_3^2$

x_1x_3
$x_1x_3x_4$
$x_1x_4^2$
$x_2x_3^2x_4^3$
$x_3x_4^4$

$\longleftarrow$

α'

A_1'	1
	x_2
	x_1
	x_1x_2
A_2'	x_1
	x_1x_3
	$x_1x_2x_3^2$
A_3'	x_1x_3
	$x_1x_3x_4$
	$x_1x_4^2$
	$x_2x_3^2x_4^3$
	$x_3x_4^4$

$\longrightarrow$

γ'

x_1x_3
$x_1x_3x_4$
$x_1x_4^2$
$x_2x_3^2x_4^3$
$x_3x_4^4$
$x_1x_2x_3$
$x_1x_2x_3x_4$
$x_1x_2x_4^2$
$x_3^2x_4^3$
$x_2x_3x_4^4$
x_2x_3
$x_2x_3x_4$
$x_2x_4^2$
$x_1x_2x_3^2x_4^3$
$x_1x_3x_4^4$
x_3
x_3x_4
x_4^2
$x_1x_3^2x_4^3$
$x_1x_2x_3x_4^4$

x_1
x_1x_3
$x_1x_2x_3^2$

Note that if $\mathcal{G}$ is non-abelian, the result after fusing two consecutive blocks is transversal. If two non-consecutive blocks are fused, the product will in general not be a logarithmic signature, but only a cover for a subset of $\mathcal{G}$.

An Example

Let $\alpha := [A_1, A_2, A_3]$ be a transversal logarithmic signature of type $(3, 4, 5)$ for Alternating group $\mathfrak{A}_5$ of order 60. Fusing A_1 and A_2 results in transversal β of type $(12, 5)$. If we however fuse A_1 with A_3, the result is cover γ of type $(15, 4)$ for a subset of $\mathfrak{A}_5$ of size 36. Let $\gamma := (h_{i,j})$, it is easy to verify that for example

$$h_{1,6} \cdot h_{2,1} = (2\ 3\ 4) \cdot \mathsf{id} = h_{1,1} \cdot h_{2,2} = (1\ 3)(2\ 4) \cdot (1\ 4\ 3),$$

i.e. element $g = (2\ 3\ 4)$ has two factorizations. Therefore γ is not a logarithmic signature.

Figure 2.7: Fusion of blocks of a transversal logarithmic signature for (non-abelian) group $\mathfrak{A}_5$.

β $\longleftarrow$ α $\longrightarrow$ γ

β
id
(1 4 3)
(2 3 4)
(1 4)(2 3)
(1 3 2)
(2 4 3)
(1 4 2)
(1 2 4)
(1 2 3)
(1 2)(3 4)
(1 3)(2 4)
(1 3 4)
(1 3)(2 4)
(1 5 4 2 3)
(1 4)(3 5)
(1 5 2 4 3)
(1 5)(3 4)

	α
A_1	id
	(1 3 2)
	(1 2 3)
A_2	id
	(1 4 3)
	(2 3 4)
	(1 4)(2 3)
A_3	(1 3)(2 4)
	(1 5 4 2 3)
	(1 4)(3 5)
	(1 5 2 4 3)
	(1 5)(3 4)

γ
(1 3)(2 4)
(1 5 4 2 3)
(1 4)(3 5)
(1 5 2 4 3)
(1 5)(3 4)
(2 3 4)
(2 5 4)
(1 5 3 2 4)
(2 5)(3 4)
(1 4 3 2 5)
(1 4 2)
(1 3 5 4 2)
(1 2 5 3 4)
(1 4 3 5 2)
(1 2 4 3 5)
id
(1 4 3)
(2 3 4)
(1 4)(2 3)

2.5 Cryptosystems Based on Logarithmic Signatures and Covers

In this section, we present a short overview of cryptosystems based on covers and logarithmic signatures for finite groups. The problem is still largely open.

The first scheme is a *symmetric* cryptosytem based on transversal logarithmic signatures for finite permutation group $\mathcal{G}$ where the shared private key induces a permutation on the message space $\mathbb{Z}_{|\mathcal{G}|}$.

2.5.1 PGM

A cryptographic system called *Permutation Group Mappings* (PGM), was invented in the late 1970's by S. Magliveras [Mag86, MM89a, MM92]. PGM is a private-key, endomorphic cryptosystem based on logarithmic signatures for finite permutation groups.

Let $[\alpha, \beta]$ be a pair of transversal logarithmic signatures for a permutation group $\mathcal{G}$. The *encryption* transformation $\mathsf{E}_{\alpha,\beta} : \mathbb{Z}_{|\mathcal{G}|} \to \mathbb{Z}_{|\mathcal{G}|}$ is defined by

$$\mathsf{E}_{\alpha,\beta} := \breve{\alpha} \circ \breve{\beta}^{-1}$$

The corresponding *decryption* transformation is obtained by

$$\mathsf{D}_{\alpha,\beta} := \mathsf{E}_{\alpha,\beta}^{-1} = \mathsf{E}_{\beta,\alpha} = \breve{\beta} \circ \breve{\alpha}^{-1}$$

As both logarithmic signatures α and β are tame, they have to be kept secret. The operations with respect to transversal logarithmic signatures are done efficiently, therefore the encryption and decryption are efficiently computable.

Denote the set of all transversal signatures of $\mathcal{G}$ with respect of a chain γ by $\Lambda(\gamma)$. The key space of PGM is $\Lambda \times \Lambda$, the collection of all ordered pairs of transversal logarithmic signatures of $\mathcal{G}$. We denote by $\mathcal{T}_{\mathcal{G}}$, the set of transformations defined by the key space and by $\mathsf{G}_{\mathcal{G}}$ the group generated by $\mathcal{T}_{\mathcal{G}}$ under functional composition.

The authors of [MM92] show that $\mathcal{T}_{\mathcal{G}}$ is not closed under functional composition and hence not a group. Moreover, they show that $\mathsf{G}_{\mathcal{G}} = \langle \mathcal{T}_{\mathcal{G}} \rangle$ is nearly always the symmetric group $\mathfrak{S}_{|\mathcal{G}|}$.

Theorem 2.5.1 (Magliveras–Memon [MM92]) *If $\mathcal{G}$ is a finite non-hamiltonian group with $|\mathcal{G}|$ different from* $q, (1+q^2), (1+q^3), \frac{(q^n-1)}{(q-1)}, 2^{n-1}(2^n \pm 1), 11, 12, 15, 22, 23, 24, 176, 276$, *where* q *is the power of a prime and* n *is a positive integer, then* $\mathcal{T}_{\mathcal{G}}$ *is 2-transitive and* $G_{\mathcal{G}} \cong \mathfrak{S}_{|\mathcal{G}|}$.

Later, Caranti and Volta published the following result.

Theorem 2.5.2 (Caranti–Volta [CV06]) *Let $\mathcal{G}$ be a nontrivial finite group. Suppose $\mathcal{G}$ is not cyclic of order a prime, or the square of a prime. Then the group $\langle \mathcal{T}_{\mathcal{G}} \rangle$ generated by $\mathcal{T}_{\mathcal{G}}$ is the full symmetric group $\mathfrak{S}_{|\mathcal{G}|}$.*

As a consequence of Theorems 2.5.1 and 2.5.2 we obtain the following theorem.

Theorem 2.5.3 *Let $\mathcal{G}$ be a nontrivial finite group wich is not cyclic of order a prime, or the square of a prime. Then any permutation in $\mathfrak{S}_{|\mathcal{G}|}$ can be written as a product of a finite number of transversal logarithmic signatures for $\mathcal{G}$.*

Besides the application in symmetric-key cryptography, the encryption function $E_{\alpha,\beta}$ of PGM has also been used to build a random number generator [MOS84].

Later in [MST02], Magliveras et al. explored the possibility of using logarithmic signatures and their generalizations to build *asymmetric* encryption schemes. They designed two public key cryptosystems, MST_1 by contructing trapdoor-permutations using logarithmic signatures for finite non-abelian groups, and MST_2 as a generalization of ElGamal encryption for non-abelian groups where the trapdoor one-way functions are induced by $[s, r]$-meshes.

Remark 2.5.1 Suppose that α is wild, then $\breve{\alpha} \circ \breve{\beta}^{-1}$ is efficiently computable, while $\breve{\beta} \circ \breve{\alpha}^{-1}$ is not and therefore gives rise to one-way function. However, if there exist transformations which map wild logarithmic signature α to a tame one, then we would have a trap-door which allows to invert $E_{\alpha,\beta}$ efficiently and we could build a public-key scheme.

2.5.2 MST_1

The first potential public key cryptosystem based on logarithmic signatures, called MST_1[2], has been introduced in [MST02]. Based on Theorem 2.5.3 we have, in particular, that each wild logarithmic signature can be written as a finite product of transversal logarithmic signatures. This fact leads to the designing of MST_1.

Let η be a fixed tame logarithmic signature. Then for any logarithmic signature α we define permutation $\hat{\alpha} := \breve{\alpha} \circ \breve{\eta}^{-1} \in \mathfrak{S}_{|\mathcal{G}|}$. Before we discuss MST_1 we have to make the following assumptions.

Assumption 2.5.1

(1) Wild logarithmic signatures induce one way functions, i.e. it is infeasible[3] to compute $\breve{\alpha}^{-1}$ for a given wild logarithmic signature α.

(2) Given a wild logarithmic signature α it is infeasible to find a set of transversal logarithmic signatures $\theta_1, \dots, \theta_k$ such that $\hat{\alpha} = \hat{\theta}_1 \circ \cdots \circ \hat{\theta}_k$.

The security of MST_1 relies on the hardness of these problems. Although there is no known proof, there is strong evidence that they are valid.

Let α be a wild and β a tame logarithmic signature for a finite permutation group $\mathcal{G}$. Then the mapping $\breve{\alpha} \circ \breve{\beta}^{-1} : \mathbb{Z}_{|\mathcal{G}|} \to \mathbb{Z}_{|\mathcal{G}|}$ is a one-way permutation in $\mathfrak{S}_{|\mathcal{G}|}$. However, if $\breve{\alpha} \circ \breve{\beta}^{-1}$ is written as a product of finite (hopefully small) number of transversal logarithmic signatures, it can be inverted efficiently. Let $\theta_1, \dots, \theta_k$ be a set of tame logaritmic signatures such that $\hat{\alpha} \circ \hat{\beta}^{-1} = \breve{\alpha} \circ \breve{\beta}^{-1} = \hat{\theta}_1 \circ \cdots \circ \hat{\theta}_k$. Alice publishes $[\alpha, \beta]$ and $\mathcal{G}$ as her public key, but keeps $[\theta_1, \dots, \theta_k]$ secret. The *encryption* transformation $E_{\alpha,\beta} : \mathbb{Z}_{|\mathcal{G}|} \to \mathbb{Z}_{|\mathcal{G}|}$ is defined by

$$E_{\alpha,\beta} := \breve{\alpha} \circ \breve{\beta}^{-1}$$

The encryption is efficiently computable as β is tame. The decryption function for the system is defined by

$$D_{\alpha,\beta} := \hat{\theta}_k^{-1} \circ \cdots \circ \hat{\theta}_1^{-1}$$

The decryption is computable only if the factorization $\breve{\alpha} \circ \breve{\beta}^{-1} = \hat{\theta}_1 \circ \cdots \circ \hat{\theta}_k$ is known. Since all logarithmic signatures θ_i are tame it is efficient too.

Although in practice, there is no known efficient algorithm how to construct these factorizations, the previous result shows that they do exist. By Assumption 2.5.1 (2) it is assumed that the factorization of $\breve{\alpha} \circ \breve{\beta}^{-1}$ is infeasible.

[2] The name has been constructed from three initials of the authors: Magliveras, Stinson, Tran van Trung.
[3] Its cost as measured by either the amount of memory used or by the runtime required is finite but impossibly large.

It turns out that the smallest group for which there exist non-transversal logarithmic signatures is the cyclic group $\mathbb{Z}_8$. In particular there are 512 permutations of $\mathfrak{S}_8$ induced by transversal and 640 induced by non-transversal logarithmic signatures. Any one of them can be written as the product of, at most, three transversals (see [MST02]).

Some potential problems connected with the key generation have been discussed in [VS02, VRS03, BSVM05]. Unfortunately, it still remains unclear how to derive concrete instances of MST_1.

2.5.3 MST_2

The second potential public key cryptosystem MST_2 introduced in [MST02] uses special type of covers called $[s,r]$-meshes (as defined in Section 2.1). The cryptosystem MST_2 can be viewed as a generalization of the ElGamal cryptosystem [EG85] for non-abelian groups.

Let $\alpha = (a_{i,j})$ be an $[s,r]$-mesh for a sufficiently large permutation group $\mathcal{G}$. Let $\mathcal{H}$ be a second group, and $f : \mathcal{G} \to \mathcal{H}$ an epimorphism. Then $\beta = (b_{i,j})$, where $b_{i,j} = f(a_{i,j})$, is an $[s,r]$-mesh for $\mathcal{H}$. The $[\alpha,\beta]$ is public key, the mapping f is kept secret.

To encrypt a message $h \in \mathcal{H}$

(1) Choose a random integer $R \in \mathbb{Z}_{r^s}$

(2) Compute values
$y_1 = \breve{\alpha}(R)$
$y_2 = h \cdot \breve{\beta}(R)$

The pair $y = (y_1, y_2)$ is the ciphertext for the message h.

To decrypt the message

(1) Knowing f compute $g = \breve{\beta}(R) = f(\breve{\alpha}(R)) = f(y_1)$

(2) Obtain the message $h = y_2 \cdot g^{-1}$

There are two known types of possible attacks against MST_2. The first attack tries to determine a random R such that $y_1 = \breve{\alpha}(R)$. Note that in general R is not unique, but finding any R' with $y_1 = \breve{\alpha}(R')$ constitutes breaking the system. Effectively, computing an R with $y_1 = \breve{\alpha}(R)$ means to factorize y_1 with respect to α, i.e. finding $(j_1, \dots, j_s)$ such that $y_1 = a_{1,j_1} \dots a_{s,j_s}$. The security of the system against this type of attack is based on the following assumption.

Assumption 2.5.2

(1) Given an $[s,r]$-mesh α for a group $\mathcal{G}$, and an element $g \in \mathcal{G}$, then finding a factorization $g = a_{1,j_1} \dots a_{s,j_s}$ is in general an intractable problem.

The second attack tries to find a homomorphism f' such that $\beta = f'(\alpha)$. Finding such an f' equals to breaking the private key. The security of the scheme in this case relies on the fact that given an arbitrary finite group $\mathcal{G}$ and a collection of elements $\{g_i\} \subset \mathcal{G}$ computing the intersection of centralizers in $\mathcal{G}$ of g_i is generally hard. However, for certain choices of the group $\mathcal{G}$, this problem can be solved in polynomial time. For example, in the case that the underlying group $\mathcal{G}$ is the symmetric group $\mathfrak{S}_n$, the resulting system MST_2 is not secure. There are no known practical implementations of MST_2.

2.5.4 MST_3

Most recently, in [LMTW09], the public key cryptosystem MST_3 was established on the basis of random covers and logarithmic signatures of non-abelian finite groups. The authors proposed a class of Suzuki 2-groups for a possible realization and proved its security under conceivable attacks. A large part of our research was to show that this realization could be implemented and used in practice. We prove new bound for the security, show the possible weaknesses and provide techniques to strenghten this system. Until now, the realization on the Suzuki 2-groups remains the only known realization of MST_3.

To the study of the realization of MST_3 using the Suzuki 2-groups we devote Chapters 4, 5 and 6.

2.6 Suzuki 2-groups

To begin with, we recall some basic facts about finite p-groups, where p denotes a prime number. A finite group $\mathcal{G}$ of order a power of p is called a p*-group*, i.e. $|\mathcal{G}| = p^n$ for a certain positive integer n. The least common multiple of the orders of the elements of $\mathcal{G}$ is called the *exponent* of $\mathcal{G}$. An abelian (commutative) p-group $\mathcal{G}$ of exponent p is called *elementary abelian* p-group. The set $Z(\mathcal{G}) = \{z \in \mathcal{G} : \; zg = gz, \; \forall g \in \mathcal{G}\}$ is called the *center* of $\mathcal{G}$. It is well-known that $Z(\mathcal{G})$ is a subgroup of order at least p for any p-group $\mathcal{G}$. The subgroup $\mathcal{G}'$ generated by all the elements of the form $x^{-1}y^{-1}xy$ with $x, y \in \mathcal{G}$ is called the *commutator subgroup* of $\mathcal{G}$. The so-called *Frattini* subgroup of $\mathcal{G}$ denoted $\Phi(\mathcal{G})$ is by definition the intersection of all the maximal subgroups of $\mathcal{G}$. If $\mathcal{G}$ is a p-group, the factor group $\mathcal{G}/\Phi(\mathcal{G})$ is elementary abelian. In particular, if $\mathcal{G}$ is a 2-group, $\Phi(\mathcal{G}) = < g^2 | g \in \mathcal{G} >$. Finally, an element of order 2 in a group is called an *involution*.

Formally a *Suzuki 2-group* is defined as a nonabelian 2-group with more than one involution, having a cyclic group of automorphisms which permutes its involutions transitively. This class of 2-groups was studied and characterized by G. Higman [Hig63]. In particular, in any Suzuki 2-group $\mathcal{G}$ we have $Z(\mathcal{G}) = \Phi(\mathcal{G}) = \mathcal{G}' = \Omega_1(\mathcal{G})$, where $\Omega_1(\mathcal{G}) = < g \in \mathcal{G} : g^2 = 1 >$ and $|Z(\mathcal{G})| = q = 2^m, \; m > 1$. It is shown in [Hig63] that the order of $\mathcal{G}$ is either q^2 or q^3.

Thus all the involutions of $\mathcal{G}$ are in the center of $\mathcal{G}$, therefore $Z(\mathcal{G})$ and the factor group $\mathcal{G}/\Phi(\mathcal{G})$ are elementary abelian. Consequently, all elements not in $Z(\mathcal{G})$ have order 4, i.e. $\mathcal{G}$ is of exponent 4. It is known that $\mathcal{G}$ has an automorphism ξ of order $q-1$ cyclically permuting the involutions of $\mathcal{G}$ (see [Hig63] and [HB82]).

In our realization of MST_3, we only consider the class of Suzuki 2-groups having order q^2. Using Higman's notation, a Suzuki 2-group of order q^2 will be denoted by $A(m,\theta)$. Let $q = 2^m$ with $3 \leqslant m \in \mathbb{N}$ such that the field $\mathbb{F}_q$ has a nontrivial automorphism θ of odd order. This implies that m is not a power of 2. The groups $A(m,\theta)$ can be defined as matrix groups.

In fact, if we define
$$\mathcal{G} := \{S(a,b) \mid a,b \in \mathbb{F}_q\},$$
where
$$S(a,b) = \begin{pmatrix} 1 & a & b \\ 0 & 1 & a^\theta \\ 0 & 0 & 1 \end{pmatrix}$$
is a 3×3 -matrix over $\mathbb{F}_q$, then it is shown that the group $\mathcal{G}$ is isomorphic to $A(m,\theta)$. Thus $\mathcal{G}$ has order q^2 and we have
$$\mathcal{Z} := Z(\mathcal{G}) = \Phi(\mathcal{G}) = \mathcal{G}' = \Omega_1(\mathcal{G}) = \{S(0,b) \mid b \in \mathbb{F}_q\}.$$

As the center $Z(\mathcal{G})$ is elementary abelian of order q, it can be identified with the additive group of the field $\mathbb{F}_q$. Also the factor group $\mathcal{G}/\Phi(\mathcal{G})$ is an elementary abelian group of order q. It is then easily verified that the multiplication of two elements in $\mathcal{G}$ is given by the rule:
$$S(a_1,b_1)S(a_2,b_2) = S(a_1 + a_2\ ,\ b_1 + b_2 + a_1 a_2^\theta). \tag{2.6.1}$$

In this matrix form representation, the Suzuki 2-groups $A(m,\theta)$ can be considered as subgroups of the *general linear* group $GL(3,q)$ over $\mathbb{F}_q$.

It has been shown in [Hig63] that the groups $A(m,\theta)$ and $A(m,\phi)$ are isomorphic if and only if $\phi = \theta^{\pm 1}$.

For any $0 \neq \lambda \in \mathbb{F}_q$ the matrix
$$\Lambda = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \lambda & 0 \\ 0 & 0 & \lambda^{\theta+1} \end{pmatrix}$$
induces an automorphism of $A(m,\theta)$. And Λ acts on $A(m,\theta)$ according to the rule
$$\Lambda^{-1}S(a,b)\Lambda = S(a\lambda, b\lambda^{\theta+1}).$$

If $\lambda = \phi$ is a primitive element in $\mathbb{F}_q$, then Λ has order $q-1$ and permutes cyclically the $q-1$ involutions in the center of $A(m,\theta)$.

2.7 Summary

This chapter provides a brief introduction to covers and logarithmic signatures and their induced mappings. We define classes of transversal and non-transversal logarithmic signatures and provide some essential algorithms related to their use. We also give an overview of transformations that can apply to covers. In particular, the operation of fusing blocks is underscored as it allows to transform transversal logarithmic signatures for an abelian groups to a non-transversal ones. We briefly present some approaches in designing cryptosystems based on random covers and logarithmic signatures of finite groups. At the end, we give a description of the Suzuki 2-groups proposed for the known realization of the cryptosystem MST_3.

Chapter 3

Generation of Random Covers for Finite Groups

Before we could build any cryptosystem based on [s,r]-meshes or covers in general, we require an efficient way of constructing these structures in practice. This chapter is devoted to the study of the problem of generating covers and uniform covers of large finite groups.

3.1 Uniform Random Cover

Let $\mathcal{G}$ be a finite abstract group, and $\mathcal{S}$ be a subset of $\mathcal{G}$. Let $\alpha = [A_1, A_2, \ldots, A_s]$ be a cover of type $(r_1, \ldots, r_s)$ for $\mathcal{G}$ (or $\mathcal{S}$), i.e. for every element $g \in \mathcal{G}$ (resp. $g \in \mathcal{S}$) there are elements $a_{i,j_i} \in A_i$ such that

$$g = a_{1,j_1} a_{2,j_2} \ldots a_{s,j_s} \tag{3.1.1}$$

Let λ_g denote the number of ways for which an element $g \in \mathcal{G}$ has a representation given in Equation 3.1.1. Let $\lambda_{min} = \min\{\lambda_g : g \in \mathcal{G}\}$ and $\lambda_{max} = \max\{\lambda_g : g \in \mathcal{G}\}$. The ratio $\lambda := \lambda_{max}/\lambda_{min} \geqslant 1$ measures the degree of uniformity of α. A cover α is *uniform* if $\lambda \leqslant 2$. We note here that the reason for taking $\lambda \leqslant 2$ as bound for uniform covers is that we want to include the case of 1-*quasi logarithmic signatures* [MST02], for which $\lambda_{min} = 1$ and $\lambda_{max} = 2$. If however the value of λ_{min} is large, we shall obviously expect that the ratio λ is much smaller than 2, namely close to 1, in the above definition.

3.2 New Bound for Random Cover

Assume that we are given a collection $\alpha = [A_1, \ldots, A_s]$ of random subsets A_i of a group $\mathcal{G}$. We want to determine the probability, proving the "covering property" for α. It should be noted that in a real cryptographic application, the order of $\mathcal{G}$ is very large. Hence, a direct inspection of the covering property of α by running through all elements of $\mathcal{G}$ is obviously impossible.

In what follows, we show that the issue is strictly related to the well known classical occupancy problem (see e.g. [Fel57]), and can therefore be completely solved.

Let $n = |\mathcal{G}|$ be the order of $\mathcal{G}$ and let $\alpha = [A_1, \ldots, A_s]$ be an ordered collection of random subsets A_i of $\mathcal{G}$, i.e. each element of A_i is chosen with probability $1/n$. Let $N = r_1 \ldots r_s$, where $r_i = |A_i|$. The elements of A_i, $i = 1, \ldots, r$, can be interpreted as a set of random chosen elements from $\mathcal{G}$ with replacement. The set of all elements $g \in \mathcal{G}$ which can be expressed by Equation 3.1.1 is said to be created by α. Note that g can be written in more than one way by this equation. As with the elements of α, we may assume that the set of elements created by α is thus a random set of elements of $\mathcal{G}$. We assign each of n *cells* $1, 2, \ldots, n$ to each of n elements $g_1, g_2, \ldots g_n$ of $\mathcal{G}$. An element $g_i \in \mathcal{G}$ of the form $g_i = a_{1,j_1}.a_{2,j_2} \ldots a_{s,j_s}$ is interpreted as a *ball* in cell i. Thus the problem becomes that of a random distribution of N balls (N elements generated by α) in n cells, where each arrangement has probability n^{-N}. Let $E_{j_1,j_2,\ldots j_m}$ be the event that elements $g_{j_1},\ g_{j_2},\ \ldots,\ g_{j_m} \in \mathcal{G}$ are not created by α, i.e. $E_{j_1,j_2,\ldots j_m}$ be the event that cells $j_1, j_2, \ldots j_m$ are empty. In this event all N balls are placed in the remaining $n-m$ cells, and this can be done in $(n-m)^N$ different ways. Thus $p_{j_1,j_2,\ldots j_m} = (1 - \frac{m}{n})^N$ is the probability of event $E_{j_1,j_2,\ldots j_m}$. Set

$$T_m := \binom{n}{m}\left(1 - \frac{m}{n}\right)^N.$$

The method of inclusion and exlusion shows that the probability that at least one cell is empty equals

$$\sum_{i=1}^{n}(-1)^{i-1}T_i = \sum_{i=1}^{n}(-1)^{i-1}\binom{n}{i}\left(1 - \frac{i}{n}\right)^N.$$

Let $p_m(N, n)$ denote the probability that exactly m cells remain empty. Then the probability that all elements of $\mathcal{G}$ are covered by α (i.e. no cell is empty) is $p_0(N, n)$ and we have

$$p_0(N, n) = \sum_{j=0}^{n}(-1)^j\binom{n}{j}\left(1 - \frac{j}{n}\right)^N \tag{3.2.1}$$

Consider now $p_m(N, n)$. Since m cells can be chosen in $\binom{n}{m}$ ways and since each of the remaining $n - m$ cells is occupied, the number of patterns of these distributions is

$(n-m)^N p_0(N, n-m)$. Dividing by n^N we obtain $p_m(N,n)$. Thus

$$p_m(N,n) = \binom{n}{m} \sum_{j=0}^{n-m} (-1)^j \binom{n-m}{j} \left(1 - \frac{m+j}{n}\right)^N.$$

Define $\mu := ne^{-\frac{N}{n}}$. It has been shown (see [Fel57]) that if N, $n \to \infty$ but μ remains bounded, then

$$p_m(N,n) - e^{-\mu}\frac{\mu^m}{m!} \to 0$$

for each fixed m. Hence we have

$$p_m(N,n) \approx e^{-\mu}\frac{\mu^m}{m!}$$

for large n. In particular,

$$p_0(N,n) \approx e^{-\mu}.$$

This implies that for any given value $0 < \nu < 1$ there is an $N_0 \in \mathbb{Z}_+$ such that for any $N \geqslant N_0$ random covers of type $(r_1, \ldots, r_s)$ with $r_1 \ldots r_s = N$ can be generated with probability $p_0(N,n) \geqslant \nu$. This means that we can choose N so that $1 - p_0(N,n)$ is close to 0.

Thus we have the following theorem.

Theorem 3.2.1 *Let $\mathcal{G}$ be a finite group with $|\mathcal{G}| = n$. For any given value $0 < \nu < 1$ there is an $N_0 \in \mathbb{Z}_+$ such that any collection $\alpha = [A_1, \ldots, A_s]$ of random subsets A_i of $\mathcal{G}$ with $N = |A_1| \times \ldots \times |A_s| \geqslant N_0$ is a cover of $\mathcal{G}$ with a probability $p_0(N,n) \geqslant \nu$. Moreover, for large n we have*

$$p_0(N,n) \approx e^{-\mu}, \ \mu = ne^{-\theta},$$

where $\theta := \frac{N}{n}$.

Experimental results in the next section show that even with moderate values of n the error of the approximation of $p_0(N,n)$ in Theorem 3.2.1 is small.

3.3 Experimental results for generating random covers

We present the experimental results with the alternating groups $\mathfrak{A}_8$, $\mathfrak{A}_9$ and $\mathfrak{A}_{10}$. For each group $\mathcal{G} = \mathfrak{A}_i$ and for each *test* we randomly generate 10000 collections $\alpha = [A_1, \ldots, A_s]$ of subsets of $\mathcal{G}$ of a certain type $(r_1, \ldots, r_s)$, where $r_i = |A_i|$, and then count the number of elements of $\mathcal{G}$ covered by α. We repeat the test for several types of α eventually obtaining the probabilities of the covering of α, i.e. α being a cover. The results show that these probabilities are almost identical with those of the theoretical results in Theorem 3.2.1.

Table 3.1: Experimental results for generating random covers for small Alternating groups.

					theoretical [%]				experimental [%][1]			
$\mathcal{G}$	s	ℓ	N	θ	P_0	P_1	P_2	P_3	P_0	P_1	P_2	P_3
$\mathfrak{A}_8$	8	38	$5^6 \cdot 4^2$	12.4	92.0	7.6	0.4	0	90.5	8.6	0.8	0.1
		39	$5^7 \cdot 4$	15.5	99.6	0.4	0	0	99.6	0.4	0	0
		40	5^8	19.4	100	0	0	0	100	0	0	0
		41	$6 \cdot 5^7$	23.3	100	0	0	0	100	0	0	0
		42	$6^2 \cdot 5^6$	27.9	100	0	0	0	100	0	0	0
$\mathfrak{A}_9$	9	47	$6^2 \cdot 5^7$	15.5	96.7	3.2	0.1	0	97.0	3.0	0	0
		48	$6^3 \cdot 5^6$	18.6	99.8	0.2	0	0	100	0	0	0
		49	$6^4 \cdot 5^5$	22.3	100	0	0	0	100	0	0	0
		50	$6^5 \cdot 5^4$	26.8	100	0	0	0	100	0	0	0
$\mathfrak{A}_{10}$	10	56	$6^6 \cdot 5^4$	16.1	82.7	15.7	1.5	0.1	83.7	14.7	1.6	0
		57	$6^7 \cdot 5^3$	19.3	99.2	0.8	0	0	100	0	0	0
		58	$6^8 \cdot 5^2$	23.1	100	0	0	0	99.3	0.7	0	0
		59	$6^9 \cdot 5$	27.8	100	0	0	0	100	0	0	0

Each line of Table 3.1 presents a test with 10000 random collections of α. The theoretical bounds for the probabilities $P_m := p_m(N, n)$ and the corresponding test results from the experiment, also denoted by P_m, are in the second and third column respectively. We are mainly interested in P_0 given in Theorem 3.2.1. For instance, the second line of the table shows that $\mathcal{G} = \mathfrak{A}_8$, $s = 8$, $r_1 = \cdots = r_7 = 5$ and $r_8 = 4$, i.e. $\ell = r_1 + \cdots + r_8 = 39$ and $N = r_1 \times \cdots \times r_8 = 5^7.4$ and $\theta = \frac{N}{n} = 15.5$, where $n = |\mathfrak{A}_8| = \frac{1}{2}8! = 20160$.

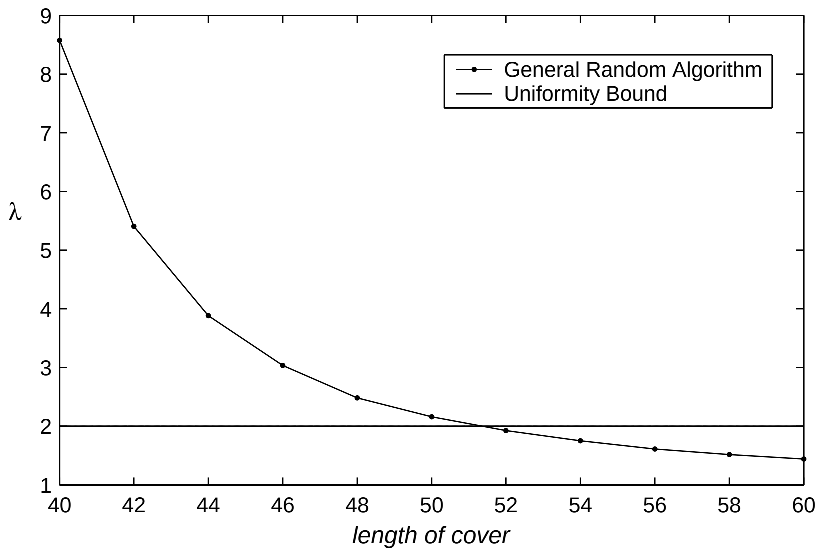
3.4 Random generating covers

The value $\lambda = \lambda_{max}/\lambda_{min} \geqslant 1$ measures the degree of uniformity of a cover. Our further experiment shows that the values of λ decrease and tend to 1 when the lengths $\ell = r_1 + \cdots + r_s$ of the covers increase, i.e. the uniformity of the covers increases with their lengths ℓ. We conjecture that this fact is true in general. This would imply that we could generate random covers with high degree of uniformity by increasing the lengths. It is therefore

[1] The experiment has been done with 10000 repeats for each type of cover.

interesting to find a formula expressing the degree of uniformity of covers with respect to their lengths. The diagram in the Figure 3.1 shows the degree of uniformity λ as a function of the length ℓ of the covers for the alternating group $\mathfrak{A}_8$. The experiment has been done with 1000 repeats for each length. The table shows for instance that random generated covers of length $\ell \geqslant 52$ for $\mathfrak{A}_8$ are uniform.

Figure 3.1: General random algorithm for covers of *specified length* for $\mathfrak{A}_8$.



3.5 Comparison of covers from a random algorithm and a greedy algorithm

As we have seen in the previous experiment, the uniformity of the covers increases with their lengths. A natural question emerges.

Question: For a given length ℓ can we construct random covers with a higher degree of uniformity than that of randomly generated covers?

An experiment has been made with a greedy algorithm for the groups $\mathfrak{A}_i$, $i = 5, 6, 7, 8$. The results in the table below show that improvements can indeed be obtained.

Algorithm 4 Greedy algorithm

```
Input: Random cover $\alpha := (a_{i,j})$ of type $(r_1, \ldots, r_s)$
Output: Improved cover
1: for $i \leftarrow 2$ to $s$ do
2:     for $j \leftarrow 2$ to $r_i$ do
3:         $a_{i,j} \leftarrow$ Find_best_element for position $(i, j)$
4:     end for
5: end for
```

where the function *Find_best_element* returns for position (i, j):

$$\begin{cases} \text{element which } \textit{maximizes product}, & \text{if group is not yet covered} \\ \text{element with } \textit{best uniformity product}, & \text{otherwise} \end{cases}$$

Table 3.2: Greedy algorithm compared to general random method for small Alternating groups.

$\mathcal{G}$	s	ℓ	N	θ	λ_{rand}	λ_{greedy} [2]	$\frac{\lambda_{greedy}}{\lambda_{rand}}$
$\mathfrak{A}_5$	5	20	4^5	17.1	2.66	1.64	0.62
$\mathfrak{A}_6$	6	27	$5^3 \cdot 4^3$	17.4	4.40	2.40	0.55
$\mathfrak{A}_7$	7	33	$5^5 \cdot 4^2$	18.6	6.56	3.85	0.53
$\mathfrak{A}_8$	8	40	5^8	19.4	8.49	5.25	0.62

s : number of blocks in cover
ℓ : length of cover
$N = r_1 \times \cdots \times r_s$
$\theta = N/n$

Finally, we have carried out two further experiments concerning the degree of uniformity for random covers for $\mathfrak{A}_8$. These include random covers generated with elements of specified orders and random covers generated with elements of specified distance to the identity. In both cases no improvement of the degree of uniformity has been obtained when compared with general generated random covers.

[2] The experiment has been done with $T = n/10$ (tries for each possition in cover) and 1000 repeats for each cover.

3.6 Number of representations

Let α be a random cover of type $(r_1, \dots, r_s)$ for a group $\mathcal{G}$. Denote by $\mathcal{S}$ the subset of $\mathcal{G}$ covered by α. An obvious question arrises here.

Question: What is the average number of representations for each element of $\mathcal{S}$ with respect to α?

Let $N = \prod_{i=1}^{s} r_i$. Consider the following scenario. Suppose we build a scheme where the security is based on the problem of factoring with respect to a randomly generated cover, which is assumed to be hard by the Cryptographic Hypothesis 1. Let α be a random cover for $\mathcal{S} \subset \mathcal{G}$ such that the ratio $\rho := N/|\mathcal{S}|$ is sufficiently large. Then the scheme remains secure even if the cryptographic hypothesis for α is removed. Clearly, within this scenario, finding a factorization with respect to α of given $y = \breve{\alpha}(x)$ provides only a small probability of retrieving the correct x, as the number of different factorizations is expected to be large.

As pointed out in Section 3.2, the covering property for α, and therefore the problem of estimating the size of $\mathcal{S}$ as well, is related to the occupancy problem. Interpreting elements of $\mathcal{G}$ as cells and all the products of the form $a_{1,j_1} \dots a_{s,j_s}$ as balls, the problem becomes that of the random distribution of $r_1 \dots r_s = N$ balls in $n = |\mathcal{G}|$ cells. The expected number of empty cells m is given by the formula

$$m = n\left(1 - \frac{1}{n}\right)^N$$

(see e.g. [Fel57]). This value can be approximated by $n \cdot e^{-\frac{N}{n}}$. We may then estimate the size of $\mathcal{S}$ as $n - m$, and approximate the quotient ρ by

$$\rho \approx \frac{N}{n - n \cdot e^{-\frac{N}{n}}} = \frac{\frac{N}{n}}{1 - e^{-\frac{N}{n}}}$$

For the case $N = n$ we become $e/(e-1) \approx 1.58$. In [VPD10], the authors present the data from a simple experiment evidencing that already for small parameters the ratio is very close to this value.

This result gives us an additional tool for protecting system based on random covers against factorization attacks, or helps us to prove their security in the cases where this ratio is chosen to be large (see e.g. Sections 5.4.1 and 7.4).

Another relevant result from the related occupancy problem provides an estimate for the maximum number of balls in any bin, i.e. maximum number of representations for any element in $\mathcal{S}$. This problem has been originally studied within the context of hashing functions, and for case $N = n$ has been estimated to be $\Theta(\frac{\log n}{\log \log n})$ [Mit96]. For the case $N \geqslant n \log n$ well known result says that the maximum load of any bin is $\Theta(\frac{N}{n})$, i.e. of the order of the mean.

3.7 Conclusions

This chapter provides an answer for the basic question that we must address before we start to use the covers in practice. How to efficiently generate covers?

We propose a method for doing this in, for cryptography, natural and highly efficient way, i.e. by choosing the elements of covers randomly. Obviously, as the order of the underlying group is very large, the direct checking of the covering property becomes impossible. We show the relation between this problem and the classical occupancy problem. As a consequence, we can estimate a new bound for the probability for which randomly chosen collection of elements forms a cover. For large $n = |\mathcal{G}|$ and random cover of type $(r_1, \ldots, r_s)$ this probability is approximated by

$$\lambda \left(\frac{e^{\lambda}}{e^{\lambda} - 1} \right)$$

where $\lambda = (r_1 \cdots r_s)/n$. We present results from computer experiment with several alternating groups showing that even with moderate values of n, the error of the approximation is rather small. In addition, we investigate the use of a greedy algorithm for generating covers with high degree of uniformity. The computer experiments show that improvements can indeed be obtained. The connection with the classical occupancy problem also helps us to estimate the average and maximal number of represenations for any element of the group with respect to a given random cover.

These results provide useful hints for generating covers with appropriate properties for groups of arbitrarily large order.

Chapter 4

First Realization of MST_3 on Suzuki 2-Groups

This chapter is devoted to the description of the cryptosystem MST_3 as published in [LMTW09]. The Suzuki 2-groups have been suggested for a possible realization of the generic version of MST_3. On one hand, due to their structure, the Suzuki 2-groups allow one to study the security of the system. On the other hand, they possess a simple presentation allowing an efficient implementation of the scheme. We present a detailed study of the security of this realization by exploiting properties of the group operation as well as specific properties of the transversal logarithmic signatures used here.

4.1 Generic Version of MST_3

The security basis of the system is formed by the cryptographic hypothesis that random covers induce one-way functions. We presently describe cryptosystem MST_3 in its generic form.

Let $\mathcal{G}$ be a finite non-abelian group with nontrivial center $\mathcal{Z}$. The group $\mathcal{G}$ should satisfy the following property:

Property 4.1.1 *$\mathcal{G}$ has a nontrivial center $\mathcal{Z}$ such that $\mathcal{G}$ does not split over $\mathcal{Z}$, i.e. there is no subgroup $\mathcal{H} < \mathcal{G}$ with $\mathcal{H} \cap \mathcal{Z} = 1$ such that $\mathcal{G} = \mathcal{Z} \cdot \mathcal{H}$.*

Moreover, we assume that the order of $\mathcal{Z}$ is sufficiently large so that exhaustive search problems are computationally infeasible in $\mathcal{Z}$.

Algorithm 5 Generic version of MST_3

Key generation

Input: A large group $\mathcal{G} = A(m, \theta)$, $q = 2^m$, as described above.

Output: A public key $[\alpha, \gamma]$ with corresponding private key $[\beta, (t_0, \dots, t_s)]$.

```
For all $i = 1, \dots, s;\ j = 1, \dots, r_i$ generate
  1: a *tame logarithmic signature* $\beta = [B_1, \dots, B_s] := (b_{i,j})$ of type $(r_1, \dots, r_s)$ for $\mathcal{Z}$
  2: a *random cover* $\alpha = [A_1, \dots, A_s] := (a_{i,j})$ of the same type as $\beta$ for a certain subset
     $\mathcal{S}$ of $\mathcal{G}$ such that $A_1, \dots, A_s \subseteq \mathcal{G} \setminus \mathcal{Z}$
Futher choose elements
  3: $t_0, t_1, \dots, t_s \in \mathcal{G} \setminus \mathcal{Z}$
And compute
  4: $\tilde{\alpha} = [\tilde{A}_1, \dots, \tilde{A}_s]$, where $\tilde{A}_i = t_{i-1}^{-1} A_i t_i$ for $i = 1, \dots, s$
  5: $\gamma := (h_{i,j}) = (b_{i,j} \tilde{a}_{i,j})$
Publish $[\alpha, \gamma]$, keep $[\beta, (t_0, \dots, t_s)]$ private.
```

Encryption

Input: A message $x \in \mathbb{Z}_{|\mathcal{Z}|}$ and the public key $[\alpha, \gamma]$.

Output: A ciphertext (y_1, y_2) for the message x.

```
  1: compute values
     $y_1 = \breve{\alpha}(x)$
     $y_2 = \breve{\gamma}(x) = t_0^{-1}\ \breve{\alpha}(x)\ \breve{\beta}(x)\ t_s$
```

Decryption

Input: A ciphertext pair (y_1, y_2) and the private key $[\beta, (t_0, \dots, t_s)]$.

Output: The message $x \in \mathbb{Z}_{|\mathcal{Z}|}$ that corresponds to the ciphertext (y_1, y_2).

```
  1: knowing  $y_2 = \breve{\gamma}(x)$
                 $= b_{1,j_1} \tilde{a}_{1,j_1} \cdot b_{2,j_2} \tilde{a}_{2,j_2} \cdots b_{s,j_s} \tilde{a}_{s,j_s}$
                 $= b_{1,j_1} t_0^{-1} a_{1,j_1} t_1 \cdots b_{s,j_s} t_{s-1}^{-1} a_{s,j_s} t_s$
                 $= b_{1,j_1} b_{2,j_2} \cdots b_{s,j_s} t_0^{-1} a_{1,j_1} a_{2,j_2} \cdots a_{s,j_s} t_s$
                 $= \breve{\beta}(x) \cdot t_0^{-1} \breve{\alpha}(x) t_s$
                 $= \breve{\beta}(x) \cdot t_0^{-1} y_1 t_s$
     compute  $\breve{\beta}(x) = y_2 t_s^{-1} y_1^{-1} t_0$
  2: recover $x$ from $\breve{\beta}(x)$ using $\breve{\beta}^{-1}$ which is efficiently computable as $\beta$ is tame
```

Before we introduce the first realization of MST_3 on Suzuki 2-groups, we present some basic facts about special type of logarithmic signatures used here.

4.2 Logarithmic Signatures for Elementary Abelian 2-Groups

An elementary abelian 2-group of order 2^m can be identified with a vector space of dimension m over the field $\mathbb{F}_2$. We focus on a special type of factorizable logarithmic signatures for vector spaces. These are used as a basis for key generation in the realization of MST_3 based on Suzuki 2-groups. We are particularly interested in vector spaces over the field $\mathbb{F}_2$.

Definition 4.2.1 *Let V be a vector space of dimension m over the finite field $\mathbb{F}_2$. Further, let $\mathcal{P} = C_1 \cup \cdots \cup C_s$, $|C_i| = k_i$, $\sum_{i=1}^{s} k_i = m$, be a random partition of the set $\{1, \ldots, m\}$. A logarithmic signature $\beta = [B_1, \ldots, B_s]$ for V is said to be* ***canonical*** *if for each $i \in \{1, \ldots, s\}$, block B_i consists of all possible 2^{k_i} vectors with bits set on the positions defined by the subset C_i and zeros elsewhere.*

A canonical signature β for V of the type $(r_1, r_2, \ldots, r_s)$, $r_i = 2^{k_i}$, can be created by the following algorithm:

Algorithm 6 Construction of a canonical logarithmic signature for V

```
1: Create a random partition $\mathcal{P} = C_1 \cup \cdots \cup C_s$ of the set $\{1, \ldots, m\}$ with $|C_i| = k_i$.
2: Now, for each $i \in \{1, \ldots, s\}$ create block $B_i$ of $\beta = [B_1, \ldots, B_s] := (b_{i,j})$ as follows.
   Take all possible $2^{k_i}$ vectors $u_j$ of dimension $k_i$, and position each such vector on the
   bits of $C_i$. Each time, construct the vector $b_{i,j}$ of dimension $m$ which has $u_j$ in the
   positions defined by the subset $C_i$ and zeros everywhere else.
```

Definition 4.2.2 *Let V be a vector space of dimension m over $\mathbb{F}_2$. We say that a canonical logarithmic signature $\beta = [B_1, \ldots, B_s] := (b_{i,j})$ for V is in* ***standard form****, if it also fulfils the following conditions:*

(i) $C_1 = \{1, \ldots, k_1\}$, $C_2 = \{k_1 + 1, \ldots, k_1 + k_2\}, \ldots,$ $C_s = \{k_1 + \cdots + k_{s-1} + 1, \ldots, m\}$ *(the lowest k_1 bits are used for block B_1, the next k_2 bits for B_2, etc.)*

(ii) for all i, $j_1 < j_2$: $b_{i,j_1} < b_{i,j_2}$ *(the vectors within B_i are sorted)* [1]

[1] $a < b \Leftrightarrow \mathtt{int}(a) < \mathtt{int}(b)$, where function $\mathtt{int}(x)$ returns the integer value of the vector x expressed with respect to Radix 2.

It is clear that β forms a logarithmic signature for V.

Figure 4.1 shows canonical logarithmic signature $\beta = [B_1, B_2, B_3]$ of type $(4, 8, 8)$ in standard form, i.e. $\mathcal{P} = \big\{\{1, 2\}, \{3, 4, 5\}, \{6, 7, 8\}\big\}$.

Figure 4.1: Canonical logarithmic signature for V of dimension 8 over $\mathbb{F}_2$ in standard form.

	β
	00 000 000
B_1	**01** 000 000
	10 000 000
	11 000 000
	00 **000** 000
B_2	00 **001** 000
	00 **010** 000
	00 **011** 000
	00 **100** 000
	00 **101** 000
	00 **110** 000
	00 **111** 000
	00 000 **000**
	00 000 **001**
B_3	00 000 **010**
	00 000 **011**
	00 000 **100**
	00 000 **101**
	00 000 **110**
	00 000 **111**

Proposition 4.2.1 *Canonical signatures are tame.*

Proof: From Definition 4.2.1, the elements of block B_i act only on the bits of C_i, and each B_i contains a complete set of 2^{k_i} vectors of dimension k_i on the positions of C_i. To "factorize" element $y \in V$ in the form $y = b_{1,j_1} b_{2,j_2} \ldots b_{s,j_s}$ we split the bits of y into vectors b_{i,j_i} with respect to partition $\mathcal{P}$ as follows. We copy the bits of y on the positions determined by C_i to appropriate vector b_{i,j_i}, and set the rest of the bits of b_{i,j_i} to zero. The position of such created vector b_{i,j_i} within the block B_i then defines index j_i.

If the vectors in each B_i are sorted in ascending order, then the integer value (with respect to Radix 2) of the subvector u_i constructed from b_{i,j_i} by concatenation of the bits on the positions of C_i, is equal to the index of b_{i,j_i} within the block B_i (starting with index 0). This factorization procedure has time complexity $\mathcal{O}(1)$. □

The following statement ensues naturally.

Proposition 4.2.2 *Transforming a canonical logarithmic signature of* V *by means of a non-singular linear transformation results in a tame logarithmic signature for* V.

Sketch of Proof: As this transformation is reversible, the resulting logarithmic signature is tame. □

Using this proposition we may construct tame logarithmic signatures.

Algorithm 7 Construction of tame logarithmic signature for V

```
1: Create a canonical logarithmic signature $\beta := (b_{i,j})$ of a given type for $V$ over the field
   $\mathbb{F}_{2^m}$ (using Algorithm 6).
2: Generate a random matrix $M \in GL(m, 2)$ and transform $\beta$ to a tame logarithmic
   signature $\beta^* := (b^*_{i,j}) = (b_{i,j}M)$.
```

The use of random matrices in Algorithm 7 for tame signature generation introduces some level of randomness essential for the cryptography. However, as shown in later sections, it does not prevent an attack that exploits the properties of canonical signatures used in this algorithm.

Proposition 4.2.3 *Let* $\beta = [B_1, B_2, \ldots, B_s]$ *be a canonical logarithmic signature for* V. *Let* $\beta^* := (b^*_{i,j})$*, where* $b^*_{i,j} = b_{i,j} d_i$ *with* $d_i \in B_i$*, then* β^* *is also canonical for* V.

Proof: From Definition 4.2.1, the blocks of β act on disjoint sets of bits C_i, and every block B_i contains the complete set of 2^{k_i} vectors with bits set on the positions of C_i. If we multiply all elements of B_i with a fixed element $d_i \in B_i$, we switch the bits in the positions of 1's in d_i in each of 2^{k_i} possible vectors, so B_i remains the same up to order, and β remains canonical for V. □

In general we have

Proposition 4.2.4 *Let* $\mathcal{G}$ *be a finite group. Let* $\beta = [B_1, B_2, \ldots, B_s] := (b_{i,j})$ *be a tame logarithmic signature for* $\mathcal{G}$. *Let* $\beta^* := (b^*_{i,j})$*, where* $b^*_{i,j} = b_{i,j} d_i$*,* $d_i \in \mathcal{G}$. *Then* β^* *is tame, if one of the following conditions is fulfilled:*

(i) $d_i \in Z(\mathcal{G})$ *for* $i = 1, \dots, s$

(ii) $d_i \in \mathcal{G}_{i-1}$ *for* $i = 1, \dots, s$, *if* β *is exact-transversal for* $\mathcal{G}$ *with a chain of subgroups* $\gamma : 1 = \mathcal{G}_0 < \mathcal{G}_1 < \cdots < \mathcal{G}_s = \mathcal{G}$, *and* B_i *a complete set of right (left) coset representatives of* $\mathcal{G}_{i-1}$ *in* $\mathcal{G}_i$.

Sketch of Proof: In case (i), the elements $b_{i,j}$ and d_i commute, so we can find logarithmic signature $\beta' := (b'_{i,j})$ equivalent to β^* such that

$$b'_{i,j} = \begin{cases} b_{i,j} & \text{for all } i = 1, \dots, s-1, \\ b_{s,j}d & \text{where } d = d_1 d_2 \dots d_s \end{cases}$$

As β is tame, we are able to factorize in the first $s-1$ blocks of β' and find the last index j_s by exhaustive search.

In case (ii), suppose we are able to factorize $g = b_{1,j_1} b_{2,j_2} \dots b_{s,j_s}$ with respect to β, and trying to factorize $g^* = b^*_{1,j_1} b^*_{2,j_2} \dots b^*_{s,j_s}$ with respect to β^*. As $B^*_i = B_i d_i$ with $d_i \in \mathcal{G}_{i-1}$, it follows that $b_{i,j}$ and $b^*_{i,j}$ are in the same coset of $\mathcal{G}_{i-1}$ in $\mathcal{G}_i$.

We start the factorization of $g^* = b^*_{1,j_1} \dots b^*_{s,j_s}$ with respect to the block B^*_s. Because $b^*_{1,j_1} \dots b^*_{s-1,j_{s-1}} \in \mathcal{G}_{s-1}$, g^* and b^*_{s,j_s} are in the same coset of $\mathcal{G}_{s-1}$ in $\mathcal{G}_s$. This means we can identify the coset of g^*, say $\mathcal{G}_{s-1} b^*_{s,j_s}$ uniquely. Thus, we have found the first factor of g^*, namely b^*_{s,j_s}. We continue with the factorization of $g^*(b^*_{s,j_s})^{-1}$ with respect to the block B^*_{s-1} and identify element $b^*_{s-1,j_{s-1}}$, etc. □

4.3 First Realization on Suzuki 2-groups

To be able to further study the system MST_3, we have to choose a class of groups for a possible realization. We choose $\mathcal{G}$ according to the Property 4.1.1.

Remark 4.3.1 The assumption that $\mathcal{G}$ does not split over $\mathcal{Z}$ implies that there is no subgroup $\mathcal{H} < \mathcal{G}$ with $\mathcal{H} \cap \mathcal{G} = 1$ such that $\mathcal{G} = \mathcal{Z} \times \mathcal{H}$. Without this assumption, the system may be vulnerable to attacks based on permutation group algorithms (see [LMTW09]).

Until now, the only proposed groups for MST_3 remain the class of Suzuki 2-groups described in Section 2.6.

Let $q = 2^m$ with $3 \leqslant m \in \mathbb{N}$ and let θ be a nontrivial automorphism of odd order of the field $\mathbb{F}_q$. Then m cannot be power of 2. Let $\mathcal{G}$ be the Suzuki 2-group $A(m, \theta)$ of order q^2. Then we may realize the MST_3 over $\mathcal{G}$ as described in Algorithm 5. Let $\alpha = [A_1, \dots, A_s]$. The elements of each block A_i are randomly selected according to the following property:

Property 4.3.1 *For all $A_i, i = 1, \dots, s$, $\forall\ x, y \in A_i$, $x \neq y$, the product xy^{-1} is an element of order 4 in $\mathcal{G}$. This means that every two distinct elements x, y of A_i belong to different cosets of $\mathcal{Z}$.*

For efficiency reason, the *Frobenius automorphism* has been chosen for θ to minimize the number of squaring operations needed to extend a group element to its triple representation.

An important requirement of the realization of MST_3 is the efficiency of the factorization with respect to a tame logarithmic signature β.

Remark 4.3.2 As elements of the center $\mathcal{Z}$ are of the form $S(0, b)$, we can identify the center with the additive group of the field $\mathbb{F}_q$, i.e. with a vector space V of dimension m over $\mathbb{F}_2$.

We may use the canonical logarithmic signatures for V as a basis for the key generation. In this realization of MST_3, Algorithm 7 is used to generate a tame logaritmic signature β which is a part of the private key. This reduces the complexity of factoring with respect to β to $\mathcal{O}(1)$.

4.4 Security Analysis of the First Realization

4.4.1 Used notation

Here we define notation used below and note some facts resulting from the usage of the Suzuki 2-group $A(m, \theta)$ in the realization of MST_3.

If $g = S(x, y) \in \mathcal{G}$, $x, y \in \mathbb{F}_q$, we denote x by $g_{.a}$, and y by $g_{.b}$, that is, we denote the projections of $g \in \mathcal{G}$ along the first and second coordinates by $g_{.a}$ and $g_{.b}$ respectively. Thus, we write $g = S(g_{.a},\ g_{.b})$. Accordingly, we denote the elements of the public key $\alpha := (a_{i,j})$, $\gamma := (h_{i,j})$, known to the adversary, by pairs $S(a_{(i,j).a},\ a_{(i,j).b})$, and $S(h_{(i,j).a},\ h_{(i,j).b})$ respectively. Similarly, the secret $\beta := (b_{i,j})$, $(t_0, \dots, t_s)$ are denoted by pairs $S(b_{(i,j).a},\ b_{(i,j).b})$, and $S(t_{(i).a},\ t_{(i).b})$.

The following lemma is quite easy to see:

Lemma 4.4.1 *In terms of the notation introduced thus far we have:*

(i) The inverse of $g = S(g_{.a},\ g_{.b}) \in \mathcal{G}$ *is given by rule*

$$S(g_{.a},\ g_{.b})^{-1} = S(g_{.a},\ g_{.b} + {g_{.a}}^{\theta+1})$$

(ii) Both operations, inversion and matrix transformation, keep the "$.a$ *part" of an element* g *invariant.*

(iii) Elements from the same coset of the center $\mathcal{Z}$*, have identical "*$.a$ *part" projections, i.e. if* $t_j \in t_i\mathcal{Z}$ *then,*

$$t_{(i).a} = t_{(j).a}$$

4.4.2 Attack on a private key

In this attack, an adversary attempts to extract information about the private key $\big[\beta, (t_0, \ldots, t_s)\big]$ from knowledge of the public key $\big[\alpha, \gamma\big]$. However, we will show that it is sufficient for him to obtain $\big[\beta^*, (t^*_0, \ldots, t^*_s)\big]$ such that for all $i = 1, \ldots, s$ and each $j = 1, \ldots, r_i$:

$$h_{i,j} = b^*_{i,j}\ t^{*-1}_{i-1}\ a_{i,j}\ t^*_i$$

Assumption 4.4.1 *Assume that* $t_i = t^*_i z_i$ *for some* $z_i \in \mathcal{Z}$*, i.e. the elements* t_i *and* t^*_i *are from the same coset* $t_i\mathcal{Z}$*.*

Let $b_{i,j} = b^*_{i,j} d_{i,j}$ for some $d_{i,j} \in \mathcal{Z}$. By assumption 4.4.1, $t_0 = t^*_0 z_0$, so for the first block of γ:

$$h_{1,j} = b_{1,j}\ t^{*-1}_0\ z_0\ a_{1,j}\ t_1$$

We fix $b^*_{i,1} = id$, then $d_{i,1} = b_{i,1}$ and

$$\begin{aligned} h_{1,1} &= b^*_{1,1}\ d_{1,1}\ t^{*-1}_0\ z_0\ a_{1,1}\ t_1 \\ &= b^*_{1,1}\ t^{*-1}_0\ a_{1,1}\ (t_1\ d_{1,1}\ z_0) \quad \Rightarrow \quad t^*_1 = t_1\ d_{1,1}\ z_0 \end{aligned}$$

$$h_{1,j} = b_{1,j}\ t_0^{*-1}\ z_0\ a_{1,j}\ t_1^*\ d_{1,1}\ z_0 \qquad \text{for all } j = 2, \dots, r$$
$$b_{1,j}^* = b_{1,j}\ d_{1,j} = \ h_{1,j}\ t_1^{*-1}\ a_{1,j}^{-1}\ t_0^* \quad \Rightarrow \quad d_{1,j} = d_{1,1} = b_{1,1}$$

$$h_{2,1} = b_{2,1}^*\ d_{2,1}\ t_1^{*-1}\ d_{1,1}\ z_0\ a_{2,1}\ t_2 \quad \Rightarrow \quad t_2^* = t_2\ d_{2,1}\ d_{1,1}\ z_0$$
$$h_{2,j} = b_{2,j}\ t_1^{*-1}\ d_{1,1}\ z_0\ a_{2,j}\ t_2^*\ d_{2,1}\ d_{1,1}\ z_0 \qquad \text{for all } j = 2, \dots, r$$
$$b_{2,j}^* = b_{2,j}\ d_{2,j} = \ h_{2,j}\ t_2^{*-1}\ a_{2,j}^{-1}\ t_1^* \quad \Rightarrow \quad d_{2,j} = d_{2,1} = b_{2,1}$$
$$\vdots$$

It follows that $d_{i,j} = d_{i,1} = b_{i,1}$, for all $i = 1, \dots, s$. Denote $d_{i,j} = d_i$.
Notice that $t_i^* = t_i\ z_0 \prod_{k=1}^{i} d_k$.

We encrypt message $x \in \mathbb{Z}_{|\mathcal{Z}|}$:

$$\begin{aligned} \breve{\gamma}(x) = &\ \breve{\beta}(x)\ t_0^{-1}\ \breve{\alpha}(x)\ t_s \\ &= \breve{\beta}(x)\ t_0^{*-1}\ z_0\ \breve{\alpha}(x)\ t_s^*\ z_0 \prod_{k=1}^{s} d_k \\ &= \left(\breve{\beta}(x) \prod_{k=1}^{s} d_k\right)\ t_0^{*-1}\ \breve{\alpha}(x)\ t_s^* \end{aligned}$$

Let $\breve{\beta}(x) = b_{1,x_1}\ b_{2,x_2}\ \dots b_{s,x_s}$ and $\beta^* := (b_{i,j}^*)$, $b_{i,j}^* = b_{i,j}\ d_i$, then

$$\begin{aligned} \breve{\beta^*}(x) &= b_{1,x_1}^*\ b_{2,x_2}^*\ \dots b_{s,x_s}^* \\ &= b_{1,x_1}\ d_1\ b_{2,x_2}\ d_2\ \dots b_{s,x_s}\ d_s \\ &= \breve{\beta}(x) \prod_{k=1}^{s} d_k \end{aligned}$$

We recover message x using $[\beta^*, (t_0^*, \dots, t_s^*)]$ correctly, and as β^* is tame (see Proposition 4.2.4) we factorize efficiently too.

From the above, we can summarize to following conclusion.

Conclusion 4.4.1 In the realization of MST_3 with Suzuki 2-group, it is sufficient for an adversary to obtain element $t_0^* \in t_0\mathcal{Z}$. With this knowledge he can compute an "equivalent" private key $[\beta^*, (t_0^*, \dots, t_s^*)]$ uniquely and use it to decrypt messages correctly. The logarithmic signature β^* is tame so the attacker decrypts efficiently. As there are $q = |\mathcal{G}/\mathcal{Z}|$ possible choices for t_0^* in $t_0\mathcal{Z}$, the workload required for this attack is $\mathcal{O}(q)$.

4.4.3 Attack on MST_3 when using canonical signature

We show how an adversary can choose t_0^* to break MST_3. He exploits a special property of the canonical logarithmic signature used as a basis for the key generation.

In this attack, we have to step down from group $\mathcal{G}$ to the underlying field $\mathbb{F}_q$.

For the first block of γ:

$$h_{1,j} = b_{1,j}\ t_0^{-1}\ a_{1,j}\ t_1$$

Particularly, for each part

$$\begin{aligned} h_{(1,1).a} &= t_{(0).a} + a_{(1,1).a} + t_{(1).a} \\ h_{(1,j).b} &= b_{(1,j).b} + t_{(0).b} + t_{(0).a}{}^{\theta+1} + a_{(1,j).b} + t_{(1).b} + t_{(0).a}(a_{(1,j).a})^{\theta} + \\ &\quad + t_{(0).a}(t_{(1).a})^{\theta} + a_{(1,j).a}(t_{(1).a})^{\theta} \end{aligned}$$

For an index set J yet to be determined, with $|J|$ even

$$\begin{aligned} \sum_{j\in J} h_{(1,j).b} &= \sum_{j\in J} b_{(1,j).b} + \sum_{j\in J} a_{(1,j).b} + t_{(0).a}\sum_{j\in J}(a_{(1,j).a})^{\theta} + \\ &\quad + (t_{(1).a})^{\theta}\sum_{j\in J} a_{(1,j).a} \\ &= \sum_{j\in J} b_{(1,j).b} + \sum_{j\in J} a_{(1,j).b} + t_{(0).a}\sum_{j\in J}(a_{(1,j).a})^{\theta} + \\ &\quad + (h_{(1,1).a} + t_{(0).a} + a_{(1,1).a})^{\theta}\sum_{j\in J} a_{(1,j).a} \end{aligned}$$

Considering $t_{(0).a}$ as an unknown we end up with a trinomial:

$$A(t_{(0).a})^{\theta} + B(t_{(0).a}) + C = 0 \qquad (4.4.1)$$

where

$$\begin{aligned} A &= \sum_{j\in J} a_{(1,j).a} \\ B &= \sum_{j\in J} (a_{(1,j).a})^{\theta} \\ C &= \sum_{j\in J} b_{(1,j).b} + \sum_{j\in J} h_{(1,j).b} + \sum_{j\in J} a_{(1,j).b} + (h_{(1,1).a} + a_{(1,1).a})^{\theta}\sum_{j\in J} a_{(1,j).a} \end{aligned}$$

The term $(t_{(0).a})^\theta$ in the trinomial expresses the action of θ on element $t_{(0).a} \in \mathbb{F}_q$. Since Frobenius automorphism $\theta: \ a \to a^2$ has been chosen for efficiency reasons, the term $(t_{(0).a})^\theta$ becomes $(t_{(0).a})^2$.

The question here is how to choose J. The unknown in the quadratic equation is $\sum_{j\in J} b_{(1,j).b}$, the sum of elements of the first block of β. As it is part of the private key, and to guess it is infeasible, we choose J such that $\sum_{j\in J} b_{(1,j).b}$ will sum up to zero.

Here the special structure of the canonical logarithmic signature used as a basis for β can be exploited. As $\beta := (b_{i,j}) = (e_{i,j}.M)$, for some canonical signature $\varepsilon := (e_{i,j})$ and $M \in GL(m, 2)$, it is true that

$$\sum_{j\in J} b_{(1,j).b} = \sum_{j\in J} e_{(1,j).b}.M = (\sum_{j\in J} e_{(1,j).b})M$$

The first block E_1 of the canonical logarithmic signature ε consists of a complete set of 2^{k_i} vectors, so we are always able to find $J \subseteq \{1, \dots, r_1\}$ such that

$$\sum_{j\in J} e_{(i,j).b} = 0 \tag{4.4.2}$$

Case 1: $\sum_{A_1} a_{(1,j).a} \neq 0$
Choosing the whole block E_1 of ε, we get the sum $\sum_{E_1} e_{(1,j).b} = 0$, so is $\sum_{B_1} b_{(1,j).b} = 0$. Therefore, choosing $J = \{1, \dots, r_1\}$ in this case leads to a nontrivial solution of the Equation 4.4.1 and therefore allows to determine the coset of t_0.

Case 2: $\sum_{A_1} a_{(1,j).a} = 0$
In this case, choosing the whole block B_1 will not lead to the solution of the trinomial. Therefore, we have to find a proper subset $J \subset \{1, \dots, r_1\}$, such that the required condition 4.4.2 will be satisfied and $\sum_J a_{(1,j).a} \neq 0$. As the elements $e_{(1,j).b}$ in E_1 contain at most k_1 ones, $r_1 = 2^{k_1}$, the probability that the arbitrary subset J gives required condition is $1/2^{k_1}$. In practice, k_1 is sufficiently small to allow the adversary to determine $t_{(0).a}$ using the brute force attack. Increasing k_1 extends the key size exponentially and is therefore not reasonable.

In both cases, breaking the system reduces to the problem of finding the root of the trinomial in Equation 4.4.1 over $\mathbb{F}_q$. Well known results of Berlekamp or Shoup solve the problem of factoring polynomial of degree n over $\mathbb{Z}_p[x]$ in time polynomial in n and p [Ber70, Sho90].

Conclusion 4.4.2 From the above, an adversary can use the public key and knowledge of the structure of canonical logarithmic signatures to uniquely determine the coset $t_0\mathcal{Z}$. This information is sufficient for recovering an "equivalent" private key and decrypting messages correctly. These observations reduce the complexity of an attack to $\mathcal{O}(1)$, when θ is the Frobenius automorphism of $\mathbb{F}_q$.

4.4.4 Normalized MST_3

Here we present a simplification of MST_3, published in [BCM09], which, independently of the group used, reduces private key material required by the cryptosystem. We also show, that in general, the knowledge of one element t_0 of the private key $[\beta, (t_0, \dots, t_s)]$ constitute to breaking the system.

Let $[\alpha, \gamma]$ be a public key for MST_3, with $[\beta, (t_0, \dots, t_s)]$ the corresponding private key. Recall the construction of the public key in Algorithm 5. For $i \in \{1, \dots, s\}$ and $j \in \{1, \dots, r_i\}$, where $(r_1, \dots, r_s)$ is the type of the following covers

$$\begin{aligned}
\beta &= [B_1, \dots, B_s] := (b_{i,j}) \\
\alpha &= [A_1, \dots, A_s] := (a_{i,j}) \\
\tilde{\alpha} &= [\tilde{A_1}, \dots, \tilde{A_s}] \text{ where } \tilde{A_i} = t_{i-1}^{-1} A_i t_i \\
\gamma &= [H_1, \dots, H_s] := (h_{i,j}) = (b_{i,j}\tilde{a}_{i,j})
\end{aligned}$$

where

$$h_{i,j} = b_{i,j}\ t_{i-1}^{-1}\ a_{i,j}\ t_i$$

Define elements p_i, q_i, z_i, such that $p_0 = q_0 = z_0 = 1$ and for $k \in \{1, \dots, s\}$

$$p_k = \prod_{i=1}^{k} a_{i,1}\ , \quad q_k = \prod_{i=1}^{k} h_{i,1} \text{ and } z_k = \prod_{i=1}^{k} b_{i,1}\ .$$

Also define the covers $\alpha' := (a'_{i,j})$, $\gamma' := (h'_{i,j})$ and $\beta' := (b'_{i,j})$ where for $i \in \{1, \dots, s\}$ and $j \in \{1, \dots, r_i\}$

$$a'_{i,j} = p_{i-1} a_{i,j} p_i^{-1}\ , \quad h'_{i,j} = q_{i-1} h_{i,j} q_i^{-1} \text{ and } b'_{i,j} = z_{i-1} b_{i,j} z_i^{-1}\ .$$

Then

$$\breve{\alpha}'(x) = \breve{\alpha}(x) p_s^{-1}\ , \quad \breve{\gamma}'(x) = \breve{\gamma}(x) q_s^{-1} \text{ and } \breve{\beta}'(x) = \breve{\beta}(x) z_s^{-1}\ .$$

Lemma 4.4.2 (Blackburn et al. [BCM09]) *Let* $[\alpha, \gamma]$ *be a public key for* MST_3*, with* $[\beta, (t_0, \ldots, t_s)]$ *the corresponding private key. Define* α', β' *and* γ' *as above. Also let* $t'_0 = \cdots = t'_s = t_0$*. Then* $[\alpha', \gamma']$ *is a public key for* MST_3*, with corresponding private key* $[\beta', (t'_0, \ldots, t'_s)]$.

Proof: Let $[\alpha', \beta',\ t'_0, \ldots, t'_s]$ be a private key used to generate a public key $[\alpha', \delta]$ and $\delta := (d_{i,j})$ with $d_{i,j} = b'_{i,j} t'^{-1}_{i-1} a'_{i,j} t'_i$. It suffices to show that $\delta = \gamma'$.

$$\begin{aligned} d_{i,j} &= b'_{i,j}\ t_0^{-1} a'_{i,j} t_0 \\ &= z_{i-1} b_{i,j} z_i^{-1}\ t_0^{-1}\ p_{i-1} a_{i,j} p_i^{-1}\ t_0 \\ &= b_{i,j} b_{i,1}^{-1}\ t_0^{-1}\ p_{i-1} a_{i,j} p_i^{-1}\ t_0 \end{aligned}$$

Equation

$$q_k = \prod_{i=1}^{k} h_{i,1} = \prod_{i=1}^{k} b_{i,1} t_{i-1}^{-1} a_{i,1} t_i = z_k\ t_0^{-1}\ p_k\ t_k$$

implies

$$t_0^{-1} p_{k-1} = z_{k-1}^{-1} q_{k-1} t_{k-1}^{-1} \quad \text{and} \quad p_k^{-1} t_0 = t_k q_k^{-1} z_k\ .$$

Then

$$\begin{aligned} d_{i,j} &= b_{i,j} b_{i,1}^{-1}\ z_{i-1}^{-1} q_{i-1} t_{i-1}^{-1}\ a_{i,j}\ t_i q_i^{-1} z_i \\ &= b_{i,j}\ q_{i-1}\ t_{i-1}^{-1} a_{i,j} t_i\ q_i^{-1} \\ &= q_{i-1}\ b_{i,j} t_{i-1}^{-1} a_{i,j} t_i\ q_i^{-1} \\ &= q_{i-1}\ h_{i,j}\ q_i^{-1} \end{aligned}$$

So $\delta = \gamma'$, as required. □

Suppose $[\alpha, \gamma]$ is a public key with corresponding private key $[\beta, (t_0, \ldots, t_s)]$ for MST_3. Let (y_1, y_2) be a ciphertext for a message x, where $y_1 = \breve{\alpha}(x)$ and $y_2 = \breve{\gamma}(x)$.

We define *normalized* MST_3 as above, with public key $[\alpha', \gamma']$ and corresponding private key $[\beta', (t'_0, \ldots, t'_0)]$. The covers α' and γ' can be efficiently constructed using public information $[\alpha, \gamma]$. By encryption of the cleartext x we receive ciphertext (y'_1, y'_2), where $y'_1 = \breve{\alpha}'(x) = y_1 p_s^{-1}$ and $y'_2 = \breve{\gamma}'(x) = y_2 q_s^{-1}$. Both p_s and q_s are defined using public information, thus y'_1 and y'_2 can be efficiently computed. Then decrypting the ciphertext $[y_1, y_2]$ with the private key $[\beta, (t_0, \ldots, t_s)]$ in MST_3 is equivalent to decrypting of the ciphertext $[y'_1, y'_2]$ with the private key $[\beta', (t'_0, \ldots, t'_0)]$ in normalized MST_3, and the cleartexts obtained are identical.

We get the following Collorary (see also [BCM09]).

Corollary 4.4.1 *There exists a polynomial time algorithm to transform* MST_3 *into normalized* MST_3*.*

Sketch of Proof: Suppose $[\alpha, \gamma]$ is a public key for MST_3 and (y_1, y_2) a ciphertext for some unknown cleartext x. We define $[\alpha', \gamma']$ as above. As

$$\breve{\alpha}(x) = \breve{\alpha}'(x)p_s = y_1'p_s = y_1 p_s^{-1} p_s = y_1 \quad \text{and}$$
$$\breve{\gamma}(x) = \breve{\gamma}'(x)q_s = y_2'q_s = y_1 q_s^{-1} q_s = y_2,$$

we also define $y_1' = y_1 p_s^{-1}$ and $y_2' = y_2 q_s^{-1}$. Note that all $\alpha', \gamma', y_1', y_2'$ can be efficiently constructed using public information only. $\square$

Remark 4.4.1 Let $\beta := (b_{i,j})$ be a tame logarithmic signature for $\mathcal{Z}$. Define $z_0 = z_s = 1_{\mathcal{Z}}$ and $z_i = \prod_{k=1}^{i} b_{k,1}$ for $i = 1, \ldots, s-1$. Now define $\beta'' := (z_{i-1}\, b_{i,j}\, z_i^{-1})$. Note that β'' is normalized logarithmic signature equivalent to β. Now define β' as a right translation of β'' by the inverse of element $w = \prod_{k=1}^{s} b_{k,1}$. Clearly, $\breve{\beta}'(x) = \breve{\beta}''(x)\, w^{-1} = \breve{\beta}(x)\, w^{-1}$, hence β' is also tame.

Conclusion 4.4.3 Supose $[\beta, (t_0, \ldots, t_s)]$ is a private key for a realization of MST_3. There exists a polynomial time algorithm to transform MST_3 into normalized MST_3 with a private key $[\beta', (t_0, \ldots, t_0)]$. Moreover, if the element $u_0 \in t_0\mathcal{Z}$ is known, an adversary can use the public key $[\alpha, \gamma]$ to construct tame logarithmic signature β' and use it to decrypt messages correctly.

Remark 4.4.2 Consider the following problem.

Question: Can $u_0 \notin t_0\mathcal{Z}$ be used to break the system if the Suzuki 2-group is used?

Suppose $[\beta, (t_0, \ldots, t_0)]$ is a private key and $[\alpha, \gamma]$ an appropriate public key for a normalized MST_3 in Suzuki 2-group. Now let $[\varepsilon, (u_0, \ldots, u_0)]$ be another private key for the same cryptosystem. For $\beta := (b_{i,j})$, $\varepsilon := (e_{i,j})$, $\alpha := (a_{i,j})$, $\gamma := (h_{i,j})$ and $i = 1, \ldots, s$; $j = 1, \ldots, r_i$

$$h_{i,j} = t_0^{-1} a_{i,j} t_0 b_{i,j} = u_0^{-1} a_{i,j} u_0 e_{i,j}$$

Now let $u_0 = w_0^{-1} t_0$ for some $w_0 \in \mathcal{G}$, then

$$e_{i,j} = a_{i,j}^{-1} w_0^{-1} a_{i,j} w_0 b_{i,j}$$

implies

$$e_{i,j} = S(0,\ b_{(i,j).b} + w_{(0).a} a_{(i,j).a}^{\theta} + a_{(i,j).a} w_{(0).a}^{\theta})$$

As α is a random cover for $\mathcal{G}$, the cover $\delta := (d_{i,j})$ with $d_{i,j} := S(0,\ a_{(i,j).a})$ is random cover for $\mathcal{Z}$, so is ε. Clearly, choosing $u_0 \in t_0\mathcal{Z}$ results in $\beta = \varepsilon$, because $w_0 \in \mathcal{Z}$ implies $w_{(0).a} = 0$. If $u_0 \notin t_0\mathcal{Z}$ the resulting ε must not in general be a logarithmic signature. Moreover, due to the Cryptographic Hypothesis 1 we cannot assume that the factorization with respect to ε is feasible (even in the case that ε accidentally is a logarithmic signature).

4.5 Conclusions

In this chapter, we give a description of the generic version of MST_3 and its realization with Suzuki 2-groups. We prove a new general lower bound for the security complexity of MST_3. This bound has size $q = 2^m$, where q is the order of the field $\mathbb{F}_q$, on which the Suzuki 2-group $A(m, \theta)$ is defined.

Afterwards, we present an attack on this realization when a special type of transversal logarithmic signatures, called canonical, is used as a basis for the private key. Canonical signatures are easily contructed and allow a very efficient factorization. We provide an attack exploiting the properties of the group operation in the Suzuki 2-groups as well as the special structure of canonical signatures for elementary abelian 2-groups. This attack shows that canonical transversal logarithmic signatures cannot be used for a secure realization of MST_3 with Suzuki 2-groups.

We also present a simplification of the MST_3 scheme, showing that instead of private key $[\beta,\ (t_0, \ldots, t_s)]$, the corresponding key $[\beta', (t_0, \ldots, t_0)]$ can be constructed and used to decrypt ciphertexts. Moreover, we argue that the knowledge of the coset of t_0 (or coset of any one element t_i) suffices to recover signature β' in this realization. Choosing $u_0 \notin t_0\mathcal{Z}$ however leads to the problem of factorization with respect to a random cover.

Chapter 5

Second Realization of MST_3 on Suzuki 2-Groups

Motivated by the attack in the previous chapter, we construct a modification of the original scheme. In this new design, we introduce a secret homomorphism f from $\mathcal{G}$ to $\mathcal{G}/\mathcal{Z}$. This homomorphism is used to mask a secret logarithmic signature β with a tranformation of random cover α. We propose a new set-up with randomized encryption.

As before, the underlying group $\mathcal{G}$ has to be chosen according to the Property 4.1.1 (with nontrivial center $\mathcal{Z}$ such that $\mathcal{G}$ does not split over $\mathcal{Z}$, see also Remark 4.3.1), with $\mathcal{Z}$ sufficiently large so that exhaustive search problems are computationally not feasible in $\mathcal{Z}$.

5.1 Set-up

Let $\mathcal{G}$ be the Suzuki 2-group $A(m, \theta)$ of order q^2, and $\mathcal{Z} = Z(\mathcal{G})$ an elementary abelian 2-group of order q viewed as a vector space of dimension m over $\mathbb{F}_2$. The automorphism group of $\mathcal{Z}$ is the general linear group $GL(m, 2)$. Denote $Aut(\mathcal{Z}) := GL(m, 2)$. If $z = S(0, b) \in \mathcal{Z}$ and $\varphi \in Aut(\mathcal{Z})$, then the action of φ on z is defined by $z^{\varphi} := S(0, b^{\varphi})$.

Remark 5.1.1 Let f be any homomorphism from $\mathcal{G}$ to $\mathcal{Z}$. Let $N = Ker(f)$. Then N is normal subgroup of $\mathcal{G}$ and $\mathcal{G}/N \cong f(\mathcal{G}) \subseteq \mathcal{Z}$. So, the factor group $\mathcal{G}/N$ is abelian. As the commutator group $\mathcal{G}' = \mathcal{Z}$ we have $N \geqslant \mathcal{Z}$. It follows that $f(z) = 1$ for every $z \in \mathcal{Z}$.

Algorithm 8 Set-up of MST_3

Key generation

Input: A large group $\mathcal{G}$ as described above.

Output: A public key $[\alpha, \gamma]$ with corresponding private key $[\beta, (t_0, \dots, t_s), f]$.

```
1: a factorizable logarithmic signature $\beta = [B_1, \dots, B_s] := (b_{i,j})$ of type $(r_1, \dots, r_s)$ for $\mathcal{Z}$.
2: a random cover $\alpha = [A_1, A_2, \dots, A_s] := (a_{i,j})$ of the same type as $\beta$ for a certain subset $\mathcal{J}$ of $\mathcal{G}$ such that $A_1, \dots, A_s \subseteq \mathcal{G} \setminus \mathcal{Z}$. The elements in each block $A_i = [a_{i,1}, a_{i,2}, \dots, a_{i,r_i}]$ are chosen to satisfy the following conditions:
    (i) $a_{(i,j_1).a} \neq a_{(i,j_2).a}$, for $j_1 \neq j_2$. This is equivalent to saying that $a_{(i,j_1)}$ and $a_{(i,j_2)}$ are not in the same coset of $\mathcal{Z}$.
    (ii) $\sum_{j=1,\dots,r_i} a_{(i,j).a} = 0$. The meaning of this condition will be obvious when we discuss the security of the system in the subsequent section.
Further select
3: $t_0, t_1 \dots, t_s \in \mathcal{G} \setminus \mathcal{Z}$.
4: a homomorphism $f : \mathcal{G} \longrightarrow \mathcal{Z}$
and compute
5: $\gamma = (h_{i,j}), \quad h_{i,j} = t_{i-1}^{-1} \cdot a_{i,j} \cdot f(a_{i,j}) \cdot b_{i,j} \cdot t_i$
```

Encryption (randomized)

Input: A message $x \in \mathcal{Z}$ and the public key α and γ.

Output: A ciphertext (y_1, y_2) of the message x.

```
1: choose a random $R \in \mathbb{Z}_{|\mathcal{Z}|}$ and compute
2: $y_1 = \breve{\alpha}(R) \cdot x$,
   $y_2 = \breve{\gamma}(R) \cdot x = t_0^{-1} \cdot \breve{\alpha}(R) \cdot f(\breve{\alpha}(R)) \cdot \breve{\beta}(R) \cdot t_s \cdot x$
```

Decryption

Input: A ciphertext pair (y_1, y_2) and the private key $\beta = (b_{i,j}),\ t_0, t_s,\ f$.

Output: The message $x \in \mathcal{Z}$ that corresponds to the ciphertext (y_1, y_2).

```
1: Using the fact that $f(y_1) = f(\breve{\alpha}(R))$ (from Remark 5.1.1) compute
   $\breve{\beta}(R) = f(\breve{\alpha}(R))^{-1} \cdot y_1^{-1} \cdot t_0 \cdot y_2 \cdot t_s^{-1} = f(y_1)^{-1} \cdot y_1^{-1} \cdot t_0 \cdot y_2 \cdot t_s^{-1}$
2: Recover $R$ from $\breve{\beta}(R)$ which is efficiently computable as $\beta$ is factorizable. By computing $\breve{\alpha}(R)$ we then recover $x$ from $y_1$.
```

Recall $\mathsf{g}_{.\mathfrak{a}}$ and $\mathsf{g}_{.\mathfrak{b}}$ denoting the projections of $g \in \mathcal{G}$ along the first and second coordinates for some $g \in \mathcal{G}$, i.e. $g = S(g_{.a},\ g_{.b}),\ g_{.a}, g_{.b} \in \mathbb{F}_q$.

Specification of the homomorphism f

For the realization of the cryptosystem MST_3 we use the following class of homomorphisms. Let $g = S(g_{.a}, g_{.b}) \in \mathcal{G}$, and let $\sigma \in \mathrm{Aut}(\mathcal{Z}) := \mathrm{GL}(m, 2)$. Define

$$f : \mathcal{G} \longrightarrow \mathcal{Z}$$
$$f(g) := S(0, g_{.a}^{\sigma}).$$

Then f is a homomorphism from $\mathcal{G}$ to $\mathcal{Z}$.

The MST_3 as just described for the Suzuki 2-groups can be generalized, of course, for many other classes of finite groups, for example, the class of special p-groups. An interesting class of p-groups, also dubbed Suzuki p-groups, for odd primes p, see [BJ08], may be viewed as a natural candidate for the underlying groups of MST_3.

The encryption method of MST_3, as described above, is a randomized encryption. However, if we consider $\mathbb{Z}_{|\mathcal{Z}|}$ as the message space and encrypt a message $z \in \mathbb{Z}_{|\mathcal{Z}|}$ by computing

$$(y_1, y_2) = (\breve{\alpha}(z), \breve{\gamma}(z))$$

as ciphertext, we obtain a non-randomized encryption. It is worth noting that the non-randomized encryption can be set up within the framework of the randomized encryption method: replace R by z and take $x = 1_{\mathcal{Z}}$.

Note that there is no additional ciphertext expansion if the randomized encryption is used. The cost is only in time complexity for encryption and decryption where additional operations are required.

To make the discussion of the cryptanalysis of the scheme in the subsequent sections simpler, we only consider the non-randomized encryption.

5.2 Generation of logarithmic signature β and its factorization

In this section, we describe a method of generating logarithmic signature β for the realization of MST_3 and show methods of factorization with respect to β. As the center $\mathcal{Z}$ of $\mathcal{G}$ is an elementary abelian group, we will use the following possible transformations in generating logarithmic signature β.

5.2.1 Transformations of logarithmic signatures

Let $\varepsilon = [E_1, \dots, E_\nu] := (e_{i,j})$ be a logarithmic signature of type $(t_1, \dots, t_\nu)$ for an abelian group H. We define the following transformations on ε:

(T1) transform each element of ε with an automorphism φ of H

(T2) fuse j blocks $E_{k_1}, \dots, E_{k_j}$, i.e. replace blocks $E_{k_1}, \dots, E_{k_j}$ with a new block of the form $(((E_{k_1}.E_{k_2}).E_{k_3}) \dots E_{k_j})$, where $E_i \cdot E_j := E_i \otimes E_j = [e_{i,1}e_{j,1}, \dots, e_{i,1}e_{j,t_j}, e_{i,2}e_{j,1}, \dots, e_{i,2}e_{j,t_j}, \dots, e_{i,t_i}e_{j,t_j}]$

(T3) permute the elements within each block E_i with a permutation π_i in $\mathfrak{S}_{t_i}$

(T4) permute the blocks E_i's with a permutation $\xi \in \mathfrak{S}_\nu$ (where $\mathfrak{S}_\nu$ is symmetric group on ν symbols)

It is obvious that β obtained from ε by using transformations T1,T2,T3 and T4 is a logarithmic signature for H. If ε is tame, we can factorize with β using the knowledge of the transformations T1,. . . ,T4 in polynomial time (as shown by an algorithm presented in a subsequent section).

5.2.2 Algorithm for generation of β

We will describe an algorithm for generating logarithmic signature β for use in MST_3. As a base we use canonical logarithmic signature for elementary abelian 2-group defined in Section 4.2. We identify the center $\mathcal{Z}$ of $\mathcal{G}$ with a vector space V of dimension m over $\mathbb{F}_2$ and use Algorithm 6 to generate canonical logarithmic signature for V.

The following statement is not difficult to prove, see Proposition 4.2.2.

Let $\delta := (d_{i,j})$ be a canonical logarithmic signature for an elementary abelian 2-group V of order 2^m. Let $\rho \in GL(m, 2)$ be an $m \times m$ matrix and define $\delta^* := (d_{i,j}^\rho)$. Then δ^* is a tame logarithmic signature. It is clear that the signature δ^* is a transversal signature for a certain chain of subgroups $1_V = V_0 < V_1 < \dots < V_s = V$ of V. Moreover, it is shown in Section 4.2 that the factorization with respect to a canonical logarithmic signature will have time complexity $\mathcal{O}(1)$.

Definition 5.2.1 *We call* β **fused transversal** *(FT) logarithmic signature, if* β *is generated by Algorithm 9. If step 3 (i.e. fusion of blocks) of the algorithm is not applied,* β *is called* **non-fused transversal** *(NFT) logarithmic signature.*

We now describe an algorithm for generating logarithmic signature β.

Algorithm 9 Generation of logarithmic signature β

1: Let $\varepsilon = [E_1, E_2, \ldots, E_\nu] := (e_{i,j})$ be the canonical logarithmic signature in standard form of type $(t_1, t_2, \ldots, t_\nu)$ for $\mathcal{Z}$ (viewed as an m dimensional vector space over $\mathbb{F}_2$) corresponding to the partition $\{K_1, K_2, \ldots, K_\nu\}$ on the set $\{1, \ldots, m\}$ with $|K_i| = k_i$ and $t_i = 2^{k_i}$ (see Definition 4.2.2).
Denote $\varepsilon^* = (e^*_{i,j})$ a logarithmic signature obtained from ε by filling the positions $K_1 \cup \ldots \cup K_{i-1}$ of each block E_i with random bits, $i = 2, \ldots, \nu$. We call ε^* a *randomized canonical* logarithmic signature.

2: [transformation T1] Select a random matrix $\rho \in GL(m, 2)$ and compute

$$\delta = [D_1, \ldots, D_\nu] = (d_{i,j}) := ((e^*_{i,j})^\rho).$$

3: [transformation T2] Select a partition $\mathcal{P} = \{P_1, \ldots, P_s\}$, $0 < |P_j|$, of the set $\{1, \ldots, \nu\}$, such that for each $P_j = \{i_1, \ldots, i_u\}$, i.e. $|P_j| = u$, we have $i_h \neq i_\ell + 1$ for $h, \ell \in \{1, \ldots, u\}$. Fuse blocks $D_{i_1}, \ldots, D_{i_u}$, i.e. construct the product $C_j := (((D_{i_1}.D_{i_2}).D_{i_3} \ldots D_{i_u})$. Let $\omega = [C_1, \ldots, C_s] := (c_{i,j})$ be the resulting logarithmic signature of type $(r_1, \ldots, r_s)$ obtained after this step.

4: [transformation T3] Select random permutations $\pi_i \in \mathfrak{S}_{r_i}$, $i = 1, \ldots, s$, where $\mathfrak{S}_{r_i}$ is the symmetric group of degree r_i. Define

$$C^*_i := C_i^{\pi_i} = [c_{i,\pi_i(1)}, c_{i,\pi_i(2)}, \ldots, c_{i,\pi_i(t_i)}],$$

i.e. C^*_i is obtained from C_i by permuting the positions of its elements with permutation π_i. Denote $\chi = [C^*_1, \ldots, C^*_s]$.

5: [transformation T4] Select a random permutation $\xi \in \mathfrak{S}_s$ and define

$$\beta = [B_1, \ldots, B_s] := [C^*_{\xi(1)}, \ldots, C^*_{\xi(s)}],$$

i.e. β is obtained from χ by permuting the positions of its blocks with ξ.

It should be noted that in order to have an efficient factorization with respect to β created using Algorithm 9, we keep track of the information about matrix ρ, logarithmic signature ε^*, partition $\mathcal{P}$ and all permutations used in steps 4 and 5.

5.2.3 Factorization with β

In this section, we present algorithms for the factorization with β generated by Algorithm 9. We begin by proving the following useful proposition.

Proposition 5.2.1 *Let $\beta := [B_1, \dots, B_v]$ be a transversal logarithmic signature for an abelian group H. Let $\beta' := [B'_1, \dots, B'_s]$ be a fused logarithmic signature of H obtained by fusion of blocks of β [transformation T2]. Then β' is equivalent to a non-fused logarithmic signature β'' obtained from β by using certain permutation $\mu \in \mathfrak{S}_v$ on blocks B_i [transformation T4]. In other words β' and β'' induce the same function, i.e. $\breve{\beta}' = \breve{\beta}''$.*

Proof: We observe that β' is obtained from β by using the following two operations:

(a) select an appropriate permutation $\mu \in \mathfrak{S}_v$ and compute

$$\beta'' := [B''_1, \dots, B''_v] := [B_{1^\mu}, \dots, B_{v^\mu}].$$

(b) select a partition $\mathcal{R} = \{R_1, \dots, R_s\}$ on the set $\{1, \dots, v\}$ with $R_1 = \{1, \dots, i_1\}, R_2 = \{i_1 + 1, \dots, i_2\}, \dots, R_s = \{i_{s-1} + 1, \dots, i_s\}$ with $|R_j| = u_j$ for $j \in \{1, \dots, s\}$. Fusing the blocks of β'' according to this partition yields the logarithmic signature $\beta' := [B'_1, \dots, B'_s]$ of type $(r_1, \dots, r_s)$ with $B'_j = ((B''_{i_{j-1}+1}.B''_{i_{j-1}+2}) \dots B''_{i_j})$, where $r_j = |B''_{i_{j-1}+1}|.|B''_{i_{j-1}+2}| \dots |B''_{i_j}|$ for $j = 1, \dots, s$ and $i_0 = 0$.
(i.e. each block B'_i is obtained by fusing certain consecutive blocks of β''.)

It is clear that β' is equivalent to β''. □

Remark 5.2.1 Let $\mathcal{P} = \{P_1, \dots, P_s\}$ be a partition on the set $\{1, \dots, v\}$ with $P_1 = \{i_{1,1}, \dots, i_{1,u_1}\}$, $P_2 = \{i_{2,1}, \dots, i_{2,u_2}\}, \dots, P_s = \{i_{s,1}, \dots, i_{s,u_s}\}$ from the step 3 of Algorithm 9. The permutation $\mu \in \mathfrak{S}_v$ from Proposition 5.2.1 is given by

$$\begin{pmatrix} 1 & 2 & \dots & u_1 & u_1+1 & \dots & u_1+u_2 & \dots & (u_1+u_2+\dots+u_s) \\ i_{1,1} & i_{1,2} & \dots & i_{1,u_1} & i_{2,1} & \dots & i_{2,u_2} & \dots & i_{s,u_s} \end{pmatrix}$$

and the corresponding partition is

$$\begin{aligned} \mathcal{R} := \big\{ R_1 = \{1, 2, \dots, u_1\}, R_2 = \{u_1+1, \dots, u_1+u_2\}, \dots, \\ R_s = \{u_1 + \dots + u_{s-1} + 1, \dots, u_1 + \dots + u_s\} \big\}. \end{aligned}$$

An important consequence of Proposition 5.2.1 is the construction of the Algorithm 10 which allows efficient factorization with respect to the FT logarithmic signature β.

Let ε^* be the randomized canonical signature created after step 1 of Algorithm 9. Also, let μ be the permutation with corresponding partition $\mathcal{R}$ from Remark 5.2.1. Then, we may efficiently factorize $\breve{\beta}(x)$ using the following algorithm.

Algorithm 10 Factorization with FT signature β

```
Input: $y$, $\varepsilon^*$, $\mu$, $\mathcal{R} = \{R_1, \dots, R_s\}$, $\xi$, $\pi_1, \dots, \pi_s, \rho$
Output: $x = x_1 \| x_2 \| \dots \| x_s$, where $y = \breve{\beta}(x)$
1: Compute $z = (y^{\rho^{-1}})$ and write $z = z_1 \| z_2 \| \dots \| z_\nu$. Each $z_i$ is of bit length $k_i$.
2: Factorize $z$ with respect to $\varepsilon^*$ by using Algorithm 11. Let denote $j'_1, \dots, j'_\nu$ the indices obtained by this factorization.
3: Compute $j_\ell = j'_{\mu^{-1}(\ell)}$ for $\ell = 1, \dots, \nu$.
4: According to $R_\ell = \{i_1, i_2, \dots, i_{u_\ell}\}$ set $x'_\ell = j_{i_1} \| j_{i_2} \| \dots \| j_{i_{u_\ell}}$ for $\ell = 1, \dots, s$.
5: Compute $x''_\ell = \pi_\ell^{-1}(x'_\ell)$ and finally $x_\ell = x''_{\xi^{-1}(\ell)}$ for $\ell = 1, \dots, s$.
```

In the following, we present an algorithm for factorization with respect to an NFT logarithmic signature. To make the description clearer, we start with an algorithm for factorization with respect to a randomized canonical logarithmic signature ε^* generated in step 1 of Algorithm 9.

Let $x = x_1 \| x_2 \| \dots \| x_\nu$ be a binary vector of length m, where x_i is of length k_i for $i = 1, \dots, \nu$ and $t_i = 2^{k_i}$. Let $y = \breve{\varepsilon}^*(x)$. Write $y = y_1 \| y_2 \| \dots \| y_\nu$, where each y_i is of bit length k_i.

In order to factorize y with respect to ε^* we have to determine indices x_i, for $i = 1, \dots, \nu$. This can be done using the following algorithm.

Algorithm 11 Factorization with ε^*

```
Input: $y = y_1 \| y_2 \| \dots \| y_\nu$, $\varepsilon^*$.
Output: $x = x_1 \| x_2 \| \dots \| x_\nu$, where $y = \breve{\varepsilon}^*(x)$.
1: Starting with $y_\nu$ we find an element $e^*_{\nu,j}$ in block $E^*_\nu$ such that the last $k_\nu$ bits of $e^*_{\nu,j}$
   are equal to $y_\nu$. Such $e^*_{\nu,j}$ is uniquely determined since the last $k_\nu$ bits of elements in
   $E^*_\nu$ form a vector space of dimension $k_\nu$. The index $j$ of $e^*_{\nu,j}$ in block $E^*_\nu$ determines the
   index $x_\nu$.
2: Compute $y' = y * (e^*_{\nu,j})^{-1}$ and write $y' = y'_1 \| y'_2 \| \dots \| y'_{\nu-1}$ where each $y'_i$ is of bit
   length $k_i$. Repeat step 1 with $y'_{\nu-1}$ for block $E^*_{\nu-1}$ to find $x_{\nu-1}$. Continue this process
   until $x_1$ is found.
```

Again, let $x = x_1 \| x_2 \| \dots \| x_\nu$ be a binary vector of length m where x_i is of bit length k_i for $i = 1, \dots, \nu$ and $t_i = 2^{k_i}$. Let $z = \breve{\beta}^*(x)$. Write $z = z_1 \| z_2 \| \dots \| z_\nu$ where each z_i is of bit length k_i. We describe an algorithm for factorization with respect to an NFT logarithmic signature β^*.

Algorithm 12 Factorization with NFT signature β^*

```
Input: $z = z_1 \| z_2 \| \dots \| z_\nu$, $\beta^*$, $\xi$, $\pi_1, \dots, \pi_\nu$, $\rho$.
Output: $x = x_1 \| x_2 \| \dots \| x_\nu$, where $z = \breve{\beta}^*(x)$.
1: Using $\xi^{-1}$, $\pi_1^{-1}, \dots, \pi_\nu^{-1}$ and $\rho^{-1}$ construct $\varepsilon^*$ from $\beta^*$.
2: Compute $y = (z^{\rho^{-1}})$ and write $y = y_1 \| y_2 \| \dots \| y_\nu$. Each $y_i$ is of bit length $k_i$.
3: Factorize $y$ with respect to $\varepsilon^*$ by using Algorithm 11. Let denote $x'_1, \dots, x'_\nu$ the indices
   obtained by this factorization.
4: Compute $x''_i = \pi_i^{-1}(x'_i)$ and finally $x_i = x''_{\xi^{-1}(i)}$ for $i = 1, \dots, \nu$.
```

In Appendix A, we give a toy example showing the generation of factorizable logarithmic signature β for $\mathcal{Z}$ and how to factorize with respect to it. We also show how to use it to set-up MST_3 with a small Suzuki 2-group.

5.3 Attack on private key

In this section, we investigate various types of possible direct attacks on the private key of MST_3. We aim to find lower bounds on the workload with respect to those attacks. It turns out that those bounds are very large in terms of the order of the groups used.

We denote the elements of the public key $\alpha := (a_{i,j})$, $\gamma := (h_{i,j})$, known to the adversary by pairs $S(a_{(i,j).a},\ a_{(i,j).b})$, and $S(h_{(i,j).a},\ h_{(i,j).b})$ respectively. Similarly, the private key elements $\beta := (b_{i,j})$, $(t_0, \ldots, t_s)$ are denoted by pairs $S(b_{(i,j).a},\ b_{(i,j).b})$, and $S(t_{(i).a},\ t_{(i).b})$.

5.3.1 Logarithmic signatures for $\mathcal{Z}$ and their two sided transformations

First, we remark that if the adversary attempts to extract information about $\beta = (b_{i,j})$, a main part of the private key, it is sufficient for him to obtain a logarithmic signature β' *equivalent* to β, i.e. any β' which is a *sandwich* transform of β. A stronger result in Section 4.4.2 shows that it is even sufficient for the adversary to break the system if he is able to determine a logarithmic signature β^* for $\mathcal{Z}$ such that

$$\breve{\beta}^*(x) = \breve{\beta}(x) \cdot c \qquad (5.3.1)$$

for all $x \in \mathbb{Z}_{|\mathcal{Z}|}$, where $c \in \mathcal{Z}$ is a fixed element. For example, if $\beta^* = [B_1^*, \cdots, B_s^*]$ with $B_i^* = z_{i-1}^{-1} \cdot B_i \cdot z_i$ is a two sided transformation of β with $z_0, z_1, \cdots, z_s \in \mathcal{Z}$, then $\breve{\beta}^*(x) = \breve{\beta}(x) \cdot c$, where $c = z_0^{-1} \cdot z_s$.

The result shows a fact relevant to the way of counting the number of elements t_i used in generating γ. In fact, if we replace t_i by $t_i^* = t_i \cdot z_i$, for $z_i \in \mathcal{Z}$, $i = 0, \ldots, s$, we obtain a β^* such that $\breve{\beta}^*(x) = \breve{\beta}(x) \cdot z_0^{-1} \cdot z_s$. Consequently, the adversary only needs to know the cosets of $\mathcal{Z}$ in $\mathcal{G}$ with coset representatives t_i's. Then (s)he can use any coset representative $t_i^* = t_i \cdot z_i$ in place of t_i. Hence, in the security analysis of the system, it suffices to determine the cosets of t_i with respect to $\mathcal{Z}$ and not the element t_i itself.

We call a logarithmic signature β^* for $\mathcal{Z}$ satisfying (5.3.1) a *translation* of β.

Definition 5.3.1 *Let* $\mathcal{K} = [\beta, f, t_0, \ldots, t_s]$ *be a private key for* MST_3. *We say that key* $\mathcal{K}' = [\beta', f, t_0', \ldots, t_s']$ *is* **equivalent** *to* $\mathcal{K}$ *if* β' *is a translation of* β *and* $t_i' = t_i \cdot z_i$ *for some* $z_i \in \mathcal{Z}$ *and all* $i \in \{0, \ldots, s\}$.

Our aim is to prove lower bounds on the work effort required for recovering an equivalent private key. The workload is measured in terms of the size of the involved groups and we will apply heuristic and algebraic methods to this analysis.

Now, the adversary attempts to extract information about the private key from the public knowledge of $\alpha = (a_{i,j})$ and $\gamma = (h_{i,j})$.

By this attack, as adversary, we try to construct a key $\mathcal{K}' = [\beta', f, t'_0, \ldots, t'_s]$ equivalent to the private key $\mathcal{K} = [\beta, f, t_0, \ldots, t_s]$. We first build an equation with unknowns involving information about the private key and then investigate the complexity of solving this equation. For this purpose we particularly exploit the operation (multiplication) in the underlying Suzuki 2-groups.

5.3.2 Building an equation

For convenience recall that

- $S(a_1, b_1)S(a_2, b_2) = S(a_1 + a_2\ ,\ b_1 + b_2 + a_1 a_2^\theta)$
- $t_i \in \mathcal{G},\ \ t_i = S(t_{(i).a}\ ,\ t_{(i).b}),\ \ \alpha := (a_{i,j}),\ \ a_{i,j} = S(a_{(i,j).a}\ ,\ a_{(i,j).b})$.
- $g = S(g_{.a}, g_{.b}) \in \mathcal{G},\ \ f(g) := S(0, g^\sigma_{.a})$ where $\sigma \in \mathrm{Aut}(\mathcal{Z}) = \mathrm{GL}(m, 2)$.

Further recall that $g\mathcal{Z} = h\mathcal{Z}$ in $\mathcal{G}$ with $g = S(x_1, x_2)$ and $h = S(y_1, y_2)$, if and only if $x_1 = y_1$.

We start with

$$\gamma = (h_{i,j}) = (t_{i-1}^{-1}\ a_{i,j}\ b_{i,j}\ f(a_{i,j})\ t_i) = (S(h_{(i,j).a}\ ,\ h_{(i,j).b}))$$

and focus on one block of γ. W.l.o.g., let us consider the first block. The elements in this block are $h_{1,1}, h_{1,2}, \cdots, h_{1,r_1}$. Let $J \subseteq \{1, \ldots, r_1\}$ be a subset such that $|J|$ is even. Then, if we sum up the elements of the first block having indices in J, we obtain the following two expressions corresponding to the "$.a$ part" and "$.b$ part" of the sum.

$$\sum_{j\in J,\ |J| even} h_{(1,j).a} = \sum_{j\in J} a_{(1,j).a} \tag{5.3.2}$$

$$\begin{aligned} \sum_{j\in J,\ |J| even} h_{(1,j).b} &= \sum_{j\in J} a_{(1,j).b} + \sum_{j\in J} b_{(1,j).b} + \sum_{j\in J} a^\sigma_{(1,j).a} \\ &\quad + t_{(0).a}\cdot \sum_{j\in J} a^\theta_{(1,j).a} + t^\theta_{(1).a}\cdot \sum_{j\in J} a_{(1,j).a} \end{aligned} \tag{5.3.3}$$

Adding $\sum_{j\in J} a_{(1,j).b}$ to both sides of (5.3.3) results in the equation

$$\begin{aligned}\sum_{j\in J} a_{(1,j).b} + \sum_{j\in J} h_{(1,j).b} &= \sum_{j\in J} b_{(1,j).b} + \sum_{j\in J} a^{\sigma}_{(1,j).a} \\ &\quad + t_{(0).a}. \sum_{j\in J} a^{\theta}_{(1,j).a} + t^{\theta}_{(1).a}. \sum_{j\in J} a_{(1,j).a} \end{aligned} \tag{5.3.4}$$

Note that the left side of Equation (5.3.4) is known.

From $h_{(1,1).a} = t_{(0).a} + a_{(1,1).a} + t_{(1).a}$ we obtain

$$t_{(0).a} = h_{(1,1).a} + t_{(1).a} + a_{(1,1).a}$$

Replacing $t_{(0).a}$ in (5.3.4) yields

$$\begin{aligned}\sum_{j\in J,\ |J| even} (a_{(1,j).b} + h_{(1,j).b}) = & \sum_{j\in J} b_{(1,j).b} + \sum_{j\in J} a^{\sigma}_{(1,j).a} \\ & + \Big(a_{(1,1).a} + t_{(1).a} + h_{(1,1).a}\Big). \sum_{j\in J} a^{\theta}_{(1,j).a} \\ & + t^{\theta}_{(1).a}. \sum_{j\in J} a_{(1,j).a}\end{aligned}$$

Considering $t_{(1).a}$ as an unknown we end up with a trinomial of the form

$$At^{\theta}_{(1).a} + Bt_{(1).a} + X = 0 \tag{5.3.5}$$

where

$$\begin{aligned} A &= \sum_{j\in J} a_{(1,j).a} \\ B &= \sum_{j\in J} a^{\theta}_{(1,j).a} \\ X &= \sum_{j\in J} a_{(1,j).b} + \sum_{j\in J} h_{(1,j).b} + \sum_{j\in J} b_{(1,j).b} + \sum_{j\in J} a^{\sigma}_{(1,j).a} \\ &\quad + (a_{(1,1).a} + h_{(1,1).a}). \sum_{j\in J} a^{\theta}_{(1,j).a}\end{aligned}$$

We should remark that the term $(t_{(1).a})^{\theta}$ in the trinomial expresses the action of θ on element $t_{(1).a} \in \mathbb{F}_q$. Since θ is an automorphism of $\mathbb{F}_q$ with $q = 2^m$, it can be written as a power of the Frobenius automorphism $\phi :\ a \to a^{\phi} = a^2$ of $\mathbb{F}_q$. Thus the term $(t_{(1).a})^{\theta}$ becomes $(t_{(1).a})^{2^n}$, if $\theta = \phi^n$ for some $1 \leqslant n < m$.

Note that A and B are known but the term X contains two unknown sums $\sum_{j\in J} b_{(1,j).b}$, and $\sum_{j\in J} a^{\sigma}_{(1,j).a}$.

5.3.3 Analysis of the equation

The aim of the adversary is to extract information about β. As in Equation (5.3.5) the value of X is unknown, the adversary has to guess a value for $t_{(1).a}$. There are $(q-1)$ possible choices for $t_{(1).a}$.

Having guessed a value for $t_{(1).a}$, the adversary can compute a corresponding value for X from Equation (5.3.5). In particular, (s)he can subsequently compute

$$C_J := \sum_{j\in J} b_{(1,j).b} + \sum_{j\in J} a^{\sigma}_{(1,j).a}. \tag{5.3.6}$$

It is important to note that in (5.3.6) both sums $\sum_{j\in J} b_{(1,j).b}$, $\sum_{j\in J} a^{\sigma}_{(1,j).a}$ remain unknown. For the sake of simplicity define

$$\begin{aligned} b_J &:= \sum_{j\in J} b_{(1,j).b} \\ a^{\sigma}_J &:= \sum_{j\in J} a^{\sigma}_{(1,j).a}. \end{aligned} \tag{5.3.7}$$

Thus

$$C_J = b_J + a^{\sigma}_J, \tag{5.3.8}$$

where the values of b_J and a^{σ}_J are not determined. Note that we have to determine σ to recover values b_J and thus gain partial information about β. On the other hand, knowing β would lead to reconstructing σ.

Attack on b_J

By this attack the adversary seeks to determine a value for b_J in order to get an equation of the form $a^{\sigma}_J = C_J - b_J$ for σ. Note here that a_J is known. (S)he will try constructing a system of those linearly independent equations and then attempt to solve the system to determine σ. Now as the elements in the first block $B_1 = [b_{11}, \ldots, b_{1r_1}]$ of β are not known, (s)he needs to guess a value for b_J for a given even subset J. As each b_J can take on any value from $\mathbb{F}_q$, where $q = 2^m$, and as the adversary needs at least m equations to reconstruct σ, this approach leads to a complexity of size $\mathcal{O}(q^m)$. Obviously, this type of brute force attack is not feasible as q is large.

Attack on a_J

We describe a more subtle and involved attack using Equation (5.3.8) on the first block of γ. The attack is described by the following algorithm.

Algorithm 13 Attack on a_J

```
1: Determine subsets $J \subseteq \{1, \ldots, r_1\}$ of even size such that $a_J = 0$ and collect equations $b_J = C_J$.
2: Try to solve a system of equations from step 1 for a set of unknown $D_1 \subseteq B_1$.
3: Let $D_1 = \{b_{1,j_1}, \ldots, b_{1,j_t}\}$. Use the $b_{1,j_i} \in D_1$ from step 2 to build non-trivial equations of the form $a^{\sigma}_{J^*} = d_{J^*}$, where $d_{J^*} := C_{J^*} - b_{J^*}$ is known and $J^* \subseteq \{j_1, \ldots, j_t\}$ is a subset of even size. Then solve the system of these equations to determine $\sigma$.
```

We observe that in order to apply this attack the block size of B_1 should satisfy: $r_1 > m$. If this is not the case, we have to fuse block B_1 and $B_2, \ldots, B_\ell$ (i.e. $B_1 \otimes B_2 \otimes \ldots \otimes B_\ell$), to form a larger block satisfying the condition. So, for the rest of the analysis of Algorithm 13 we implicitly assume that $r_1 > m$.

Before we go into a detailed analysis of Algorithm 13, it is worth mentioning that if $a_J = 0$ then $a^{\sigma}_J = 0$. An equation $a_J = 0$ does not give any information about σ, however it does yield an equation for b_J, namely $b_J = C_J$.

We now examine the complexity of the three steps of Algorithm 13.

1: As $a_{(1,j).a}$'s are known, the best known efficient way of determining $a_J = 0$ for a certain subset J is to use the birthday attack. More precisely, take two disjoint random subsets J_1 and J_2 of $\{1, \ldots, r_1\}$ such that $|J_1 \cup J_2|$ is even and check whether $a_{J_1} = a_{J_2}$. If this is so, an $a_J = 0$ has been found, where $J = J_1 \cup J_2$. Such a subset J gives rise to an equation $b_J = C_J$. Finding a subset J with $a_J = 0$ by the birthday attack has a complexity of size roughly $\mathcal{O}(q^{1/2})$. Note that in step 1 each even subset J has size at least four. This is because all elements in each block of α belong to distinct cosets modulo $\mathcal{Z}$, i.e $a_{(1,j).a} \neq a_{(1,h).a}$ for $h \neq j$. Of course, the assumption $\sum\limits_{j \in \{1,\ldots,r_1\}} a_{(1,j).a} = 0$ is taken into account. We discuss this condition in the remark below.

2: Let $\mathcal{P} = \{J_0 = \emptyset, J_1, \ldots, J_w\}$ where $J_i \subseteq \{1, \ldots, r_1\}$ is a subset of even size with $|J_i| \geqslant 4$ such that $a_{J_i} = 0$ for $i \geqslant 1$. Let $\bigcup_{i=0}^{w} J_i = \{j_1, \ldots, j_t\}$. Each subsum $a_J = 0$ from step 1 corresponds to an equation $b_J = C_J$. The unknowns of these equations are elements $b_{1,j_1}, \ldots, b_{1,j_t}$ of B_1. Let

$$E_{\mathcal{P}} = \{b_J = C_J : J \in \mathcal{P}\}.$$

Since there are t unknowns, we can view the coefficients of each equation in $E_{\mathcal{P}}$ as a vector in $\mathbb{F}_{2^t}$, viewed as a vector space of dimension t over $\mathbb{F}_2$. Each such 0-1 vector has an even Hamming weight of size at least 4. Any linear combination of two such vectors gives rise to a vector corresponding to a subsum $a_J = 0$ with $J \in \mathcal{P}$ and hence to an equation in $E_{\mathcal{P}}$. In other words, the coefficient vectors of the equations in $E_{\mathcal{P}}$ span a linear subspace V of $\mathbb{F}_{2^t}$, where each non-zero vector of V has a weight at least 4. And therefore the dimension of V is at most $t-3$. This is equivalent to saying that by using elementary row operations the coefficient matrix, say M, of any system of equations from $E_{\mathcal{P}}$ will be transformed into a matrix of row echelon form, for which each row necessarily has a weight at least 4. Hence, such a system of equations gives rise to at least 3 parameters that can be freely chosen, i.e. the rank of M, denoted by $\mathsf{rank}(M)$, is at most $t-3$. Since each parameter can take on any value from $\mathbb{F}_q$, solving equations for $b_{1,j} \in D_1$ in this step requires a complexity of size at least $\mathcal{O}(q^3)$. Having an accurate estimate of the $\mathsf{rank}(M)$ appears to be a difficult problem. This is because the rank of M depends on the set $\mathcal{P}$, which in turn depends on the random values of $a_{(1,i_1).a}, \ldots, a_{(1,i_t).a}$.

Note that $t \geqslant 4$. If $t = 4$, we have $\mathsf{rank}(M) = t - 3 = 1$. This fact is easy to see since $J^* = \{j_1, j_2, j_3, j_4\}$ is the only non-empty subset with $a_J = 0$. Consequently $b_{1,j_1} + b_{1,j_2} + b_{1,j_3} + b_{1,j_4} = C_J$ is the only possible equation with 4 unknowns we can obtain.

If $t > 4$, we can prove an even stronger bound that $\mathsf{rank}(M) \leqslant t - 4$. As above we denote by V the linear subspace of $\mathbb{F}_{2^t}$ spanned by the coefficient vectors of the equations in $E_{\mathcal{P}}$. If any vector of V has the weight at least 6, the dimension of V is at most $t-5$. And therefore $\mathsf{rank}(M) \leqslant t-5 < t-4$. So, we assume that V contains a vector v of weight 4. Without loss of generality, we can assume that v is of the form $v = 111100\ldots0$ (just by renaming the unknowns). Consider $w_4 = 111110\ldots0 \in \mathbb{F}_{2^t}$. Let $w_1 = 1000\ldots0$, $w_2 = 0100\ldots0$, $w_3 = 0010\ldots0$. Then $w_1, w_2, w_3, w_4 \notin V$. Let W be the subspace of $\mathbb{F}_{2^t}$ spanned by w_1, w_2, w_3, w_4. Then W has dimension 4. It can be checked that $x + v$ has weight at most 3 for 14 non-zero vectors $x \in W$, i.e. $x \notin V$, except for $x = y = 000110\ldots0 \in W$. But $y \notin V$, as its weight is 2. So we have $W \cap V = \{0\}$. Hence the dimension of V is at most $t-4$. Consequently, $\mathsf{rank}(M) \leqslant t-4$. In order to continue the attack we need to guess the values for at least 4 unknowns $b_{1,j} \in \mathbb{F}_q$. Therefore the complexity of step 2 in this case is at least $\mathcal{O}(q^4)$.

3: Let $D_1 = \{b_{1,j_1}, \ldots, b_{1,j_t}\}$ be the subset determined after step 2. In order to be able to recover $\sigma \in \mathsf{GL}(m, 2)$ it is necessary that $t \geqslant m$. Using elements in D_1 the adversary can construct non-zero subsum $a_J^\sigma = C_J - b_J \neq 0$ from Equation (5.3.6) and try to solve such a system of equations to recover σ. This can be done in polynomial time.

Note that $t \geqslant m > 4$. We record the result of this attack in the following proposition.

Proposition 5.3.1 *The complexity required to recover a key equivalent to the private key* $[\beta, f, t_0, \ldots, t_s]$ *by using Algorithm 13 amounts to a size at least* $\mathcal{O}(q^5.q^{1/2})$.

This complexity is composed by

- the complexity $\mathcal{O}(q)$ of selecting a correct value for $t_{(1).a}$ in trinomial (5.3.5),
- the complexity $\mathcal{O}(q^{1/2})$ of the birthday attack in step 1 and the complexity of size at least $\mathcal{O}(q^4)$ of solving equations for $b_{1,j}$'s in step 2.

It is a challenging open problem to determine a better lower bound on the workload to recover the private key of the system. The task appears to be difficult.

Remark 5.3.1 We observe that the upper bound for the rank of the matrix M obtained in step 2 above is far from its actual value since, to simplify the discussion, we did not impose any restriction on $a_{1,j}$'s, i.e. in the argumentation we freely use all possible values for $a_{1,j}$'s, when we estimate the dimension of V. Evidently, the dimension of V depends on the choice of $a_{1,j}$'s. One can expect that the dimension of V is much smaller and so is the rank of M. Therefore the complexity of the attack on a_J is much higher than $\mathcal{O}(q^5.q^{1/2})$. We conjecture that the values $t - \mathrm{rank}(M) - 1$ increase in proportion to the growth of t (i.e. $\mathrm{rank}(M)$ becomes proportionally smaller, when t becomes larger).

Remark 5.3.2 In step 2 of Algorithm 13 the assumption $\sum_{j\in\{1,\ldots,r_1\}} a_{(1,j).a} = 0$ is taken into account. If this condition is removed, we shall have $\sum_{j\in\{1,\ldots,r_1\}} a_{(1,j).a} \neq 0$ in general. Suppose that we guess a value for $u = \sum_{j\in\{1,\ldots,r_1\}} b_{(1,j).b}$. This can be done with complexity $\mathcal{O}(q)$. Consequently, each subsum $a_J = 0$ obtained from the birthday attack likely yields $C_K - (u - b_J) = a_K = \sum_{j\in K} a_{(1,j).a} \neq 0$, where $K = \{1, \ldots, r_1\} \setminus J$. Each $a_K \neq 0$ corresponds to a non-trivial linear equation for σ. So, if the adversary would collect m linearly independent equations, (s)he could reconstruct σ as in step 3. In this case, the complexity of recovering a key equivalent to the private key $[\beta, f, t_0, \ldots, t_s]$ would reduce to $\mathcal{O}(q^2.q^{1/2})$.

More on Attack on a_J

Consider the following modification of the previous attack.

Algorithm 14 Modified attack on a_J

```
1: Choose elements $t_0$ and $t_s$. Knowing that logarithmic signature $\beta'$ equivalent to $\beta$ can be used to attack the scheme, $t_0'$ and $t_s'$ have to be chosen from the cosets $t_0\mathcal{Z}$ and $t_s\mathcal{Z}$. The workload required in this step is bounded by $\mathcal{O}(q^2)$.
2: From the public key $[\alpha, \gamma]$ compute cover $\gamma' := (h'_{i,j})$ with $h'_{i,j} = t_{i-1} \cdot h_{i,j} \cdot t_i^{-1}$. Then $h'_{i,j} = f(a_{i,j}) \cdot b_{i,j}$.
3: Search ciphertexts $x$ with $\breve{\alpha}(x)_{.a} = 0$ (using the Birthday attack), i.e. $f(\breve{\alpha}(x)) = id$ and therefore $\breve{\gamma}'(x) = \breve{\beta}(x)$. Each such an equation gives a non-trivial equation with unknown $b_{i,j}$. The workload required in this step is bounded by $\mathcal{O}(\sqrt{q})$.
4: Similar to the previous attack, collect enough equations from step 3 and try to solve the system of these equations to determine logarithmic signature $\beta$.
```

Remark 5.3.3 The attack can also be applied on a subset of blocks of α and γ. For example, to attack the first three blocks, we choose t_0 and t_3 in step 1 and reduce $\alpha := [A_1, \dots, A_s]$ to $\alpha' := [A_1, A_2, A_3]$, $\gamma := [H_1, \dots, H_s]$ to $\gamma' := [H_1, H_2, H_3]$, and cleartext $x = (j_1, \dots, j_s)$ to $x' = (j_1, j_2, j_3)$. Then the attack can only recover $b_{i,j}$'s from blocks B_1, B_2, B_3 and has to be repeated for the remaining blocks.

As the number of collected equations in step 4 depends on the randomly generated elements of the cover α, determining how many such linearly independent equations can be found in general seems to be a rather difficult problem. Therefore, we consider the following computer experiment. We repeatedly apply the three steps of Algorithm 15 below with covers of various types and for various sizes of the underlying group. From the experimental results, we obtain a conjecture about the lower bound for the workload required for the Algorithm 14.

Let $q = 2^m$ and $\mathcal{G} = A(m, \theta)$. Let $\alpha := (a_{i,j})$ be a random cover for $\mathcal{G}$ of type $(r_1, \dots, r_s)$.

Algorithm 15 Modified attack on a_J: Experiment

```
1: Generate random $\alpha$.
2: Use Birthday attack on randomly generated inputs to find a set $U$ of $x^{(u)} = (j_1^{(u)}, \ldots, j_s^{(u)})$ such that
       $$\sum_{x^{(u)} \in U} \breve{\alpha}(x^{(u)})_{.a} = \sum_{x^{(u)} \in U} \sum_{i=1}^{s} a_{(i,j_i^{(u)}).a} = 0.$$
   Construct a vector $b$ with ones on positions $j_i$ and zeros elsewhere.
3: Collect maximal possible linearly independent vectors $b$ from step 2. Construct a matrix $B$ with vectors $b$ as rows.
```

In regard to the attack in Algorithm 14, the number of columns in B gives the number of (unknown) elements $b_{i,j}$ in β. If no full rank matrix can be found, the elements $b_{i,j}$ cannot be uniquely determined. Set $c = \mathtt{length\ of}\ \beta - \mathtt{rank}(B)$, then the workload required to determine β in step 4 of Algorithm 14 is $\mathcal{O}(q^c)$.

From the experiment we obtain the following result. Let $\ell = \sum_{i=1}^{s} r_i$ and $n = \sum_{i=1}^{s} k_i$ where $k_i = \log_2 r_i$ for $i = 1, \ldots, s$.

(a) If $m \geqslant n$ then $\mathtt{rank}(B) \leqslant \ell - n - s + 1$

(b) If $m < n$ then $\mathtt{rank}(B) \leqslant \ell - n - s + 1 + (n - m)$

Note that in realization of MST_3 the type of β is chosen in general such that $m = n$. As α and β are of the same type,

$$c \geqslant \ell - \mathtt{rank}(B) = s - 1 + \sum_{i=1}^{s} k_i$$

Clearly, we achieve a better result by attacking a single block, say B_i, in a time. The workload required to recover B_i is lower bounded by $\mathcal{O}(q^{k_i})$. We may further reduce this value by attacking only a subset of B_i, i.e. we restrict the pointers j_i to lie in a chosen $\mathcal{J} \subset B_i$. However, this subset must hold enough elements to contain collision(s) searched by the Birthday attack in step 2. The smallest subset U for the attack on a single block contains three elements (due to the restriction on the choice of elements of single block of α from different cosets modulo $\mathcal{Z}$). Such a U can lead to at most one equation, i.e. the workload required to solve the system of equations for three elements of B_i on positions j_i from $\mathcal{J}$ is lower bounded by $\mathcal{O}(q^2)$. If the size of U is small, we may not be able to determine all elements $b_{i,j}$, and as shown in the discussion above, a larger set U increases the workload considerably, however. In summary, the workload for the whole attack from Algorithm 14 is lower bounded by $\mathcal{O}(q^{4.5})$. This is a rather rough and underestimate result, in practice the complexity of the workload could be much higher.

An Example

Let $m = 160$ and $(256, 256, \dots, 256)$ be the type of β with altogether 20 blocks. Using Algorithm 14, i.e. attacking all blocks at once, we may collect at most 4941 from 5120 linearly independent equations in step 4. Hence the complexity of recovering β is $\mathcal{O}(q^{179})$. Attacking the blocks one by one, the workload required to recover β is reduced to $\mathcal{O}(q^{8})$. An attack on 1/2 of block elements at once can be done with $\mathcal{O}(q^{7})$, etc..

Combined Attack on b_J and a_J

We can envisage a further method of reconstructing σ from equation $a_J^{\sigma} + b_J = C_J$. Two main steps of the following algorithm describe this attack.

Algorithm 16 Attack on b_J and a_J

```
1: Construct $2m$ linearly independent vectors of size $2m$ over $\mathbb{F}_2$ to form an $2m \times 2m$ regular binary matrix $A$. Each row of $A$ is of the form $a_J || b_J$, ($||$ denotes concatenation), where $a_J$ and $b_J$ are considered as vectors of length $m$ over $\mathbb{F}_2$.
2: Let $M$ denote the $2m \times m$ matrix, whose rows are $C_J$. Observe that $M$ is known after $t_{(1).a}$ has been chosen. Compute a $2m \times m$ binary matrix $X$ such that $A.X = M$, i.e. $X = A^{-1}.M$.
```

Let us take a closer look at Algorithm 16. We write

$$X := \begin{pmatrix} \sigma^* \\ Y \end{pmatrix}$$

and σ^* and Y are $m \times m$ binary matrices. First observe that any matrix A constructed in step 1 yields a matrix $X = A^{-1}.M$, as M is known. Each row of A takes on a value $a_J || b_J$ corresponding to an index subset J of even size. The first part a_J can be computed, because $a_{(1,j).a}$'s are known, but we have to guess a value for b_J (an m bit vector) from the unknowns $b_{(1,j)}$'s, since they are part of the private key. So, there are q possible choices for each row of A corresponding to q possible values for b_J. If all $2m$ rows of A are correctly selected (i.e. each value of b_J is guessed correctly), the matrix X will have the form

$$X := \begin{pmatrix} \sigma \\ I_m \end{pmatrix},$$

where I_m is the $m \times m$ identity matrix. This implies that the complexity of a successful reconstruction of σ (i.e. $\sigma^* = \sigma$) after all $2m$ rows of A are determined is $\mathcal{O}(q^{2m})$. In this case, we have $Y = I_m$.

Remark 5.3.4 A pertinent implication of the combined attack is the following fact. If logarithmic signature $\beta = (b_{i,j})$ is of the form $\beta = (b_{i,j}) = (e_{i,j})^{\sigma_1}$, where $(e_{i,j})$ are known and σ_1 is an unknown $m \times m$ regular matrix over $\mathbb{F}_2$, this attack will enable to reconstruct σ and σ_1 as well. The reason can be seen as follows. Equation (5.3.8) can now be written as $a_J^\sigma + b_J = a_J^\sigma + e_J^{\sigma_1} = C_J$. If in step 1 we can construct a regular $2m \times 2m$ matrix A with rows of the form $a_J \| e_J$, the matrix $X = A^{-1}.M$ obtained from step 2 will have the form

$$X = \begin{pmatrix} \sigma \\ \sigma_1 \end{pmatrix},$$

i.e. we are able to recover σ and σ_1. We see that this is only possible because both $a_{(1,j).a}$'s, $e_{(1,j)}$'s are completely known.

We close the discussion of the security analysis of the direct attacks with a record of the obtained results.

Proposition 5.3.2 *Comparing the three attacks presented in this section, the strongest one, the Attack on* a_J*, provides an actual estimate of workload required for recovering a key equivalent to the private key. The workload is bounded below by* $\mathcal{O}(q^5.q^{1/2})$*, where* $q = \sqrt{|\mathcal{G}|}$.

Remark 5.3.5 Let $\alpha := (S(a_{(i,j).a}, a_{(i,j).b}))$ be a cover used in a set-up of MST_3 such that $a_{(i,j).a} \in \mathcal{H} < \mathcal{Z}$, where $\mathcal{H}$ is a subgroup of $\mathcal{Z}$ of order $q_0 = 2^\ell$. Then the lower bound given by Proposition 5.3.2 becomes $\mathcal{O}(q^4.q_0^{3/2})$. The bound is obtained because in the previous analysis the number of possible choices for $t_{(1).a}$ and the workload required for the birthday attack in step 1 of Algorithm 13 will be reduced according to the order of $\mathcal{H}$.

5.4 Attack on ciphertexts

This section deals with an elaborate chosen plaintext attack on MST_3 when transversal logarithmic signatures are used. This is the case when β is generated by Algorithm 9 without applying the fusion step 3. In fact, those logarithmic signatures may essentially be viewed as those from a chain of subgroups of $\mathcal{Z}$. However, the structure of β will be changed if the the fusion step 3 is applied.

The Matrix-permutation attack developed in this section appears to be powerful as it provides proof of the fact that the class of non-fused transversal logarithmic signatures cannot be used in a realization of MST_3. The class of fused transversal logarithmic signatures, however, withstands the Matrix-permutation attack, as shown below.

Before we present the Matrix-permutation attack, we would like to mention a simple attack which emerges naturally from the representation of the elements in the Suzuki 2-groups.

5.4.1 The Basis attack

Based on the description of the scheme, the $.a$ part of α and γ are merely random covers for $\mathbb{F}_q$. We identify $\mathbb{F}_q$ with vector space V of dimension m over $\mathbb{F}_2$. Let us denote by $\alpha_{.a}$ the cover of V whose blocks are formed by the $.a$ part of α, i.e. $\alpha_{.a} := (a_{(i,j).a})$. Define $\mathcal{J} := \breve{\alpha}_{.a}(\mathbb{Z}_q)$. Thus $\mathcal{J}$ is a subset of V and the ratio $\rho := q/|\mathcal{J}|$ may be viewed as the average number of representations for each element of $\mathcal{J}$ with respect to $\alpha_{.a}$. More precisely, due to the connection between the generation of random covers and the occupancy problem, see Section 3.6, we can derive an approximation for the ratio ρ given by the following formula

$$\rho \approx \lambda\left(\frac{e^{\lambda}}{e^{\lambda}-1}\right)$$

where $\lambda = \frac{1}{|V_1|}\prod_{i=1}^{s} r_i$, and $(r_1, \ldots, r_s)$ is the type of $\alpha_{.a}$, and V_1 is the smallest subspace of V containing $\mathcal{J}$. As a matter of linear algebra, we may find a maximal subset of linearly independent vectors which come from all the blocks of $\alpha_{.a}$. By using the two sided transformation on $\alpha_{.a}$ we may assume that the first $s-1$ blocks contain the zero vector. The linearly independent vectors together with the zero vectors form a cover which allows an efficient factorization of a certain number of ciphertexts created by $\breve{\alpha}_{.a}$. This amount is approximately $\frac{1}{\rho}\prod_{i=1}^{s}(k_i+1)$, where $k_i = \lceil \log_2 r_i \rceil$. Therefore the probability that a given ciphertext could be correctly decrypted is given by

$$\approx \frac{1}{\rho}\prod_{i=1}^{s}\frac{(k_i+1)}{r_i}$$

As a result, if ρ or/and r_i are increased, this probability will be decreased. So, if we select the elements of $\alpha_{.a}$ from a subspace V_1 of V such that $\rho = q/|V_1|$ is large, then this simple attack becomes infeasible.

Also, if α is chosen with blocks of appropriate size, the probability that the message can be decrypted using this method is negligible.

5.4.2 The Meet-in-the-middle attack

There exists a trivial brute force attack on any random cover δ, which for a given $y = \breve{\delta}(x)$, attempts to determine x by using a time and memory trade-off method. This type of attack is generally called the Meet-in-the-middle attack.

For MST_3 it is described as follows. As in the previous attack, let $\alpha_{.a} := (a_{(i,j).a})$ be a random cover of type $(r_1, \ldots, r_s)$ for V (vector space over $\mathbb{F}_q = \mathbb{F}_{2^m}$). Assume that $y_{.a} = \breve{\alpha}_{.a}(x)$ is given for some $x \leftrightarrow (j_1, \ldots, j_s)$.

So we write $y_{.a} = \breve{\delta}_1(x_1) \cdot \breve{\delta}_2(x_2)$ where $x_1 = (j_1, \ldots, j_{\lfloor s/2 \rfloor})$, $x_2 = (j_{\lfloor s/2 \rfloor + 1}, \ldots, j_s)$ and $\breve{\delta}_1(x_1) = \sum_{i=1}^{\lfloor s/2 \rfloor} a_{(i,j).a}$, $\breve{\delta}_2(x_2) = \sum_{i=\lfloor s/2 \rfloor+1}^{s} a_{(i,j).a}$. Here we have $\delta_1 := (a_{(i,j).a})$, $i = 1, \ldots, \lfloor s/2 \rfloor$; $j = 1, \ldots, r_i$ and $\delta_2 := (a_{(i,j).a})$, $i = \lfloor s/2 \rfloor + 1, \ldots, s$; $j = 1, \ldots, r_i$.

First construct a table T of all possible pairs (u, v) with $u = (u_1, \ldots, u_{\lfloor s/2 \rfloor})$, $u_i \in \mathbb{F}_{2^{r_i}}$, and $v = \breve{\delta}_1(u)$. The size of T is roughly $\mathcal{O}(\sqrt{q})$.

The attack works as follows: for each chosen $w = (u_{\lfloor s/2 \rfloor+1}, \ldots, u_s)$, $u_i \in \mathbb{F}_{2^{r_i}}$, compute a product $g = y_{.a} \cdot (\breve{\delta}_2(w))^{-1}$. If there is a pair (u, v) in T such that $g = v$, then we have $x = u \| w$, i.e. $x \leftrightarrow (u_1, \ldots, u_{\lfloor s/2 \rfloor}, u_{\lfloor s/2 \rfloor+1}, \ldots, u_s)$. On average, we need to construct $\mathcal{O}(\sqrt{q})$ of values for g until we obtain $g = v$.

In summary, this attack requires $\mathcal{O}(\sqrt{q})$ memory and $\mathcal{O}(\sqrt{q})$ time.

Note that if $\alpha_{.a}$ is constructed from subspace V_1 of V such that $\rho = q/|V_1|$ is large, the Meet-in-the-middle attack cannot be applied.

5.4.3 The Matrix-permutation attack on NFT-MST_3

We now present the Matrix-permutation attack on a realization of MST_3 that uses a non-fused transversal logarithmic signature β (for short we call NFT-MST_3). This strong attack is a chosen plaintext type attack which attempts to reverse the encryption function of the system. The main idea of the Matrix-permutation attack is to construct a series of matrices and to recover permutations used to generate β that would eventually allow the adversary to decrypt any given ciphertext.

Used notation

Let $\omega := (w_{i,j})$ be a cover of type $(r_1, \ldots, r_s)$ for $\mathcal{G}$, and let $x \in \mathbb{Z}_{|\mathcal{Z}|}$ correspond to $(j_1, \ldots, j_s) \in \mathbb{Z}_{r_1} \oplus \ldots \oplus \mathbb{Z}_{r_s}$ (see Chapter 2). Let $\xi \in \mathfrak{S}_s$ and $\nu_\ell := \xi(\ell)$ for $\ell \in \{1, \ldots, s\}$.

Define

$$\breve{\omega}_{k,\xi}(x) := \prod_{i=1}^{k} w_{\nu_i, j_{\nu_i}} \tag{5.4.1}$$

We consider an NFT-MST_3 scheme. Let $[\alpha, \gamma]$ be the public key with the corresponding private key $[\beta, f, t_0, \ldots, t_s]$. Recall that $\alpha := (a_{i,j_i})$, $\beta := (b_{i,j_i})$ and $\gamma := (h_{i,j_i})$ are of type $(r_1, \ldots, r_s)$ and that ξ is a permutation used in step 5 and $\pi_1, \ldots, \pi_s$ are permutations used in step 4 of Algorithm 9.

Proposition 5.4.1 *Let* α, β, γ *be the covers of type* $(r_1, \ldots, r_s)$ *as described above. Let* $x \in \mathbb{Z}_{|\mathcal{Z}|}$ *correspond to* $(j_1, \ldots, j_s) \in \mathbb{Z}_{r_1} \oplus \ldots \oplus \mathbb{Z}_{r_s}$ *and* $\nu_\ell := \xi(\ell)$ *for* $\ell \in \{1, \ldots, s\}$. *Further let* $\breve{\alpha}_{\ell,\xi}(x)$, $\breve{\beta}_{\ell,\xi}(x)$, $\breve{\gamma}_{\ell,\xi}(x)$ *be the values computed by Equation (5.4.1). Let* $k_\ell := \lceil \log_2 r_\ell \rceil$. *Then there exists a binary* $(2m+1) \times k_{\nu_\ell}$ *matrix* M_{ν_ℓ} *such that*

$$(\ \breve{\alpha}_{\ell,\xi}(x)_{.a} \parallel \breve{\alpha}_{\ell,\xi}(x)_{.b} + \breve{\gamma}_{\ell,\xi}(x)_{.b} \parallel 1\ )\ M_{\nu_\ell} = \pi_{\nu_\ell}(j_{\nu_\ell}). \tag{5.4.2}$$

where "1" *is the bit set to one.*

Proof: First we show that there exists a binary $(2m+1) \times m$ matrix N_{ν_ℓ} such that

$$(\ \breve{\alpha}_{\ell,\xi}(x)_{.a} \parallel \breve{\alpha}_{\ell,\xi}(x)_{.b} + \breve{\gamma}_{\ell,\xi}(x)_{.b} \parallel 1\ )\ N_{\nu_\ell} = \Big(\ \breve{\beta}_{\ell,\xi}(x)_{.b}\ \Big) \tag{5.4.3}$$

We begin with

$$\begin{aligned}\breve{\alpha}_{\ell,\xi}(x)_{.b} + \breve{\gamma}_{\ell,\xi}(x)_{.b} = &\sum_{i=1}^{\ell} b_{(\nu_i, j_{\nu_i}).b} + t_{(\nu_0).a} \sum_{i=1}^{\ell} a^{\theta}_{(\nu_i, j_{\nu_i}).a} + t^{\theta}_{(\nu_\ell).a} \sum_{i=1}^{\ell} a_{(\nu_i, j_{\nu_i}).a} \\ &+ \sum_{i=1}^{\ell} a^{\sigma}_{(\nu_i, j_{\nu_i}).a} + C_\ell\end{aligned}$$

where $C_\ell = t_{(\nu_0).b} + t^{\theta+1}_{(\nu_0).a} + t_{(\nu_\ell).b} + t_{(\nu_0).a} t^{\theta}_{(\nu_\ell).a}$. As the elements $t_{(\nu_0).a}, t_{(\nu_\ell).a} \in \mathbb{F}_q$ are constants, the products $t_{(\nu_0).a} \sum a^{\theta}_{(\nu_i, j_{\nu_i}).a}$ and $t^{\theta}_{(\nu_\ell).a} \sum a_{(\nu_i, j_{\nu_i}).a}$ present linear mappings. Therefore there exist binary $m \times m$ matrices T_{ν_0} and T_{ν_ℓ} such that

$$\begin{aligned} t_{(\nu_0).a} \sum_{i=1}^{\ell} a^{\theta}_{(\nu_i, j_{\nu_i}).a} &= \sum_{i=1}^{\ell} a_{(\nu_i, j_{\nu_i}).a}\ T_{\nu_0} \\ t^{\theta}_{(\nu_\ell).a} \sum_{i=1}^{\ell} a_{(\nu_i, j_{\nu_i}).a} &= \sum_{i=1}^{\ell} a_{(\nu_i, j_{\nu_i}).a}\ T_{\nu_\ell} \end{aligned}$$

Set

$$N_{v_\ell} = \begin{pmatrix} T_{v_0} + T_{v_\ell} + \sigma \\ I_m \\ C_\ell \end{pmatrix}$$

where I_m is the $m \times m$ identity matrix. Now, it is not difficult to check that the $(2m+1) \times m$ matrix N_{v_ℓ} satisfies Equation (5.4.3). Define $\varepsilon' := (e'_{i,j})$ with $e'_{i,j} = b_{i,j}^{\rho^{-1}}$. Clearly

$$\left(\breve{\beta}_{\ell,\xi}(x)_{.b} \right)^{\rho^{-1}} = \left(\breve{\varepsilon}'_{\ell,\xi}(x)_{.b} \right)$$

Consider the linear mapping φ_ℓ defined by

$$\breve{\varepsilon}^*_{\ell,id}(x)_{.b} = \sum_{i=1}^{\ell} e^*_{(i,j_i).b} \overset{\varphi_\ell}{\longmapsto} j_\ell$$

where id is the identity permutation.This mapping is well defined for the class of transversal logarithmic signatures, in particular for ε^* created in Algorithm 9 after step 1. Note that j_ℓ is the binary representation of the index for e_{ℓ,j_ℓ} and is identical with the k_ℓ bit vector of e_{ℓ,j_ℓ} at the positions K_ℓ. Let $\varepsilon'' := (e''_{i,j})$ be obtained from ε^* by applying step 4 of Algorithm 9. Now observe that φ_ℓ acts on $\breve{\varepsilon}''_{\ell,id}(x)_{.b}$ as follows

$$\breve{\varepsilon}''_{\ell,id}(x)_{.b} = \sum_{i=1}^{\ell} e''_{(i,j_i).b} \overset{\varphi_\ell}{\longmapsto} \pi_\ell(j_\ell)$$

Applying step 5 of Algorithm 9 on ε'' we get ε'. Therefore φ_ℓ acts on $\breve{\varepsilon}'_{\ell,\xi}(x)_{.b}$ according to

$$\breve{\varepsilon}'_{\ell,\xi}(x)_{.b} = \sum_{i=1}^{\ell} e'_{(v_i,j_{v_i}).b} \overset{\varphi_\ell}{\longmapsto} \pi_{v_\ell}(j_{v_\ell})$$

Let P_{v_ℓ} be the $m \times k_{v_\ell}$ binary matrix represention of the mapping φ_ℓ. Then we can write

$$\left(\breve{\varepsilon}'_{\ell,\xi}(x)_{.b} \right) P_{v_\ell} = \pi_{v_\ell}(j_{v_\ell})$$

Define the matrix M_{v_ℓ} as

$$M_{v_\ell} := N_{v_\ell} \cdot \rho^{-1} \cdot P_{v_\ell}$$

Then M_{ν_ℓ} is the binary matrix that satisfies Equation (5.4.2). □

Let $M_{\ell,p}$ denote the p-th column of the matrix M_ℓ, where $p = 1, \ldots, k_\ell$. We observe that $\pi_\ell(j_\ell)$ is a binary vector of length k_ℓ. Similarly, we denote $\pi_{\ell,p}(j_\ell)$ the p-th bit of $\pi_\ell(j_\ell)$.

By using this notation and Proposition 5.4.1, where ξ is defined, we obtain the following proposition.

Proposition 5.4.2 *Let* $\nu_\ell := \xi(\ell)$ *and* $M_{\nu_\ell,p}$ *be the* p*-th column of* M_{ν_ℓ} *and* $\pi_{\nu_\ell,p}(j_{\nu_\ell})$ *be the* p*-th bit of* $\pi_{\nu_\ell}(j_{\nu_\ell})$ *from Propositon 5.4.1. Then we have*

$$\left(\ \breve{\alpha}_{\ell,\xi}(x)_{.a} \parallel \breve{\alpha}_{\ell,\xi}(x)_{.b} + \breve{\gamma}_{\ell,\xi}(x)_{.b} \parallel 1\ \right) M_{\nu_\ell,p} = \pi_{\nu_\ell,p}(j_{\nu_\ell}). \tag{5.4.4}$$

Proposition 5.4.3 *Let* α, β, γ *be the covers of type* $(r_1, \ldots, r_s)$ *as described above. Let* $x \in \mathbb{Z}_{|\mathcal{Z}|}$ *correspond to* $(j_1, \ldots, j_s) \in \mathbb{Z}_{r_1} \oplus \ldots \oplus \mathbb{Z}_{r_s}$*. Further let* $\nu_\ell := \xi(\ell)$ *for* $\ell \in \{1, \ldots, s\}$ *and* $k_\ell := \lceil \log_2 r_\ell \rceil$*. Then there exists a binary* $(2m+1) \times k_{\nu_\ell}$ *matrix* L_{ν_ℓ} *such that*

$$\left(\ \breve{\alpha}(x)_{.a} + \mathcal{A}_\ell \ \Big\|\ \breve{\alpha}(x)_{.b} + \breve{\gamma}(x)_{.b} + \mathcal{B}_\ell \ \Big\|\ 1\ \right) L_{\nu_\ell} = \pi_{\nu_\ell}(j_{\nu_\ell}) \tag{5.4.5}$$

where

$$\begin{aligned}
\mathcal{A}_\ell &:= \sum_{i=\ell+1}^{s} a_{(\nu_i, j_{\nu_i}).a} \\
\mathcal{B}_\ell &:= \sum_{i=\ell+1}^{s} \left(a_{(\nu_i, j_{\nu_i}).b} + h_{(\nu_i, j_{\nu_i}).b} \right) + \sum_{i=\ell+1}^{s} a^{\theta}_{(\nu_i, j_{\nu_i}).a} \left(t_{(\nu_0).a} + t_{(\nu_{i-1}).a} \right) + \\
&\quad \sum_{i=\ell+1}^{s} a_{(\nu_i, j_{\nu_i}).a} \left(t_{(\nu_i).a} + t_{(\nu_s).a} \right)^{\theta}
\end{aligned}$$

for $\ell \in \{1, \ldots, s-1\}$*,* $\mathcal{A}_s = \mathcal{B}_s := (0, \ldots, 0)$*, and "1" is the bit set to one.*

Proof: For $\ell = s$ Equation (5.4.5) is obtained from Proposition 5.4.1.

So, from now on we assume that $\ell \in \{1, \ldots, s-1\}$.

Note that

$$\begin{aligned}
\breve{\alpha}(x)_{.a} + \sum_{i=\ell+1}^{s} a_{(\nu_i, j_{\nu_i}).a} &= \sum_{i=1}^{\ell} a_{(\nu_i, j_{\nu_i}).a} \\
\breve{\beta}(x)_{.b} + \sum_{i=\ell+1}^{s} b_{(\nu_i, j_{\nu_i}).b} &= \sum_{i=1}^{\ell} b_{(\nu_i, j_{\nu_i}).b}
\end{aligned}$$

First we show that there exists a $(2m+1)\times m$ binary matrix N_{ν_ℓ} such that

$$\left(\sum_{i=1}^{\ell} a_{(\nu_i,j_{\nu_i}).a} \parallel \breve{\alpha}(x)_{.b}+\breve{\gamma}(x)_{.b}+\mathcal{B}_\ell \parallel 1\right) N_{\nu_\ell}=\sum_{i=1}^{\ell} b_{(\nu_i,j_{\nu_i}).b} \tag{5.4.6}$$

Here we have

$$\begin{aligned}
&\breve{\alpha}(x)_{.b}+\breve{\gamma}(x)_{.b}+\mathcal{B}_\ell\\
=\;& \breve{\alpha}(x)_{.b}+\breve{\gamma}(x)_{.b}+\sum_{i=\ell+1}^{s}\left(a_{(\nu_i,j_{\nu_i}).b}+h_{(\nu_i,j_{\nu_i}).b}\right)+\\
&\sum_{i=\ell+1}^{s} a^{\theta}_{(\nu_i,j_{\nu_i}).a}\left(t_{(\nu_0).a}+t_{(\nu_{i-1}).a}\right)+\sum_{i=\ell+1}^{s} a_{(\nu_i,j_{\nu_i}).a}\left(t_{(\nu_i).a}+t_{(\nu_s).a}\right)^{\theta}\\
=\;& \sum_{i=1}^{s} b_{(\nu_i,j_{\nu_i}).b}+\sum_{i=1}^{s} a^{\sigma}_{(\nu_i,j_{\nu_i}).a}+t_{(\nu_0).b}+t^{\theta+1}_{(\nu_0).a}+t_{(\nu_s).b}+t_{(\nu_0).a}\;t^{\theta}_{(\nu_s).a}+\\
&t_{(\nu_0).a}\sum_{i=1}^{s} a^{\theta}_{(\nu_i,j_{\nu_i}).a}+t^{\theta}_{(\nu_s).a}\sum_{i=1}^{s} a_{(\nu_i,j_{\nu_i}).a}+\sum_{i=\ell+1}^{s}\left(a_{(\nu_i,j_{\nu_i}).b}+h_{(\nu_i,j_{\nu_i}).b}\right)+\\
&\sum_{i=\ell+1}^{s} a^{\theta}_{(\nu_i,j_{\nu_i}).a}\left(t_{(\nu_0).a}+t_{(\nu_{i-1}).a}\right)+\sum_{i=\ell+1}^{s} a_{(\nu_i,j_{\nu_i}).a}\left(t_{(\nu_i).a}+t_{(\nu_s).a}\right)^{\theta}\\
=\;& \sum_{i=1}^{s} b_{(\nu_i,j_{\nu_i}).b}+\sum_{i=1}^{s} a^{\sigma}_{(\nu_i,j_{\nu_i}).a}+t_{(\nu_0).b}+t^{\theta+1}_{(\nu_0).a}+t_{(\nu_s).b}+t_{(\nu_0).a}\;t^{\theta}_{(\nu_s).a}+\\
&t_{(\nu_0).a}\sum_{i=1}^{s} a^{\theta}_{(\nu_i,j_{\nu_i}).a}+t^{\theta}_{(\nu_s).a}\sum_{i=1}^{s} a_{(\nu_i,j_{\nu_i}).a}+\sum_{i=\ell+1}^{s} a_{(\nu_i,j_{\nu_i}).b}+\sum_{i=\ell+1}^{s} h_{(\nu_i,j_{\nu_i}).b}+\\
&t_{(\nu_0).a}\sum_{i=\ell+1}^{s} a^{\theta}_{(\nu_i,j_{\nu_i}).a}+\sum_{i=\ell+1}^{s} a^{\theta}_{(\nu_i,j_{\nu_i}).a}\;t_{(\nu_{i-1}).a}+\sum_{i=\ell+1}^{s} a_{(\nu_i,j_{\nu_i}).a}\;t^{\theta}_{(\nu_i).a}+\\
&t^{\theta}_{(\nu_s).a}\sum_{i=\ell+1}^{s} a_{(\nu_i,j_{\nu_i}).a}
\end{aligned}$$

$$\begin{aligned}
= \quad & \sum_{i=1}^{s} b_{(\nu_i,j_{\nu_i}).b} + \sum_{i=1}^{s} a^{\sigma}_{(\nu_i,j_{\nu_i}).a} + t_{(\nu_0).b} + t^{\theta+1}_{(\nu_0).a} + t_{(\nu_s).b} + t_{(\nu_0).a}\ t^{\theta}_{(\nu_s).a} + \\
& t_{(\nu_0).a} \sum_{i=1}^{s} a^{\theta}_{(\nu_i,j_{\nu_i}).a} + t^{\theta}_{(\nu_s).a} \sum_{i=1}^{s} a_{(\nu_i,j_{\nu_i}).a} + \sum_{i=\ell+1}^{s} a_{(\nu_i,j_{\nu_i}).b} + \sum_{i=\ell+1}^{s} \Big(b_{(\nu_i,j_{\nu_i}).b} + \\
& a^{\sigma}_{(\nu_i,j_{\nu_i}).a} + a_{(\nu_i,j_{\nu_i}).b} + t_{(\nu_{i-1}).a} a^{\theta}_{(\nu_i,j_{\nu_i}).a} + a_{(\nu_i,j_{\nu_i}).a}\ t^{\theta}_{(\nu_i).a} + \\
& t_{(\nu_{i-1}).a}\ t^{\theta}_{(\nu_i).a} + t_{(\nu_{i-1}).b} + t^{\theta+1}_{(\nu_{i-1}).a} + t_{(\nu_i).b} \Big) + t_{(\nu_0).a} \sum_{i=\ell+1}^{s} a^{\theta}_{(\nu_i,j_{\nu_i}).a} + \\
& \sum_{i=\ell+1}^{s} a^{\theta}_{(\nu_i,j_{\nu_i}).a}\ t_{(\nu_{i-1}).a} + \sum_{i=\ell+1}^{s} a_{(\nu_i,j_{\nu_i}).a}\ t^{\theta}_{(\nu_i).a} + t^{\theta}_{(\nu_s).a} \sum_{i=\ell+1}^{s} a_{(\nu_i,j_{\nu_i}).a} \\
= \quad & \sum_{i=1}^{s} b_{(\nu_i,j_{\nu_i}).b} + \sum_{i=1}^{s} a^{\sigma}_{(\nu_i,j_{\nu_i}).a} + t_{(\nu_0).b} + t^{\theta+1}_{(\nu_0).a} + t_{(\nu_s).b} + t_{(\nu_0).a}\ t^{\theta}_{(\nu_s).a} + \\
& t_{(\nu_0).a} \sum_{i=1}^{s} a^{\theta}_{(\nu_i,j_{\nu_i}).a} + t^{\theta}_{(\nu_s).a} \sum_{i=1}^{s} a_{(\nu_i,j_{\nu_i}).a} + \sum_{i=\ell+1}^{s} b_{(\nu_i,j_{\nu_i}).b} + \sum_{i=\ell+1}^{s} a^{\sigma}_{(\nu_i,j_{\nu_i}).a} + \\
& \sum_{i=\ell+1}^{s} a^{\theta}_{(\nu_i,j_{\nu_i}).a}\ t_{(\nu_{i-1}).a} + \sum_{i=\ell+1}^{s} a_{(\nu_i,j_{\nu_i}).a}\ t^{\theta}_{(\nu_i).a} + \sum_{i=\ell+1}^{s} \Big(t_{(\nu_{i-1}).a}\ t^{\theta}_{(\nu_i).a} + \\
& t_{(\nu_{i-1}).b} + t^{\theta+1}_{(\nu_{i-1}).a} + t_{(\nu_i).b} \Big) + t_{(\nu_0).a} \sum_{i=\ell+1}^{s} a^{\theta}_{(\nu_i,j_{\nu_i}).a} + \sum_{i=\ell+1}^{s} a^{\theta}_{(\nu_i,j_{\nu_i}).a}\ t_{(\nu_{i-1}).a} + \\
& \sum_{i=\ell+1}^{s} a_{(\nu_i,j_{\nu_i}).a}\ t^{\theta}_{(\nu_i).a} + t^{\theta}_{(\nu_s).a} \sum_{i=\ell+1}^{s} a_{(\nu_i,j_{\nu_i}).a} \\
= \quad & \sum_{i=1}^{\ell} b_{(\nu_i,j_{\nu_i}).b} + \sum_{i=1}^{\ell} a^{\sigma}_{(\nu_i,j_{\nu_i}).a} + t_{(\nu_0).a} \sum_{i=1}^{\ell} a^{\theta}_{(\nu_i,j_{\nu_i}).a} + t^{\theta}_{(\nu_s).a} \sum_{i=1}^{\ell} a_{(\nu_i,j_{\nu_i}).a} + C_{\ell}
\end{aligned}$$

where the term

$$\begin{aligned}
C_{\ell} = \quad & \sum_{i=\ell+1}^{s} \big(t_{(\nu_{i-1}).a}\ t^{\theta}_{(\nu_i).a} + t_{(\nu_{i-1}).b} + t^{\theta+1}_{(\nu_{i-1}).a} + t_{(\nu_i).b} \big) + \\
& t_{(\nu_0).b} + t^{\theta+1}_{(\nu_0).a} + t_{(\nu_s).b} + t_{(\nu_0).a}\ t^{\theta}_{(\nu_s).a}
\end{aligned}$$

is viewed as a constant in $\mathbb{F}_{2^m}$. Therefore, Equation (5.4.6) becomes

$$\left(\sum_{i=1}^{\ell} a_{(\nu_i, j_{\nu_i}).a} \;\middle\|\; \begin{array}{l} \sum_{i=1}^{\ell} b_{(\nu_i, j_{\nu_i}).b} + \sum_{i=1}^{\ell} a^{\sigma}_{(\nu_i, j_{\nu_i}).a} + \\ t_{(\nu_0).a} \sum_{i=1}^{\ell} a^{\theta}_{(\nu_i, j_{\nu_i}).a} + \\ t^{\theta}_{(\nu_s).a} \sum_{i=1}^{\ell} a_{(\nu_i, j_{\nu_i}).a} + C_\ell \end{array} \;\middle\|\; 1 \right) N_{\nu_\ell} = \sum_{i=1}^{\ell} b_{(\nu_i, j_{\nu_i}).b} \qquad (5.4.7)$$

Because the elements $t_{(\nu_0).a}$ and $t_{(\nu_s).a}$ are constants and θ is a linear mapping, there exist $m \times m$ matrices T_{ν_0}, T_{ν_ℓ} such that

$$t_{(\nu_0).a} \sum_{i=1}^{\ell} a^{\theta}_{(\nu_i, j_{\nu_i}).a} = \sum_{i=1}^{\ell} a_{(\nu_i, j_{\nu_i}).a} \, T_{\nu_0}$$

$$t^{\theta}_{(\nu_s).a} \sum_{i=1}^{\ell} a_{(\nu_i, j_{\nu_i}).a} = \sum_{i=1}^{\ell} a_{(\nu_i, j_{\nu_i}).a} \, T_{\nu_\ell}$$

Now set

$$N_{\nu_\ell} = \begin{pmatrix} T_{\nu_0} + T_{\nu_\ell} + \sigma \\ I_m \\ C_\ell \end{pmatrix}$$

where I_m is the $m \times m$ identity matrix. Then it is easy to check that the $(2m+1) \times m$ matrix N_{ν_ℓ} satisfies Equation (5.4.7). Similar to the proof of Proposition 5.4.1, by using

$$\left(\sum_{i=1}^{\ell} b_{(\nu_i, j_{\nu_i}).b} \right) \rho^{-1} = \sum_{i=1}^{\ell} e'_{(\nu_i, j_{\nu_i}).b} \;:=\; \breve{\varepsilon}'_{\ell,\xi}(x)_{.b}$$

and

$$\left(\, \breve{\varepsilon}'_{\ell,\xi}(x)_{.b} \, \right) P_{\nu_\ell} = \pi_{\nu_\ell}(j_{\nu_\ell})$$

we define

$$L_{\nu_\ell} := N_{\nu_\ell} \cdot \rho^{-1} \cdot P_{\nu_\ell}$$

Then L_{ν_ℓ} is the binary matrix that satisfies Equation (5.4.5). □

We are now in a position to describe an algorithm for recovering permutations $\pi_1, \ldots, \pi_s$ by using Proposition 5.4.2. The algorithm delivers the permutation ξ as well.

Algorithm 17 Matrix-permutation Attack on NFT-MST$_3$: Permutation recovery

```
Input: Public key $[\alpha, \gamma]$.
Output: Permutations $[\pi_1, \dots, \pi_s, \xi]$.
for $\ell \leftarrow s$ downto 1 do
  A: Choose random plaintexts $x^{(i)} \mapsto (j_1^{(i)}, \dots, j_s^{(i)})$, and construct vectors $y^{(i)} := \Big( \breve{\alpha}_{\ell,id}(x)^{(i)}_{.a} \parallel \breve{\alpha}_{\ell,id}(x)^{(i)}_{.b} + \breve{\gamma}_{\ell,id}(x)^{(i)}_{.b} \parallel 1 \Big)$, as in Proposition 5.4.2. Define $n_\ell$ to be the maximum number of linearly independent vectors $y^{(i)}$, where $n_\ell = n'_\ell + 1 + \sum_{m=1}^{\ell} k_m$. Here $n'_\ell$ is the maximum number of linearly independent columns of the matrix formed by vectors $\Big( \breve{\alpha}_{\ell,id}(x)^{(i)}_{.a} \Big)$.
  B: Set $v \leftarrow 1$.
  C: for $p \leftarrow 1$ to $k_v$ do
    C.1: Select a set $J_v$ of $k_v$ randomly chosen vectors in $\mathbb{F}_{2^{k_v}}$.
    C.2: Choose a random binary vector $w = (w_1, \dots, w_{k_v}) \in \mathbb{F}_{2^{k_v}}$, and set
         $\pi_{v,p}(j_i) = w_i$ for each $j_i \in J_v$.
    C.3: Choose random plaintexts $x^{(i)} \mapsto (j_1^{(i)}, \dots, \mathbf{j}_v^{(i)}, \dots, j_s^{(i)})$, where $\mathbf{j}_v^{(i)} \in J_v$ and construct vectors $y^{(i)} := \Big( \breve{\alpha}_{\ell,id}(x)^{(i)}_{.a} \parallel \breve{\alpha}_{\ell,id}(x)^{(i)}_{.b} + \breve{\gamma}_{\ell,id}(x)^{(i)}_{.b} \parallel 1 \Big)$, as in Proposition 5.4.2.

         Repeat this step for an appropriate number of choices of $x^{(i)}$ and form a matrix $Y_v$ with rows being the linearly independent vectors $y^{(i)}$. If $\mathrm{rank}(Y_v) < n_\ell$ then return to C.1.
    C.4: Let $x^{(i)} \mapsto (j_1^{(i)}, \dots, j_v^{(i)}, \dots, j_s^{(i)})$, for $(i) = 1, \dots, n_\ell$, be the plaintext used to construct row $(i)$ of the $n_\ell \times (2m+1)$ binary matrix $Y_v$ in the previous step. Form the $n_\ell \times 1$ matrix $Z_{v,p}$ with value $\pi_{v,p}(j_v^{(i)})$ as entry in row $(i)$.
    C.5: Construct a $(2m+1) \times n_\ell$ binary encoding matrix $E_v$, such that
         $\mathrm{rank}(Y_v.E_v) = n_\ell$
    C.6: Compute matrix $M_{v,p} = E_v.(Y_v.E_v)^{-1}.Z_{v,p}$
    C.7: For each $j_v \in \mathbb{F}_{2^{k_v}} \setminus J_v$ choose a random plaintext $x \mapsto (j_1, \dots, j_v, \dots, j_s)$ and compute the value for $\pi_{v,p}(j_v)$ by

         $\pi_{v,p}(j_v) := \Big( \breve{\alpha}_{\ell,id}(x)_{.a} \parallel \breve{\alpha}_{\ell,id}(x)_{.b} + \breve{\gamma}_{\ell,id}(x)_{.b} \parallel 1 \Big).M_{v,p}$
    C.8: Choose a random plaintext $x \mapsto (j_1, \dots, j_v, \dots, j_s)$ and compute the value

         $y = \Big( \breve{\alpha}_{\ell,id}(x)_{.a} \parallel \breve{\alpha}_{\ell,id}(x)_{.b} + \breve{\gamma}_{\ell,id}(x)_{.b} \parallel 1 \Big).M_{v,p}$
```

```
            If $y \neq \pi_{\nu,p}(j_\nu)$ then return to C.2 and try another choice for $w \in \mathbb{F}_{2^{k_\nu}}$ (this
            can be done in at most $2^{k_\nu}$ times). If no choice for $w$ in C.2 is possible, then
            set $\nu \leftarrow (\nu+1)$ and return to C.

            If $y = \pi_{\nu,p}(j_\nu)$, repeat C.8 for an appropriate number of times.
        end for
    D: Set transposition $\xi_\ell := (\nu, \ell)$. Permute the blocks of $\alpha$ and $\gamma$ with transposition $\xi_\ell$
       to get $\alpha'$ and $\gamma'$. Set $\alpha \leftarrow \alpha'$ and $\gamma \leftarrow \gamma'$.
    E: For each $j_\nu \in \mathbb{F}_{2^{k_\nu}}$, by using $\pi_{\nu,p}(j_\nu)$ for $p = 1, \dots, k_\nu$, one obtains $\pi_\nu(j_\nu)$, and thus
       determines permutation $\pi_\nu$.
end for
Return $[\pi_1, \dots, \pi_s, \xi]$, where $\xi = \xi_s \circ \dots \circ \xi_1$.
```

We add some comments to clarify the steps of the Algorithm 17

A: To determine the maximum value for the parameter n_ℓ we have to run this step for a sufficient number of random inputs $x^{(i)}$.

B: This step initializes the parameter ν to start the subsequent steps of the algorithm to determine $\nu = \xi(\ell)$.

C: The inner loop is used to determine each bit $\pi_{\nu,p}(j_\nu)$ of $\pi_\nu(j_\nu)$ for $p = 1, \dots, k_\nu$, separately, for which $\pi_\nu(j_\nu) := \big(\pi_{\nu,1}(j_\nu) \| \dots \| \pi_{\nu,k_\nu}(j_\nu)\big)$ for all $j_\nu \in \mathbb{F}_{2^{k_\nu}}$.

C.1: The choice of the parameter k_ν, i.e. size of the set J_ν, has an effect on the behaviour of the algorithm. If $|J_\nu| < k_\nu$, step C.3 cannot be finished (i.e. we always get $\mathsf{rank}(Y_\nu) < n_\ell$). If $|J_\nu| > k_\nu$, the workload required in step C.2 will be increased comparing with the case $|J_\nu| = k_\nu$.

C.2: In this step, we guess the p-th bit $\pi_{\nu,p}(j_\nu)$ of $\pi_\nu(j_\nu)$ for all $j_\nu \in J_\nu$.

C.3: In this step, a plaintext $x^{(i)} \mapsto (j_1^{(i)}, \dots, \mathbf{j}_\mathbf{v}^{(i)}, \dots, j_s^{(i)})$ is chosen in such a way that the component $\mathbf{j}_\mathbf{v}^{(i)}$ belongs to J_ν (chosen in step C.1). The other components j_u with $u \neq \nu$ are arbitrarily chosen. We repeat this step until we get a matrix Y_ν with $\mathsf{rank}(Y_\nu) = n_\ell$.

If the elements of J_ν, $|J_\nu| = k_\nu$, are chosen in such a way that the set $\{\pi_\nu(j_\nu) \mid j_\nu \in J_\nu\}$ has less than k_ν linearly independent vectors (of size k_ν), the $\mathsf{rank}(Y_\nu)$ will be smaller than n_ℓ. In this case, the algorithm returns to step C.1, and generates a new set J_ν.

(The other possibility could be to extend the size of set J_ν, i.e. $|J_\nu| > k_\nu$.)

C.4: We construct the $n_\ell \times 1$ matrix $Z_{\nu,p}$ with values $\pi_{\nu,p}(j_\nu^{(i)})$ using C.2 and C.3.

C.5: In this step, we construct a binary $(2m+1) \times n_\ell$ matrix E_ν such that $\mathsf{rank}(Y_\nu.E_\nu) = n_\ell$. This is done in the following way: Let $Q = \{1, \ldots, 2m+1\}$ be the index set of columns of Y_ν. Find a subset $Q_\nu \subseteq Q$ with $|Q_\nu| = n_\ell$, such that the columns with indices in Q_ν are all linearly independent. Consider the identity $(2m+1) \times (2m+1)$ matrix $I_{(2m+1)}$. Remove all columns with indices in $Q \setminus Q_\nu$ from $I_{(2m+1)}$ to form a $(2m+1) \times n_\ell$ matrix E_ν.

C.6: Using E_ν from step C.5 we determine the p-th column $M_{\nu,p}$ of the matrix M_ν.

C.7: This step computes the p-th bit $\pi_{\nu,p}(j_\nu)$ of $\pi_\nu(j_\nu)$ for all remaining $j_\nu \in \mathbb{F}_{2^{k_\nu}}$.

C.8: This step verifies whether the bit $\pi_{\nu,p}(j_\nu)$ guessed in step C.2 or computed in step C.7 for all $j_\nu \in \mathbb{F}_{2^{k_\nu}}$ is correct and whether the value ν satisfies $\nu = \xi(\ell)$. Running this step an appropriately sufficient number of times allows us to check these requirements.

D: In this step, we use $\nu = \xi(\ell)$ determined in the previous loop to construct a transposition ξ_ℓ. We update α and γ, permuting their blocks with ξ_ℓ and continue the main loop with the new value $\ell \leftarrow (\ell - 1)$.

E: From the p-th bit $\pi_{\nu,p}(j_\nu)$ for all $p = 1, \ldots, k_\nu$ we construct $\pi_\nu(j_\nu)$. By collecting all $\pi_\nu(j_\nu)$, $j_\nu \in \mathbb{F}_{2^{k_\nu}}$, we are able to recover the permutation π_ν.

Proposition 5.4.4 *Let α, γ be the covers of type $(r_1, \ldots, r_s)$ used as the public key in NFT-MST_3. Let $k_\ell := \lceil \log_2 r_\ell \rceil$. The workload required to recover permutations $[\pi_1, \ldots, \pi_s, \xi]$ using Algorithm 17 is bounded by $\mathcal{O}(\sum_{\ell=1}^{s} \ell \, k_\ell \, 2^{k_\ell - 1})$.*

Proof: In step C.2 of Algorithm 17 we have to guess vector w of k_ν bits to set the p-th bit $\pi_{\nu,p}(j_\nu)$ of $\pi_\nu(j_\nu)$ for all $j_\nu \in J_\nu$. The complexity of the algorithm includes the times required to run through all bits $p \in \{1, \ldots, k_\nu\}$ with an average of $\ell/2$ times until step C.8 successfully terminates by finding $\nu := \xi(\ell)$, and also those for the steps in the main loop for $\ell \in \{1, \ldots, s\}$. Summing up these together yields the workload as shown in the bound stated. $\square$

Note that for any $j_m \in \{1, \ldots, r_m\}$

$$\begin{aligned}
\big(t_{(0).a} + t_{(\ell-1).a}\big) &= \sum_{m=1}^{\ell-1}\big(a_{(m,j_m).a} + h_{(m,j_m).a}\big) = \sum_{m=1}^{\ell-1}\big(a_{(m,1).a} + h_{(m,1).a}\big) \\
\big(t_{(\ell).a} + t_{(s).a}\big)^\theta &= \sum_{m=\ell+1}^{s}\big(a_{(m,j_m).a} + h_{(m,j_m).a}\big)^\theta = \sum_{m=\ell+1}^{s}\big(a_{(m,1).a} + h_{(m,1).a}\big)^\theta
\end{aligned}$$

We use Proposition 5.4.3 to design the following algorithm.

Algorithm 18 Matrix-permutation Attack on NFT-MST$_3$: Matrix recovery

```
Input: Public key $[\alpha, \gamma]$, permutations $[\pi_1, \dots, \pi_s, \xi]$.
Output: Matrices $[L_1, \dots, L_s]$.
  A: Set $\mathcal{A}_s \leftarrow (0, \dots, 0)$, an $m$-bit zero vector.
  B: for $\ell \leftarrow s$ downto 1 do
       1: Set $\nu \leftarrow \xi(\ell)$.
       2: Select random plaintexts $x^{(i)} \mapsto (j_1^{(i)}, \dots, j_s^{(i)})$, and construct vectors
          $y^{(i)} := \left( \breve{\alpha}_{\ell,\xi}(x)^{(i)}_{.a} \parallel \breve{\alpha}(x)^{(i)}_{.b} + \breve{\gamma}(x)^{(i)}_{.b} + \mathcal{A}^{(i)}_{\ell} \parallel 1 \right)$, as in Proposition 5.4.3. Define $n_\nu$ to be the maximum number of linearly independent vectors $y^{(i)}$, where $n_\nu = n'_\nu + 1 + \sum_{m=1}^{\ell} k_{\xi(m)}$. Here $n'_\nu$ is the maximum number of linearly independent columns of the matrix formed by vectors $\left( \breve{\alpha}_{\ell,\xi}(x)^{(i)}_{.a} \right)$. Repeat this step for an appropriate number of choices of $x^{(i)}$ and form a matrix $Y_\nu$ with $n_\nu$ rows being the linearly independent vectors $y^{(i)}$.
       3: Let $x^{(i)} \mapsto (j_1^{(i)}, \dots, j_s^{(i)})$, for $(i) = 1, \dots, n_\nu$, be the plaintext used to construct row $(i)$ of the $n_\nu \times (2m+1)$ binary matrix $Y_\nu$ in the previous step. Form the $n_\nu \times k_\nu$ matrix $Z_\nu$ with value $\pi_\nu(j_\nu)$ as entry in row $(i)$.
       4: Construct a $(2m+1) \times n_\nu$ binary encoding matrix $E_\nu$, such that
          $\mathrm{rank}(Y_\nu.E_\nu) = n_\nu$
       5: Compute matrix $L_\nu = E_\nu \,.\, (Y_\nu.E_\nu)^{-1}.\, Z_\nu$
          If $\ell = 1$ then return $[L_1, \dots, L_s]$.
       6: Set $\mathcal{A}_{\ell-1} \leftarrow \mathcal{A}_\ell + a_{(\nu,j_\nu).b} + h_{(\nu,j_\nu).b} + a^{\theta}_{(\nu,j_\nu).a} \sum_{m=1}^{\ell-1} \left( a_{(\xi(m),1).a} + h_{(\xi(m),1).a} \right) + a_{(\nu,j_\nu).a} \sum_{m=\ell+1}^{s} \left( a_{(\xi(m),1).a} + h_{(\xi(m),1).a} \right)^{\theta}$
     end for
```

By making use of the information computed by Algorithms 17 and 18 we now present an algorithm for the decryption of a given ciphertext $y = (y_1, y_2)$.

Algorithm 19 Matrix-permutation Attack on NFT-MST_3: Factorization

```
Input: $[\pi_1, \dots, \pi_s, \xi, L_1, \dots, L_s]$ for the public key $[\alpha, \gamma]$, ciphertext $y = (y_1, y_2)$.
Output: Plaintext $x \mapsto (j_1, \dots, j_s)$, such that $y_1 = \breve{\alpha}(x)$, $y_2 = \breve{\gamma}(x)$.
  A: Set $\mathcal{A}_s \leftarrow (0, \dots, 0)$.
  B: for $\ell \leftarrow s$ downto 1 do
       1: Set $\nu \leftarrow \xi(\ell)$.
       2: Construct a vector
          $w = \Big( y_{1.a} \parallel y_{1.b} + y_{2.b} + \mathcal{A}_\ell \parallel 1 \Big)$
       3: Compute $\pi_\nu(j_\nu) = w \cdot L_\nu$
       4: Recover $j_\nu$ using $\pi_\nu(j_\nu)$ and permutation $\pi_\nu$.
          If $\ell = 1$ then return $(j_1, \dots, j_s)$.
       5: Set $y_{1.a} \leftarrow y_{1.a} + a_{(\nu,j_\nu).a}$
          $\mathcal{A}_{\ell-1} \leftarrow \mathcal{A}_\ell + a_{(\nu,j_\nu).b} + h_{(\nu,j_\nu).b} + a^\theta_{(\nu,j_\nu).a} \sum_{m=1}^{\ell-1} \big(a_{(\xi(m),1).a} + h_{(\xi(m),1).a}\big) +$
               $a_{(\nu,j_\nu).a} \sum_{m=\ell+1}^{s} \big(a_{(\xi(m),1).a} + h_{(\xi(m),1).a}\big)^\theta$
     end for
```

As presented above, the Matrix-permutation attack on NFT-MST_3 makes use of Algorithm 17 to recover permutations $[\pi_1, \dots, \pi_s, \xi]$ and then Algorithm 18 to construct matrices $[L_1, \dots, L_s]$. The knowledge of $[L_1, \dots, L_s]$ and $[\pi_1, \dots, \pi_s, \xi]$ allows the adversary to decrypt any ciphertext by using Algorithm 19. The usage of non-fused transversal signatures permits the construction of such matrix L_i for any block $i = \{1, \dots, s\}$ and to compute the image $\pi_i(j_i)$ of j_i under permutation π_i as shown in Proposition 5.4.3. This fact is used in step 3 of Algorithm 19. As π_i is a bijection, the preimage j_i can be recovered if $\pi_i(j_i)$ is known as shown in step 4 of the same algorithm.

Remark 5.4.1 The determination of permutations $[\pi_1, \dots, \pi_s, \xi]$ and the construction of matrices $[L_1, \dots, L_s]$ could be designed in a single algorithm. However, such an algorithm would become very involved. Therefore, for the sake of clarity regarding the description of the Matrix-permutation attack we have presented two separated algorithms, namely Algorithm 17 and Algorithm 18.

As the workload required for Algorithm 18 is negligible, the complexity of the Matrix-permutation attack is reduced to the complexity of the determination of permutations $[\pi_1, \dots, \pi_s, \xi]$ by Algorithm 17. Thus we have the following proposition.

Proposition 5.4.5 *By using the same notation as in Proposition 5.4.4, the workload required to recover the cleartext for a given ciphertext by using the Matrix-permutation attack on NFT-*MST_3 *scheme is roughly of the same amount as required to recover permutations* $[\pi_1, \ldots, \pi_s, \xi]$, *and is bounded by* $\mathcal{O}(\sum_{\ell=1}^{s} \ell\, k_\ell\, 2^{k_\ell - 1})$.

The complexity as given in Proposition 5.4.5 shows in particular that for relatively small values k_i, which are usually used in a real version of the MST_3 scheme, say $k_i \leqslant 15$, the non-fused transversal logarithmic signatures cannot be used for a secure realization of MST_3.

5.4.4 The Matrix-permutation attack on FT-MST_3

In this section, we attempt to determine the complexity of the Matrix-permutation attack on FT-MST_3.

As shown in the previous section, the Matrix-permutation attack exploits fully the way of factorizing with respect to a non-fused transversal logarithmic signature β (Algorithm 12), even though the adversary does not know β. Thus, the knowledge provided by a factorization with respect to β by Algorithm 12 will be the crucial information to the estimation of the complexity of recovering the cleartext when the Matrix-permutation attack is applied.

To simplify the description of the Matrix-permutation attack on FT-MST_3 we confine ourselves to using only step 1 and 3 of Algorithm 9 to create logarithmic signature β.

Let $\{K_1, K_2, \ldots, K_v\}$ be a partition on the set $\{1, \ldots, m\}$ with $|K_i| = k_i$ and $t_i = 2^{k_i}$ as described in Algorithm 9. Let $\varepsilon^* := (e^*_{i,j})$ be a signature of type $(t_1, \ldots, t_v)$ created after step 1 of the Algorithm 9.

W.l.o.g., we consider $\beta := (b_{i,j})$ to be a logarithmic signature created by fusion of blocks $(\ell, \ell+2)$ and $(\ell+1, \ell+3)$ of ε^*. (Note that no consecutive blocks are fused.) Then $\beta = [B_1, \ldots, B_s]$ is of type $(r_1, \ldots, r_s)$, where $r_\ell = t_\ell \cdot t_{\ell+2}$ and $r_{\ell+1} = t_{\ell+1} \cdot t_{\ell+3}$. We now consider one fused block, say $B_{\ell+1}$, of β.

Let $u^{[n]}_{i,j_i}$ (resp. $e^{[n]}_{i,j_i}$) be a vector of length k_n, consisting of the bits of b_{i,j_i} on the positions corresponding to K_n.

Let $x \mapsto (j_1, \ldots, j_v)$, also let $x' \mapsto (j'_1, \ldots, j'_s)$, where $j'_\ell = j_\ell \| j_{\ell+1}$ and $j'_{\ell+1} = j_{\ell+2} \| j_{\ell+3}$.

Then

$$\begin{array}{lcccccccccc}
e^*_{1,\,j_1} & = (\dots \| & \bar{0} & \| & \bar{0} & \| & \bar{0} & \| & \bar{0} & \| \dots) \\
\vdots & & & & & & & & & \\
e^*_{\ell-1,\,j_{\ell-1}} & = (\dots \| & \bar{0} & \| & \bar{0} & \| & \bar{0} & \| & \bar{0} & \| \dots) \\
e^*_{\ell,\,j_\ell} & = (\dots \| & \mathfrak{e}^{[\ell]}_{\ell,\,j_\ell} & \| & \bar{0} & \| & \bar{0} & \| & \bar{0} & \| \dots) \\
e^*_{\ell+1,\,j_{\ell+2}} & = (\dots \| & u^{[\ell]}_{\ell+1,\,j_{\ell+2}} & \| & \mathfrak{e}^{[\ell+1]}_{\ell+1,\,j_{\ell+2}} & \| & \bar{0} & \| & \bar{0} & \| \dots) \\
e^*_{\ell+2,\,j_{\ell+1}} & = (\dots \| & u^{[\ell]}_{\ell+2,\,j_{\ell+1}} & \| & u^{[\ell+1]}_{\ell+2,\,j_{\ell+1}} & \| & \mathfrak{e}^{[\ell+2]}_{\ell+2,\,j_{\ell+1}} & \| & \bar{0} & \| \dots) \\
e^*_{\ell+3,\,j_{\ell+3}} & = (\dots \| & \underbrace{u^{[\ell]}_{\ell+3,\,j_{\ell+3}}}_{K_\ell} & \| & \underbrace{u^{[\ell+1]}_{\ell+3,\,j_{\ell+3}}}_{K_{\ell+1}} & \| & \underbrace{u^{[\ell+2]}_{\ell+3,\,j_{\ell+3}}}_{K_{\ell+2}} & \| & \underbrace{\mathfrak{e}^{[\ell+3]}_{\ell+3,\,j_{\ell+3}}}_{K_{\ell+3}} & \| \dots)
\end{array}$$

Then

$$\breve{\beta}(x') = b_{1,j'_1} \oplus \dots \oplus b_{\ell,j'_\ell} \oplus b_{\ell+1,j'_{\ell+1}} \oplus \dots \oplus b_{s,j'_s}$$

where

$$\begin{array}{lcccccccccc}
b_{\ell,j'_\ell} & = (\dots \| & \begin{array}{l}\mathfrak{e}^{[\ell]}_{\ell,\,j_\ell} \oplus \\ u^{[\ell]}_{\ell+2,\,j_{\ell+1}}\end{array} & \| & u^{[\ell+1]}_{\ell+2,\,j_{\ell+1}} & \| & \mathfrak{e}^{[\ell+2]}_{\ell+2,\,j_{\ell+1}} & \| & \bar{0} & \| \dots) \\
b_{\ell+1,\,j'_{\ell+1}} & = (\dots \| & \begin{array}{l}u^{[\ell]}_{\ell+1,\,j_{\ell+2}} \oplus \\ u^{[\ell]}_{\ell+3,\,j_{\ell+3}}\end{array} & \| & \begin{array}{l}\mathfrak{e}^{[\ell+1]}_{\ell+1,\,j_{\ell+2}} \oplus \\ u^{[\ell+1]}_{\ell+3,\,j_{\ell+3}}\end{array} & \| & u^{[\ell+2]}_{\ell+3,\,j_{\ell+3}} & \| & \mathfrak{e}^{[\ell+3]}_{\ell+3,\,j_{\ell+3}} & \| \dots)
\end{array}$$

and therefore

$$\sum_{i=1}^{\ell+1} b_{i,j'_i} = (\dots \| \overbrace{\begin{array}{l}\mathfrak{e}^{[\ell]}_{\ell,\,j_\ell} \oplus \\ u^{[\ell]}_{\ell+2,\,j_{\ell+1}} \oplus \\ u^{[\ell]}_{\ell+1,\,j_{\ell+2}} \oplus \\ u^{[\ell]}_{\ell+3,\,j_{\ell+3}}\end{array}}^{K_\ell} \| \overbrace{\begin{array}{l}u^{[\ell+1]}_{\ell+2,\,j_{\ell+1}} \oplus \\ \mathfrak{e}^{[\ell+1]}_{\ell+1,\,j_{\ell+2}} \oplus \\ u^{[\ell+1]}_{\ell+3,\,j_{\ell+3}}\end{array}}^{K_{\ell+1}} \| \overbrace{\begin{array}{l}\mathfrak{e}^{[\ell+2]}_{\ell+2,\,j_{\ell+1}} \oplus \\ u^{[\ell+2]}_{\ell+3,\,j_{\ell+3}}\end{array}}^{K_{\ell+2}} \| \overbrace{\mathfrak{e}^{[\ell+3]}_{\ell+3,\,j_{\ell+3}}}^{K_{\ell+3}} \| \dots)$$

Assume we use the factorization scheme as given by Algorithm 12. As the bits of $u^{[m]}_{i,j_i}$, are randomly chosen, only the bits of $\mathfrak{e}^{[m]}_{i,j_i}$ can be used for factoring with respect to β. Therefore, to factorize $\sum_{i=1}^{\ell+1} b_{i,j'_i}$, i.e. to recover the index $j'_{\ell+1}$ for block $B_{\ell+1}$, we may only use bits of $\mathfrak{e}^{[\ell+3]}_{\ell+3,\,j_{\ell+3}}$, i.e. the bits on positions $K_{\ell+3}$. However, as $B_{\ell+1}$ has the length $r_{\ell+1} = 2^{k_{\ell+1}+k_{\ell+3}}$, there are $2^{k_{\ell+1}}$ elements of $B_{\ell+1}$ having the same value $\mathfrak{e}^{[\ell+3]}_{\ell+3,\,j_{\ell+3}}$ on positions $K_{\ell+3}$. In other words, only $k_{\ell+3}$ bits from index $j'_{\ell+1}$ can be determined.

Having obtained this information, we now return to the Matrix-permutation attack when a fused signature β is used in an FT-MST_3. Similar to Proposition 5.4.3 we may show that there exists a matrix $L_{\ell+1}$ such that

$$\left(\ \breve{\alpha}(x)_{.a} + \mathcal{A}_{\ell+1} \parallel \breve{\alpha}(x)_{.b} + \breve{\gamma}(x)_{.b} + \mathcal{B}_{\ell+1} \parallel 1\ \right) L_{\ell+1} = \mathfrak{e}^{[\ell+3]}_{\ell+3,\, j_{\ell+3}}$$

Using such a matrix we can recover only $k_{\ell+3}$ from $(k_{\ell+1} + k_{\ell+3})$ bits of $j'_{\ell+1}$ for $B_{\ell+1}$.

This shows that the Matrix-permutation attack applied to FT-MST_3 can recover only a portion of bits of the index in each fused block of β. Thus we have the following proposition.

Proposition 5.4.6 *Let B_ℓ be a block of a fused transversal logarithmic signature β used in FT-MST_3. Let $B_\ell = ((D_{i_1}.D_{i_2})\dots D_{i_{u_\ell}})$ as defined in Algorithm 9, where $i_1 < i_2 < \dots < i_{u_\ell}$. Let $k_i = \lceil \log_2 D_i \rceil$. By using the Matrix-permutation attack on FT-MST_3 one can determine $k_{i_{u_\ell}}$ from $\sum_{j=1}^{u_\ell} k_{i_j}$ bits for the index in block B_ℓ.*

The complexity of factoring a ciphertext by using the Matrix-permutation attack on FT-MST_3 is thus given as the product of the complexities for factoring with respect to each block B_ℓ, $\ell = 1, \dots, s$. Moreover, as the factorization has to be proceeded implicitly according to the permutation ξ of Algorithm 9, it turns out that the last attacked block can be carried out by a table search and therefore has a negligible complexity. To summarize, we record the complexity of the Matrix-permutation attack on FT-MST_3 in the following proposition.

Proposition 5.4.7 *Let m be an input length of an FT-MST_3 scheme with a fused transversal logarithmic signature β created by Algorithm 9. Let $\mathcal{P} = \{P_1, \dots, P_s\}$ be a partition used in step 3 of this algorithm where $P_\ell = \{i_{\ell,1}, \dots, i_{\ell,u_\ell}\}$ for $\ell = 1, \dots, s$. Let $k_\ell = \lceil \log_2 D_\ell \rceil$, where D_ℓ is defined by the same algorithm. Then the workload still needed after the Matrix-permutation attack to recover the plaintext for a given ciphertext is of $\mathcal{O}(2^c)$ where*

$$c = (m - \sum_{\ell=2}^{s} k_{i_{\ell,u_\ell}} - \sum_{j=1}^{u_1} k_{i_{1,j}})$$

5.4.5 Attack by repeated fusion

We can envisage a further method of using the Matrix-permutation attack on the FT-MST_3 scheme. Suppose that the adversary attempts to keep fusing the blocks of α and γ to eventually obtain a new $\tilde{\alpha}$ and $\tilde{\gamma}$, in which the corresponding logarithmic signature $\tilde{\beta}$ (inside $\tilde{\gamma}$) has a block $\tilde{B}_i$ which forms a subspace of dimension m_i in $\mathcal{Z}$. Note that the adversary actually does not know $\tilde{B}_i$ and therefore cannot verify whether $\tilde{B}_i$ is a subspace or not. Assuming that $\tilde{B}_i$ is a subspace (s)he may attempt to apply the Matrix-permutation attack to $\tilde{\alpha}$ and $\tilde{\gamma}$ to compute the index in $\tilde{B}_i$ for the plaintext from a given ciphertext. It is fairly easy to prevent this type of attack by selecting a partition $\mathcal{P}$ in step 3 of Algorithm 9 in a way that such a block $\tilde{B}_i$ necessarily has a large dimension m_i. This makes the Matrix-permutation attack impossible because of its complexity, as given in Proposition 5.4.7.

An Example

Let $m = 160$ and $\varepsilon = [E_1, \ldots, E_{40}]$ be a randomized canonical logarithmic signature of type $(16, 16, \ldots, 16)$. Divide blocks of ε in two halfs, i.e. $\mathcal{I}_1 = \{E_1, \ldots, E_{20}\}$ and $\mathcal{I}_2 = \{E_{21}, \ldots, E_{40}\}$. Now construct $\beta = [B_1, \ldots, B_{20}]$, where each block B_i is created by fusing a block $E_i \in \mathcal{I}_1$ with a block $E_j \in \mathcal{I}_2$. The complexity of recovering the unknown plaintext, if Matrix-permutation attack was used on new MST_3 with such fused β, is of $\mathcal{O}(2^{114})$ (see row 3 in Table 6.2).

If we want to fuse 3 blocks of ε, we may divide blocks of ε in three parts $\mathcal{I}_1, \mathcal{I}_2, \mathcal{I}_3$ and then fuse blocks E_i, E_j, E_k where $E_i \in \mathcal{I}_1$, $E_j \in \mathcal{I}_2$, and $E_k \in \mathcal{I}_3$, etc..

Clearly, to form a subspace by repeated fusion, in both cases, an adversary has to fuse all blocks into one. This is not possible as such a block would have the size equal to $|\mathcal{Z}|$.

5.5 Conclusions

In this chapter, we present an approach to re-designing MST_3 for use with the Suzuki 2-groups. This revised scheme uses a secret homomorphism f to enhance the security of the scheme. The purpose of using f is to mask elements of the secret logarithmic signature β with the elements of random cover α transformed by f. To extract β one must first determine f. A set-up with randomized encryption of the scheme without causing additional ciphertext expansion has been proposed.

We introduce transformations T1,. . . ,T4 for generating factorizable signatures of $\mathcal{Z}$. In particular, the operation T2 (fusion of blocks) becomes necessary for generating secure private keys β. We provide algorithms for construction and factorization with respect to this new type of signatures.

The chapter focuses on a thorough study of the security of the new scheme. Using heuristic and algebraic methods, we establish lower bounds for the workload of conceivable direct attacks on the private key. We develop a powerful chosen plaintext attack called Matrix-permutation attack which allows an adversary to reconstruct partial information about the permutations $\pi_1, \ldots, \pi_s$ and ξ used to shuffle elements within blocks and shuffle the blocks among themselves (transformations T3,T4). Proposition 5.4.5 shows that non-fused transversal logarithmic signatures should not be used for the new MST_3. The complexity of recovering plaintext if the Matrix-permutation attack is applied on MST_3 with fused-transversal logarithmic signatures is stated in Proposition 5.4.7. It shows that the MST_3 remain secure by the Matrix-permutation attack, when fused logarithmic signatures are properly used. Finally, the randomized encryption also prevents the adversary from gaining any partial information about the plaintext.

Chapter 6

Implementation aspects of MST_3

In this chapter, we consider the practical implementation issues of FT-MST_3 with the underlying Suzuki 2-groups.

6.1 Set-up

The Algorithm 9 as decribed in Section 5.2 will be used for generating logarithmic signatures β. As observed, if we keep track of the information at each step of Algorithm 9, in particular the knowledge of partition $\mathcal{P} = \{P_1, \ldots, P_s\}$ used in step 3, we have a highly efficient factorization method with respect to β as shown in Algorithm 10.

By setting up MST_3 with fused transversal logarithmic signature β, using Algorithm 9, we should also take in account the following issues:

I. *Partition $\mathcal{P}$ is chosen in such a way, that no consecutive blocks of δ are fused.*

   Fusing consecutive blocks of transversal logarithmic signatures results in logarithmic signatures that remain transversal (as shown in Remark 2.4.1). Such fused transversal logarithmic signatures are vulnerable to the Matrix-permutation attack and therefore cannot be used to build secure MST_3 with Suzuki 2-groups.

II. *Minimal length of blocks of ε before fusion is 8 (except one or two blocks).*

   If we use the blocks of size 2, the resulting realization of MST_3 can be broken using Matrix-permutation attack.

   For the logarithmic signatures of type $(2, \ldots, 2)$ we obtain the following proposition.

Proposition 6.1.1 *Let ε be a logarithmic signature for elementary abelian group of type $(2, 2, \ldots, 2)$. Then ε and any signature constructed by applying transformations* T1,. . . ,T4 *on ε are tame.*

Sketch of Proof: Let $\varepsilon = \big\{\{e_{1,1}, e_{1,2}\}, \{e_{2,1}, e_{2,2}\}, \ldots, \{e_{s,1}, e_{s,2}\}\big\}$ be a transversal signature of type $(2, 2, \ldots, 2)$ for a vector space V. First, assume that each $e_{i,1} = 0$ (zero vector). Non-zero elements of ε are all linearly independent and form a basis B for V. Then factorization of y with respect to ε equals to representing y in basis B. This is unique and efficiently computed using basic linear algebra.

If there exists an i such that $e_{i,1} \neq 0$, we can use Algorithm 1 followed by a right translation (see Section 2.3.1) to create signature ε' with all $e'_{i,1} = 0$. The result of normalization is an equivalent logarithmic signature and induces the same mapping. The right translation is a “shifting” of every image by a constant.

It is not difficult to show that the resulting logarithmic signature obtained from ε of type $(2, 2, \ldots, 2)$ after applying transformations T1,. . . ,T4 remains transversal. □

II. (cont.) We do not suggest the usage of blocks of size 4 for ε either, except in cases with very large number of blocks or when more than two blocks are fused together (e.g. fuse four blocks of length 4 to one with 256 elements).

    Note that with respect to a logarithmic signature of type $(4, 4, \ldots, 4)$, 3/4 of all elements can be factorized by applying the Basis attack (see Section 5.4.1). Increasing the block length decreases the portion of elements that could be correctly factorized considerably.

Moreover, by taking the discussion of Subsection 5.4.1 into account, we may, if necessary, select the elements of $\alpha_{.a}$ in a subspace V_1 of V such that $\rho = |V|/|V_1|$ is sufficiently large.

6.2 Computing with the Suzuki 2-groups

Let $q = 2^m$, where $m \geqslant 3$ is not a power of 2 and let θ be a non trivial odd-order automorphism of $\mathbb{F}_q$. Let $\mathcal{G} = A(m, \theta)$ be the Suzuki 2-group of order q^2 described in Section 2.6. Recall that the multiplication of two elements in $\mathcal{G}$ is given by the rule:

$$S(a_1, b_1)S(a_2, b_2) = S(a_1 + a_2\ ,\ b_1 + b_2 + a_1 a_2^{\theta}). \tag{6.2.1}$$

We could store the group elements $S(a, b)$ as pairs (a, b) but this would require that we compute some a^{θ} each time we compute a product of group elements. In turn, each computation a^{θ} requires at most $2\lceil \log_2 m \rceil$ multiplications in $\mathbb{F}_q$. It is therefore more time efficient to store the group elements as triples (a, b, a^θ). Thus, the product $S(a_1, b_1) \cdot S(a_2, b_2)$ is identified with the triple

$$(a_1 + a_2\ ,\ b_1 + b_2 + a_1 a_2^\theta\ ,\ a_1^\theta + a_2^\theta)$$

and computation of the product requires just a single multiplication and four additions in $\mathbb{F}_q$. Moreover, to keep the number of squaring operations needed to extend a group element to its triple representation minimal, we choose the *Frobenius automorphism* for θ.

6.3 Public key size and cipher expansion

Let $\alpha = \big((a_{(i,j).a}, a_{(i,j).b})\big)$ and $\gamma = \big((h_{(i,j).a}, h_{(i,j).b})\big)$. For a given i we have $h_{(i,j).a} = a_{(i,j).a} + t_{(i-1).a} + t_{(i).a}$ for all $j = 1, \ldots, r_i$. This means that for each i, if $a_{(i,j).a}$'s and the sum $t_{(i-1).a} + t_{(i).a}$ are known, $h_{(i,j).a}$'s can be derived. Therefore, for the public key we need to store $[\alpha,\ (h_{(i,j).b})]$ (i.e. the $.b$ part of γ) and the s values $t_{(i-1).a} + t_{(i).a}$, for $i = 1, \ldots, s$.

However, for a practical implementation of MST_3 we describe a more efficient method of dealing with the key storage. The idea is that we generate the key by using a publicly known Algorithm $\mathcal{A}$, which generates a random cover α satisfying the conditions in Section 5.1. Essentially, Algorithm $\mathcal{A}$ utilizes a pseudo-random number generator $\mathcal{R}$. To simplify the description, we assume that a logarithmic signature β has been generated by Algorithm 9 separately.

Algorithm 20 Reduced key storage

External: Algorithm $\mathcal{A}$, a pseudo-random number generator $\mathcal{R}$

Input: $[\beta,\ f,\ t_0,\ \ldots, t_s]$, a seed S for $\mathcal{R}$

Output: $[\alpha,\ \gamma]$

```
1: Using $\mathcal{A}$, $\mathcal{R}$ and $S$ generate $\alpha = \big((a_{(i,j).a}, a_{(i,j).b})\big)$
2: Create $\gamma = \big((h_{(i,j).a}, h_{(i,j).b})\big)$ from $\alpha$ as decribed in Section 5.1.
```

From Algorithm 20 it is clear that, for the public key, one has to publish $(h_{(i,j).b})$ together with $t_{(i-1).a} + t_{(i).a}$ for $i = 1, \ldots, s$, and S only. To obtain the complete public key, i.e. $[\alpha, \gamma]$, one first generates α from step 1 using $\mathcal{R}$ and seed S, then computes $(h_{(i,j).a})$ from $a_{(i,j).a}$ and $t_{(i-1).a} + t_{(i).a}$. This approach will reduce the key size of the system roughly to only *one third* of $[\alpha, (h_{(i,j).b})]$ (i.e. one fourth of the size of public-key $[\alpha, \gamma]$). In fact, this key size appears to be the minimum key storage that can be realized for MST_3.

The cipher expansion of MST_3 is of a factor three. For, suppose (y_1, y_2) is a ciphertext pair with $y_1 = (y_{(1).a}, y_{(1).b})$ and $y_2 = (y_{(2).a}, y_{(2).b})$, then, it suffices to send y_1 and $y_{(2).b}$ as the ciphertext. This is because $y_{(2).a}$ can be obtained from the equation $y_{(2).a} = y_{(1).a} + t_{(0).a} + t_{(s).a}$ by using the private key t_0 and t_s.

6.4 Examples of generating β

We need to introduce some notation. We say a logarithmic signature (cover) β is of type $(v_1^{u_1}.v_2^{u_2}.\dots.v_t^{u_t})$ if β has the first u_1 blocks of size v_1, the next u_2 blocks of size v_2, etc.

Let $\varepsilon = [E_1, E_2, \dots, E_s]$ be a transversal logarithmic signature created by Algorithm 9 by steps 1 and 2. Then, there exists a chain of subgroups $1_{\mathcal{G}} = \mathcal{G}_0 < \mathcal{G}_1 < \cdots < \mathcal{G}_s = \mathcal{G}$, such that each block E_i consists of a complete set of coset representatives of $\mathcal{G}_{i-1}$ in $\mathcal{G}_i$.

We also write $[r_{u_1} \times r_{u_2} \times \cdots \times r_{u_\ell}]$ to denote the fusion of ℓ blocks $E_{u_1}, E_{u_2}, \dots, E_{u_\ell}$, where $|E_{u_i}| = r_{u_i}$ and $u_1 < u_2 < \cdots < u_\ell$. Specially, $[r_u]$ denotes a non-fused block E_u with $|E_u| = r_u$. We say a block B_i of β is of fusion type $[r_{u_1} \times r_{u_2} \times \cdots \times r_{u_\ell}]$ if $B_i = E_{u_1} \otimes E_{u_2} \otimes \cdots \otimes E_{u_\ell}$. We write

$$F_1^{e_1}.F_2^{e_2}\dots F_\ell^{e_\ell}$$

to denote the *fusion type of* β for which the first e_1 blocks are of fusion type F_1, the next e_2 blocks of fusion type F_2, etc. For example, the notation $[256]^2.[32\times4]^5.[32\times8\times4]^{10}$ denotes a set-up for β with the first two non-fused blocks of length 256, the next five created by the fusion of two blocks of size 32 and 4, and the remaining ten blocks obtained by fusing three blocks of size 32, 8 and 4 respectively.

Let β be a fused logarithmic signature of fusion type, say, $[v_1]^{e_1}\dots[v_{i-1}]^{e_{i-1}}.[v_i \times u_i]^{e_i}\dots[v_{j-1} \times u_{j-1}]^{e_{j-1}}.[v_j \times u_j \times w_j]^{e_j}\dots[v_\ell \times u_\ell \times w_\ell]^{e_\ell}$, generated by Algorithm 9. If β is used for a set-up of FT-MST_3, the workload of the Matrix-permutation attack required to recover the plaintext is roughly bounded by $\mathcal{O}(v_i^{e_i}\dots v_{j-1}^{e_{j-1}}.(v_j.u_j)^{e_j}\dots(v_\ell.u_\ell)^{e_\ell})$ (see Proposition 5.4.7).

Example 1

Here, we show an example of the set-up for FT-MST_3 as given in the Table 6.2 below. Let $m = 224$ and $s = 32$. The following steps are required to successfully set-up the scheme.

Set-up:

(i) Using Algorithm 9 generate a logarithmic signature β for $\mathcal{Z}$ starting from ε of type $(128^2.32^{30}.4^{30})$, i.e. $v = 62$. In particular, for step 3 of the algorithm take a partition $\mathcal{P} = \{P_1, \dots, P_{32}\}$ with $P_1 = \{1\}, P_2 = \{2\}$, and $P_i = \{i, v - i + 3\}$ for $i = 3, \dots, 30$, and

$P_{31} = \{31, 33\}$, $P_{32} = \{32, 34\}$. After finishing Algorithm 9 the logarithmic signature β is of type (128^{32}), and of fusion type $[128]^2.[32 \times 4]^{30}$.

(ii) Using β and Algorithm 20 create public key $[\alpha, \gamma]$.

Example 2

Let $m = 255$ and $s = 26$.

Set-up:

(i) Using Algorithm 9, generate β for $\mathcal{Z}$ starting from ε of type $(256.32^{25}.8^{24}.4^{25})$, i.e. $\nu = 75$. For the step 3 of this algorithm take a partition $\mathcal{P} = \{P_1, \ldots, P_{32}\}$ with $P_1 = \{1\}, P_2 = \{2, 75\}$, and $P_i = \{i,\ i + 24,\ i + 48\}$ for $i = 3, \ldots, 26$. Therefore β is of type $(256.128.1024^{24})$, and of fusion type $[256].[32 \times 4].[32 \times 8 \times 4]^{24}$.

(ii) Using β and Algorithm 20 create public key $[\alpha, \gamma]$.

6.5 Performance of the system

In this section, we show the data of the performance of MST_3 acquired from a concrete implementation of the scheme.

The Table 6.1 shows the number of operations required for one (randomized) encryption or decryption of the FT-MTS$_3$, namely addition (ADD), multiplication (MULT), exponentiation with θ ($\mathsf{EXP}(\theta)$), generation of m-bit random R (PRNG), and factorization of $\breve{\varepsilon}(R) \in \mathcal{Z}$ with respect to a transversal logarithmic signature ε using the Algorithm 11 (FACTOR).

Table 6.1: Number of basic operations for one encryption/decryption of FT-MST$_3$.

	$\mathbb{F}_{2^m}$ ADD	$\mathbb{F}_{2^m}$ MULT	$\mathbb{F}_{2^m}$ EXP(θ)	$\mathbb{F}_{2^m}$PRNG	FACTOR
encryption	$7s - 7$	$2s - 2$	-	1	-
decryption	$m + 4s + 8$	$s + 3$	2	-	1

We note that an intrinsic property of MST_3 is that there is a trade-off between the key storage and the speed of the scheme. For example, if $\mathbb{F}_{2^{320}}$ is the underlying field for the Suzuki 2-group $\mathcal{G}$, then the corresponding FT-MST$_3$ has an input of 320 bit length; if α and γ have type (4.64^{53}), the public key size is of 135 kBytes, whereas if α and γ have the type (256^{40}), we have a public key of 402 kBytes. This implementation shows that for the first case we have an encryption/decryption speed of 287/471 kB/s, whereas for the second case 377/581 kB/s.

Table 6.2: Various data for parameters, performance and security of FT-MST_3.

m	s	*type of* β	pk [kB]	*fusion type of* β	W	E [kB/s]	D [kB/s]
160	26	$(256^2 \cdot 64^{24})$	43	$[256].[16 \times 4 \times 4].[16 \times 4]^{24}$	2^{102}	607	859
160	23	$(64 \cdot 128^{22})$	57	$[64].[128].[32 \times 4]^{21}$	2^{105}	604	852
160	20	(256^{20})	100	$[256].[16 \times 4 \times 4]^{19}$	2^{114}	671	895
160	18	$(256^2 \cdot 512^{16})$	170	$[256].[16 \times 4 \times 4].[32 \times 4 \times 4]^{16}$	2^{118}	689	904
160	16	(1024^{16})	320	$[1024].[32 \times 8 \times 4]^{15}$	2^{120}	758	941
192	32	(64^{32})	49	$[64]^3.[16 \times 4]^{29}$	2^{116}	571	854
192	28	$(8 \cdot 128^{27})$	82	$[8].[128]^2.[32 \times 4]^{25}$	2^{125}	529	783
192	24	(256^{24})	145	$[256].[16 \times 4 \times 4]^{23}$	2^{138}	609	851
192	22	$(8 \cdot 512^{21})$	253	$[8].[512].[32 \times 4 \times 4]^{20}$	2^{140}	679	914
192	20	$(4 \cdot 1024^{19})$	457	$[4].[1024].[32 \times 8 \times 4]^{18}$	2^{144}	720	924
224	38	$(4 \cdot 64^{37})$	66	$[4].[64]^4.[16 \times 4]^{33}$	2^{132}	511	772
224	32	(128^{32})	113	$[128]^2.[32 \times 4]^{30}$	2^{150}	565	827
224	28	(256^{28})	197	$[256].[16 \times 4 \times 4]^{27}$	2^{162}	595	845
224	25	$(256 \cdot 512^{24})$	344	$[256].[32 \times 4 \times 4]^{24}$	2^{168}	637	875
224	23	$(256 \cdot 64 \cdot 1024^{21})$	597	$[256].[16 \times 4].[32 \times 8 \times 4]^{21}$	2^{172}	678	894
255	43	$(8 \cdot 64^{42})$	85	$[8].[64]^4.[16 \times 4]^{38}$	2^{152}	532	808
255	37	$(8 \cdot 128^{36})$	145	$[8].[128]^2.[32 \times 4]^{34}$	2^{170}	576	852
255	32	$(256^{31} \cdot 128)$	252	$[256].[16 \times 4 \times 4]^{30}.[32 \times 4]$	2^{185}	602	865
255	29	$(8 \cdot 512^{28})$	447	$[8].[512].[32 \times 4 \times 4]^{27}$	2^{189}	637	887
255	26	$(256 \cdot 128 \cdot 1024^{24})$	778	$[256].[32 \times 4].[32 \times 8 \times 4]^{24}$	2^{197}	708	932
288	48	(64^{48})	110	$[64]^5.[16 \times 4]^{43}$	2^{172}	306	502
288	41	$(256 \cdot 128^{40})$	190	$[256].[128].[32 \times 4]^{39}$	2^{195}	325	523
288	36	(256^{36})	325	$[256]^2.[16 \times 4 \times 4]^{34}$	2^{204}	381	593
288	32	(512^{32})	577	$[512].[32 \times 4 \times 4]^{31}$	2^{217}	407	595
288	29	$(512^2 \cdot 1024^{27})$	1009	$[512].[32 \times 4 \times 4].[32 \times 8 \times 4]^{27}$	2^{223}	457	668
320	54	$(4 \cdot 64^{53})$	135	$[4].[64]^5.[16 \times 4]^{48}$	2^{192}	287	471
320	46	$(8 \cdot 512 \cdot 128^{44})$	242	$[8].[512].[32 \times 4]^{44}$	2^{220}	305	490
320	40	(256^{40})	402	$[256]^2.[16 \times 4 \times 4]^{38}$	2^{228}	377	581
320	36	$(32 \cdot 512^{35})$	703	$[32].[512].[32 \times 4 \times 4]^{34}$	2^{238}	403	604
320	32	(1024^{32})	1281	$[1024].[32 \times 8 \times 4]^{31}$	2^{248}	450	650
352	59	$(16 \cdot 64^{58})$	163	$[16].[64]^6.[16 \times 4]^{52}$	2^{208}	246	408
352	51	$(4 \cdot 128^{50})$	277	$[4].[128]^3.[32 \times 4]^{47}$	2^{235}	292	475
352	44	(256^{44})	486	$[256]^2.[16 \times 4 \times 4]^{42}$	2^{252}	304	481
352	40	$(2 \cdot 512^{39})$	860	$[2].[512].[32 \times 4 \times 4]^{38}$	2^{266}	352	537
352	36	$(8 \cdot 512 \cdot 1024^{34})$	1431	$[8].[512].[32 \times 8 \times 4]^{34}$	2^{272}	378	566
384	64	(64^{64})	195	$[64]^6.[16 \times 4]^{58}$	2^{232}	252	421
384	55	$(64 \cdot 128^{54})$	330	$[64].[128]^3.[32 \times 4]^{51}$	2^{255}	287	466
384	48	(256^{48})	578	$[256]^2.[16 \times 4 \times 4]^{46}$	2^{276}	303	485
384	43	$(64 \cdot 512^{42})$	1013	$[64].[512].[32 \times 4 \times 4]^{41}$	2^{287}	352	535
384	39	$(16 \cdot 1024^{38})$	1827	$[16].[1024].[32 \times 8 \times 4]^{37}$	2^{296}	364	554

The Table 6.2 presents data related to the public key size, type and fusion type for β, the speed of the encryption and decryption together with the workload (W) of the Matrix-permutation attack required to recover the plaintext. The performance tests were implemented by using the NTL programming library (see [Ntl]) and measured on a 64-bit machine of 1.8 GHz.

6.6 Conclusions

This chapter deals with implementation issues of the revised MST_3 scheme. We provide a practical method to reduce the ciphertext expansion and the size of the public key. The data of performance obtained from an experimental result are summarized in Table 6.2. These may be used as a guide for choosing appropriate parameters for MST_3 with Suzuki 2-groups.

Chapter 7

Pseudorandom Number Generators Based on Random Covers for Finite Groups

7.1 RNG and PRNG

Generating random bit sequences is an important problem in cryptography. The security of many cryptographic systems depends on the generation of unpredictable bit sequences. Such sequences are used, for example, in stream ciphers, digital signature schemes, key materials of encryption schemes, in challenge-response identification systems, and in many other cryptographic protocols. There are two main concepts for generating random bit sequences. The first one uses non-deterministic schemes producing bit sequences that cannot be reproduced. Such schemes are usually called random number generators (RNG). These generators exploit natural sources of randomness, such as radioactive decay, thermal noise, air turbulence within a physical system, frequency instability of oscillator, etc. and they appear in the form of a hardware device. Another type of RNGs is in the form of a software program exploiting certain physical "random" sources such as content of buffers, mouse movement, and many other different operating system values in a computer. It should be noted that RNGs become impractical in applications such as stream ciphers or one-time-pads where a large sequence of bits needs to be securely transmitted or stored. The second class generates bit sequences using deterministic algorithms. Generators in this class are often referred to as pseudorandom number generators (PRNG). Such generators are given a random bit sequence of length $\mathtt{k}$ as input, and output a bit sequence of length $\mathtt{l} \gg \mathtt{k}$, which appears to be random. The input bit sequence to the PRNG is called the *seed* and the output is called a *pseudorandom bit sequence*. Although it is impossible to provide a mathematical proof that a generator indeed produces random bit sequences (uniformly distributed), various statistical tests can help to detect certain weaknesses the generator may have. It should be noted that from the nature of a PRNG the entropy of the output can never exceed the entropy of the seed. However, it can be computationally infeasible to distinguish a good PRNG from a perfect RNG. We suggest that the reader should consult [Knu98, MOV97] for information about pseudorandom number generators.

In this chapter, we introduce a new approach to designing PRNGs based on random covers of finite groups.

It should be noted that in [MOS84, MM89b] logarithmic signatures for permutation groups are used to construct pseudorandom number generators and to generate random permutations in the symmetric group. A logarithmic signature α of symmetric group $\mathfrak{S}_n$, induces a mapping $\breve{\alpha} : \mathbb{Z}_{|\mathfrak{S}_n|} \to \mathfrak{S}_n$. Hence it seems natural to use logarithmic signatures for generating random elements in a group, i.e. given a seed s_0, we could compute the sequence $\breve{\alpha}(s_0)$, $\breve{\alpha}(s_0 + 1)$, ..., $\breve{\alpha}(s_0 + \ell - 1)$ of ℓ permutations. The authors of [MOS84] claim that this sequence behaves like a sequence of random permutations and undertake statistical tests to substantiate this result.

7.2 PRNG based on random covers

In this section, we present the basic principle of building a generic PRNG based on random covers for finite groups. The most striking property of random covers regarding cryptographic applications is that they induce a large class of functions that behave randomly.

There are no restrictions on the group structure for the PRNG in this generic case. We call our random cover based pseudorandom number generator MSTg. From now on let $\mathcal{G}_1$ and $\mathcal{G}_2$ be two chosen finite groups with $|\mathcal{G}_1| = n$, $|\mathcal{G}_2| = m$ and $n \geqslant m$. In case that $\mathcal{G}_1$ contains a subgroup of order m, then we will choose $\mathcal{G}_2$ as such a subgroup. In particular, if $n = m$ we have $\mathcal{G}_1 = \mathcal{G}_2$.

Let ℓ be a integer such that $\ell \geqslant n$. Let $k \geqslant 0$ be a fixed integer.

Let α be a random cover of type $(u_1, u_2, \ldots, u_t)$ for $\mathcal{G}_1$ with $\prod_{i=1}^{t} u_i = \ell$. Let $\alpha_1, \ldots, \alpha_k$ be a set of random covers of type $(r_1, r_2, \ldots, r_s)$ for $\mathcal{G}_1$ with $\prod_{i=1}^{s} r_i = |\mathcal{G}_1|$. Let $\gamma = [H_1, H_2, \ldots, H_s]$ be a random cover of type $(r_1, r_2, \ldots, r_s)$ for $\mathcal{G}_2$. We assume that there are bijective mappings $f_1 : \mathcal{G}_1 \to \mathbb{Z}_n$ and $f_2 : \mathcal{G}_2 \to \mathbb{Z}_m$, which efficiently identify elements of $\mathcal{G}_1$ with numbers in $\mathbb{Z}_n$ and elements of $\mathcal{G}_2$ with numbers in $\mathbb{Z}_m$.

We define a function

$$F : \mathbb{Z}_\ell \longrightarrow \mathbb{Z}_m$$

as a composition of mappings as follows.

$$\mathbb{Z}_\ell \xrightarrow{\breve{\alpha}} \mathcal{G}_1 \xrightarrow{f_1} \mathbb{Z}_n \xrightarrow{\breve{\alpha}_1} \mathcal{G}_1 \xrightarrow{f_1} \mathbb{Z}_n \longrightarrow \cdots \xrightarrow{\breve{\alpha}_k} \mathcal{G}_1 \xrightarrow{f_1} \mathbb{Z}_n \xrightarrow{\breve{\gamma}} \mathcal{G}_2 \xrightarrow{f_2} \mathbb{Z}_m, \qquad (A)$$

Another alternative to define the function $F : \mathbb{Z}_\ell \longrightarrow \mathbb{Z}_m$ is as follows.

$$\mathbb{Z}_\ell \xrightarrow{\breve{\alpha}} \mathcal{G}_1 \xrightarrow{f_1} \mathbb{Z}_n \xrightarrow{\breve{\gamma}} \mathcal{G}_2 \xrightarrow{f_2} \mathbb{Z}_m \xrightarrow{\breve{\delta}_1} \mathcal{G}_2 \xrightarrow{f_2} \mathbb{Z}_m \longrightarrow \cdots \xrightarrow{\breve{\delta}_k} \mathcal{G}_2 \xrightarrow{f_2} \mathbb{Z}_m, \qquad (B)$$

where $\delta_1, \ldots, \delta_k$ is a set of random covers of type $(v_1, v_2, \ldots, v_w)$ for $\mathcal{G}_2$ with $\prod_{i=1}^{w} v_i = |\mathcal{G}_2|$. It turns out that the function F has a strong random behavior, when the involved covers are randomly generated. Using F we define the MSTg in the following algorithm.

Algorithm 21 MSTg: Pseudorandom number generator based on covers for finite groups

Input: Integers ℓ, m, function $F : \mathbb{Z}_\ell \longrightarrow \mathbb{Z}_m$ as defined above, a random and secret seed $s_0 \in \mathbb{Z}_\ell$, a constant $C \in \mathbb{Z}_\ell$.

Output: t pseudorandom numbers $z_1, \ldots, z_t \in \mathbb{Z}_m$.

```
1: for $i$ from 1 to $t$ do
2:     $s_i \leftarrow (s_{i-1} + C) \bmod \ell$
3:     $z_i \leftarrow F(s_i)$.
4: end for
5: return $(z_1, \ldots, z_t)$
```

The MSTg as presented in Algorithm 21 uses a simple counter mode to generate output sequences of pseudorandom numbers via the function F. However, we see that any other suitable mode can be used in place of the counter mode in the algorithm. Since function F is the core of MSTg, which is designed for cryptographic applications, great care must be taken in the generation of F. In other words we need a good RNG or PRNG, for example the Blum-Blum-Shub, or Mersenne Twister PRNG to create the random covers involved in F. Assume that we have chosen such a generator. Using a randomly selected seed, the generator will generate random group elements for the covers involved in defining F. For example, to generate k random covers α_i of type $(r_1, \ldots, r_s)$ for $\mathcal{G}_1$ the generator will generate an output of $k \sum_{i=1}^{s} r_i$ numbers in $\mathbb{Z}_n$. By using $f_1^{-1} : \mathbb{Z}_n \longrightarrow \mathcal{G}_1$ we obtain from these numbers $k \sum_{i=1}^{s} r_i$ group elements in $\mathcal{G}_1$, which are then used to form the k random covers α_i.

It is worthy mentioning that the induced mapping $\breve{\beta} : \mathbb{Z}_n \longrightarrow \mathcal{G}_1$, as described in Chapter 2, where β is a transversal logarithmic signature of $\mathcal{G}_1$, could be used for f_1^{-1}, see for instance [LMTW09, MST02]. For a certain class of abelian groups the mapping $f_1 : \mathcal{G}_1 \longrightarrow \mathbb{Z}_n$

becomes the trivial identity mapping, as we will see in the next section, and hence each number in $\mathbb{Z}_n$ may be used as a group element as well.

Suppose the groups $\mathcal{G}_1$ and $\mathcal{G}_2$ of order n and m have been chosen. How large does the value k need to be? The question is inherently related to the performance and quality of MSTg. For if k would need to be large, then a large amount of storage would be required to set up the function F and consequently the speed of computation with F would necessarily be reduced. Surprisingly, our experimental investigation based on statistical tests shows that when $\mathcal{G}_1$ and $\mathcal{G}_2$ are elementary abelian 2-groups (i.e. $\mathcal{G}_1$ and $\mathcal{G}_2$ may be viewed as vector spaces over $\mathbb{F}_2$), then $k = 0$ or 1 are sufficient. It means only two or three covers are required for constructing function F. The implication is that MSTg is, in fact, a good pseudorandom generator. Moreover, for these groups we may even ignore the mappings f_1 and f_2, therefore the performance of the generator is highly efficient, as presented in the subsequent section.

7.3 Statistical testing of MSTg

In this section, we investigate the randomness of the output bit sequences from MSTg, for which the underlying groups are elementary abelian 2-groups. The group operation will be written additively as XOR in this case, so that the groups may be viewed as vector spaces over $\mathbb{F}_2$. Two significant randomness test suites are used for this purpose. The first one is the NIST Statistical Test Suite in the Special Publication 800-22 Revision 1a (Revised: April 2010) issued by the National Institute of Standards and Technology. The second is the DIEHARD Statistical Tests developed by George Marsaglia. The elaborate statistical tests we have carried out have two main aims. First we investigate the quality of the approach of using random covers for building MSTg. Secondly we evaluate the qualities of the binary output sequences from an arbitrary chosen MSTg. Further, we compare the test results of MSTg with those of several well-known pseudorandom number generators.

To begin with we make a brief description of the strategy for statistical analysis of a random number generator by the NIST test, see [Nist]. The NIST Statistical Test Suite consists of fifteen core tests that, by reason of different parameter inputs, can be considered as 188 statistical tests. The input parameters for each statistical test are fixed such as sequence length N (of the test binary sequence) with $10^3 \leqslant N \leqslant 10^7$, sample size $M \geqslant 55$ (the number of binary sequences), significance level equal to 0.01.

(1) For each statistical test and each binary sequence (of length N), a P-value is computed from the test statistic of this specified statistical test. A success/failure is assigned to this P-value on the basis of the significance level.

(2) For each statistical test and each sample two further evaluations are made. The first one is the proportion of binary sequences passing the statistical test. The range of

acceptable proportions is determined by the significance level and the sample size M. The second evaluation is an additional P-value computed on the basis of χ^2 test applied to the P-values in the entire sample. This additional P-value is examined to ensure the uniformity of the test sequences and is computed, based on the distribution of P-values obtained for the statistical test on the 10 equally divided sub-intervals between 0 and 1.

(3) A sample is considered to have passed a statistical test if the proportion in step (2) is in the interval of acceptance and the additional P-value in step (2) exceeds a certain value. If either of the two evaluations in step (2) is not fulfilled, then the sample is labeled as suspect. If this occurred, additional samples would need to be tested.

The Diehard Battery of Tests of Randomness is provided by George Marsaglia [Diehard]. The Diehard Test Suite is composed of 18 tests and as the number of P-value varies over tests, it provides 219 P-values entirely. An input for a Diehard test is a binary sequence of size 10 MBytes or 12 MBytes. Contrary to the NIST test suite, the Diehard test suite does not suggest a method of how to interpret the test results. These are often evaluated (and interpreted) differently. We will make a stringent condition to interpret the results from the Diehard test by requiring that all the P-values have to belong to some fixed chosen interval. This is in fact a strong requirement, because for a truly random sequence, not all of its P-values would necessary belong to such a fixed interval. Under such a criterion the proportion of sequences passing the Diehard test will strongly be reduced.

From now on we assume that $\mathcal{G}_1$ is an elementary abelian 2-group with XOR as group operation with $|\mathcal{G}_1| = 2^{e_1}$ and $\mathcal{G}_2$ is a subgroup of $\mathcal{G}_1$ with $|\mathcal{G}_2| = 2^{e_2}$. We study MSTg based on this class of groups.

7.3.1 Evaluation of the approach to designing MSTg

In this section, we apply statistical tests of randomness to evaluate the method of using random covers for MSTg (of method (A) in Section 7.2). The strategy of our evaluation consists in generating a random MSTg that, in turn by using a random seed, generates one sample of bit sequence of certain length. The sample is then tested for randomness using NIST and Diehard. This process will be repeated for a large number of randomly generated MSTg. The percentage of MSTg that passes this special type of test including other data obtained from the tests will be then used to compare with other known pseudorandom number generators under the same defined condition.

For the test we consider MSTg generated by one, two or three random covers. For each of these types we have done the following test using the NIST test suite.

(a) Generate an MSTg at random by using the Blum-Blum-Shub PRNG.

(b) Generate a sample of 100 bit sequences of size 10^7 using this MSTg. Apply the NIST statistical tests to this sample, where an input for each test is a bit sequence of size 10^7 (i.e. one tests 100 times of different bit sequences of size 10^7 produced by this specific MSTg).

(c) Step (a) and step (b) are repeated 1056 times.

For the Diehard test the three steps are as follows.

(a') Generate an MSTg at random by using the Blum-Blum-Shub PRNG.

(b') Generate a bit sequences of size 10 MBytes using this MSTg. Apply the Diehard statistical tests to this bit sequence.

(c') Step (a') and step (b') are repeated 50000 times.

To compare the results we have tested Blum-Blum-Shub PRNG [BBS86], ARC4 [Riv92], and Mersenne Twister [MN98] with the same configuration. For example, we randomly generate 1056 Blum-Blum-Shub PRNG, each of them will generate from a random seed so many bit sequences as in step (b) and step (b') for the NIST test and the Diehard test.

This way provides an adequate comparison of the test results of MSTg with other generators. Table 7.1 presents the NIST statistical test of different versions of MSTg including the generators Blum-Blum-Shub (BBS), ARC4 and Mersenne Twister (MT). Note that Mersenne Twister is not suitable for cryptographic applications.

Table 7.1: NIST test results on 1056 MSTg.

PRNG	version [b]	out [b]	#c	C	f_{aver}	f_0/R	f_{max}	f_0	f_1	f_2	f_3	f_4	f_{5+}
$\mathsf{MST_g}$	256/128	128	2	1	0,762	0,479	7	506	366	127	48	6	3
$\mathsf{MST_g}$	256/128	128	3	1	0,798	0,450	5	475	389	131	53	7	1
$\mathsf{MST_g}$	256/128	128	1	p_1	0,743	0,490	5	517	356	135	35	11	2
$\mathsf{MST_g}$	256/128	128	2	p_1	0,686	0,514	6	543	354	116	37	4	2
$\mathsf{MST_g}$	256/128	128	3	p_1	0,720	0,509	5	537	343	129	32	12	3
BBS	1024	1			0,742	0,467	7	493	392	129	37	4	1
ARC4	2048	8			0,690	0,513	4	542	331	154	26	3	0
MT	64	64			0,784	0,493	5	521	335	134	41	23	2

p_1, R [1]

[1] $p_1 = 40938685753732063808824485997586458199112190403957785849807364652597453052901$, $R = 1056$. All statistical tests have been carried out on the CRAY XT6m. For each PRNG (8 together) in Table 7.1 we carried out 1056 tests using the NIST Statistical Test Suite. Each test analyzed an output

Some explanations need to be included for the reading of the table. As described above a test on a sample of 100 bit sequences of size 10^7 in step (b) reports a set of 188 p-values, for short we call it p-value set. Thus there are 1056 p-value sets after step (c). The Column f_0 gives the total number of p-value sets that pass the NIST tests after step (c). The column f_i, $i = 0, 1, 2, 3, 4$ records the entire number of p-value sets, which have exactly i "defect" p-values; column f_{5+} shows number of 5 or more "defect" p-values. The column f_{max} gives the maximum number and f_{aver} the average number of defect p-values (from 188 p-values) for each NIST test. Column #c gives the number of covers used in the corresponding version of MSTg. Column C indicates whether the constant C used in the counter mode is 1 or a prime p_1.

Table 7.2 records the test data of the Diehard test suite and shows the number of tests whose all 219 p-values belong to a certain given interval. Our stringent criterion to interpret a generator as having passed the Diehard test requires that there is a test sample for which all its 219 p-values are within the interval I_1. Column FB/R records the ratio of the so called "FAILS BIG" p-values, i.e. p-values that are not in I_5. The intervals can be found in Table 7.3.

Table 7.2: Diehard test results on 50000 MSTg.

PRNG	version [b]	out [b]	#c	C	I_1	I_2	I_3	I_4	I_5	FB/R
MST_g	256/128	128	2	1	2933	33694	46329	48737	49319	0.0145
MST_g	256/128	128	3	1	3109	34034	46317	48756	49319	0.0144
MST_g	256/128	128	1	p_1	3070	34010	46280	48780	49314	0.0146
MST_g	256/128	128	2	p_1	2963	33835	46337	48746	49323	0.0143
MST_g	256/128	128	3	p_1	3021	34115	46320	48735	49325	0.0143
BBS	1024	1			3047	33860	46316	48722	49303	0.0147
ARC4	2048	8			2971	33941	46200	48710	49334	0.0142
MT	64	64			2951	33701	46292	48724	49287	0.0150

p_1,I_i, R [2]

sequence of 10^9 bits generated by using a unique seed. Moreover, MSTg output bit sequences are generated from different generators. On a single processor one test of a 10^9 bit sequence took approximately 5 hours.

[2] $p_1 = 40938685753732063808824485997586458199112190403957785849807364652597453052901$, $R =$ 50000. For each PRNG in Table 7.2 we carried out 50000 tests using the Diehard Battery of Test. Each test analyzed an output bit sequence of 10 MB generated by using a unique seed. Moreover, MSTg output bit sequences are generated from different generators.

Table 7.3: Intervals for evaluation of Diehard test results on MSTg.

I_1	[0.005, 0.995]
I_2	[0.0005, 0.9995]
I_3	[0.00005, 0.99995]
I_4	[0.000005, 0.999995]
I_5	[0.000001, 0.999999]

7.3.2 Evaluation of a given MSTg

In the previous subsection, the NIST test results show that on average one from two randomly generated MSTg passes the NIST test using its first output sample of bit sequence of size 100×10^7. The NIST test shows the same behavior of the BBS, ARC4, MT generator (i.e. for example, on average, one from two randomly generated BBS generators passes the NIST test with its first output sample of size 100×10^7). However, we know that BBS as a cryptographically secure generator, where any bit from its output is unpredictable, will pass the Diehard test, i.e. any BBS generator will pass the NIST test if further output samples are tested. A question then arises. Will any randomly generated MSTg then pass the NIST test and the Diehard test when sufficient output bit sequences are tested?

In this subsection, we investigate the question by conducting further statistical tests. Our tests are described as follows.

(i) Generate an MSTg at random by using the Blum-Blum-Shub PRNG.

(ii) Generate a sample of 10 bit sequences of size $10^7 \times 100$ using this MSTg. Apply the NIST statistical tests to this sample, where an input for each test is a bit sequence of size 10^7 (i.e. one tests 10 different bit sequences of size $10^7 \times 100$ produced by this specific MSTg).

(iii) Step (i) and step (ii) are repeated 72 times.

For the Diehard test step (ii) is replaced by (ii').

(ii') Generate a sample of 100 bit sequences of size 10 MBytes using this MSTg. Apply the Diehard statistical tests to each bit sequence of size 10 MBytes in this sample (i.e. we test 100 different bit sequences of size 10 MBytes each).

Table 7.4 displays in column "$\leqslant$ t" the number of MSTg (from a set of 72 randomly generated MSTg), that pass the NIST test within the first t samples (of bit sequences of size $10^7 \times 100$). The results has shown that all 72 MSTg pass the NIST test.

Table 7.4: NIST Test Results for each given MSTg.

version [b]	#c	C	Samples									
			$\leqslant 1$	$\leqslant 2$	$\leqslant 3$	$\leqslant 4$	$\leqslant 5$	$\leqslant 6$	$\leqslant 7$	$\leqslant 8$	$\leqslant 9$	$\leqslant 10$
256/128	2	p_1	41	58	65	69	71	71	71	71	72	72
256/128	3	p_1	34	50	60	66	70	71	72	72	72	72

Similarly, Table 7.5 presents the results of the Diehard test on 72 randomly generated MSTg. Each MSTg generates 100 samples of size 10 MB which are tested. Again, the results show that all 72 MSTg pass the Diehard test under the stringent criterion that all 219 p-values have to belong to the interval I_1.

Table 7.5: Diehard Test Results for each given MSTg.

version [b]	#c	C	Samples									
			$\leqslant 10$	$\leqslant 20$	$\leqslant 30$	$\leqslant 40$	$\leqslant 50$	$\leqslant 60$	$\leqslant 70$	$\leqslant 80$	$\leqslant 90$	$\leqslant 100$
256/128	1	p_1	33	53	62	66	69	72	72	72	72	72
256/128	2	p_1	32	55	64	67	69	70	72	72	72	72
256/128	3	p_1	33	53	64	67	69	69	70	71	72	72

The results of the tests above show that any randomly generated MSTg passes the NIST as well as the Diehard test.

7.4 Security of MSTg

The test results in the previous section show in particular that even an MSTg with only one random cover already generates good output sequences that pass the NIST test and the Diehard test. For cryptographic applications we suggest that two or three random covers should be used for each MSTg. Again let $\mathcal{G}_1$ be an elementary abelian 2-group of order $|\mathcal{G}_1| = 2^{e_1}$ having XOR as the group operation. Let $\mathcal{G}_2$ be a subgroup of $\mathcal{G}_1$ with $|\mathcal{G}_2| = 2^{e_2}$. Suppose we construct a random MSTg (of type (A) as shown in Section 7.2). For simplicity we choose, for example, $\ell = 2^{e_1} = |\mathcal{G}_1| = n$, $k = 0$ and we generate two random covers α and γ for $\mathcal{G}_1$ and $\mathcal{G}_2$. Then $F = \breve{\gamma} \circ \breve{\alpha} : \mathbb{Z}_{2^{e_1}} \longrightarrow \mathbb{Z}_{2^{e_2}}$, the composition of $\breve{\alpha} : \mathbb{Z}_{2^{e_1}} \longrightarrow \mathbb{Z}_{2^{e_1}}$ and $\breve{\gamma} : \mathbb{Z}_{2^{e_1}} \longrightarrow \mathbb{Z}_{2^{e_2}}$ is the function presenting the generator. Note again that the bijections f_1 and f_2 for the identification of $\mathcal{G}_1$ with $\mathbb{Z}_{2^{e_1}}$ and $\mathcal{G}_2$ with $\mathbb{Z}_{2^{e_2}}$ become the identity (trivial) functions and therefore can be ignored.

One of the main questions regarding the security of this MSTg is whether or not the function F can be computationally invertible. Put in another way, for any given output sequence can we determine the seed which has been used to generate it? In the following, we show an argumentation giving the negative answer to the question.

If we would make use of the cryptographic hypothesis as stated in the Chapter 2, that random covers induce one-way functions, we would readily have the answer. However, we provide a proof without using the hypothesis when covers are used in an appropriate manner.

To begin with we first make a simple but important remark that we can ensure that α and γ together cannot be replaced by a single random cover β for $\mathcal{G}_2$ having the same type as α. This can be checked efficiently by computing at most $\sum_{i=1}^{s} r_i$ appropriate chosen inputs for α, where $(r_1, \dots, r_s)$ is the type of α. We will not treat this question here. However, we would like to mention that the probability, for which α and γ can be replaced by a single cover β, is actually negligible.

Now the mapping $\breve{\gamma}: \mathbb{Z}_{2^{e_1}} \longrightarrow \mathbb{Z}_{2^{e_2}}$ could be considered as a compression function of e_1 bit to e_2 bit. Since this function is constructed using random group elements of $\mathcal{G}_2$ it is expected that for each output $z \in \mathbb{Z}_{2^{e_2}}$ there are on average $2^{e_1-e_2}$ elements $y \in \mathbb{Z}_{2^{e_1}}$ such that $\breve{\gamma}(y) = z$, see [ST07]. Let $Y \subset \mathbb{Z}_{2^{e_1}}$ denote the set of $2^{e_1-e_2}$ preimages y of z. It should be noted that the set Y is not determined and experimental results also show that Y behaves like a random set taking from the universe of size 2^{e_1}. Define $X := \breve{\alpha}^{-1}(Y)$. Again, on average, we may expect that $|X| = 2^{e_1-e_2-\delta}$ with $0 \leqslant \delta \leqslant 2$, [ST10]. This set X must contain the seed s_0, for which $F(s_0) = z$. An exhaustive search needs to be done to determine s_0, and this search requires a complexity of size $\mathcal{O}(2^{e_1-e_2-\delta-1})$. Thus if $e_1 - e_2$ is sufficiently large, say $e_1 - e_2 \geqslant 100$, it is computationally impossible to determine s_0.

The one-way-ness of F, in fact, provides sufficient evidence about the security of MSTg against the determination of the seed when long output sequences are produced. This is similar to the case where strong encryption systems, for example AES, RSA, are used to build pseudorandom number generators.

An interesting property of the mappings $\breve{\alpha}$ and $\breve{\gamma}$ is that their outputs should have a strong independency. For example, consider $\breve{\alpha}$. Let x and x' be two different given input for $\breve{\alpha}$. Let $\alpha = [A_1, \dots, A_s]$ with $A_i = \{a_{i,1}, \dots, a_{i,j_{r_i}}\}$, $i = 1, \dots, s$. Assume $\breve{\alpha}(x) = \sum_{i=1}^{s} a_{i,j_i}$ and $\breve{\alpha}(x') = \sum_{k=1}^{s} a_{i,k_i}$. Then there exists at least a value i such that $j_i \neq k_i$. On the other hand since the elements for the cover are randomly generated from a large group $\mathcal{G}_1$ and since the proportion $\sum_{i=1}^{s} r_i/|\mathcal{G}_1|$ is very small, it is with a very high probability that $a_{i,j_i} \neq a_{i,k_i}$ (a precise formula for the probability can be derived by using the results from the classical occupancy problem, see for example [MOV97], p. 53). This implies that if we want to obtain $\breve{\alpha}(x')$ from $\breve{\alpha}(x)$ by changing its bits, then on average every bit of $\breve{\alpha}(x)$ has the probability 1/2 of being changed. Put in another way, with high probability $\breve{\alpha}(x)$ and $\breve{\alpha}(x')$ differ in a half of their bits. The fact that any MSTg with one random cover passes the randomness tests as shown above (i.e. their output bit sequences are not distinguishable from a truly random sequence) is further evidence proving this property.

Return to the case of MSTg with two random covers. Let $z_1 = \mathsf{F}(s_0)$ and $z_2 = \mathsf{F}(s_0 + \mathsf{C} \bmod 2^{e_1})$ be two outputs of MSTg, for which s_0 is unknown. From the discussion above $y_1 = \breve{\alpha}(s_0)$ and $y_2 = \breve{\alpha}(s_0 + \mathsf{C} \bmod 2^{e_1})$ are independent, thus the information of z_1 and z_2 does not in general reduce the complexity of finding the correct preimage y_1 of z_1 (i.e. $y_1 = \breve{\gamma}^{-1}(z_1)$), which is then used to determine s_0. Finally, note also that the composition of $\breve{\alpha}$ and $\breve{\gamma}$ makes use of two incompatible operations. The first one is the XOR operation and the second is the partition of a given bit sequence of length e_1 into s subsequences of size $\log_2 r_1, \ldots, \log_2 r_s$.

Determining an accurate complexity of computing the seed s_0 for a given output sequence $z_1, \ldots, z_t$ of MSTg appears to be very difficult, but such a rigorously mathematical result would prove the security of the generator. Thus, it is worthy to pursue further investigation.

7.5 Performance of MSTg

Experimental results have shown a high efficiency of MSTg when $\mathcal{G}_1$ and $\mathcal{G}_2$ are elementary abelian 2-groups with XOR as the group operation. Beside the security and quality of "randomness" of the outputs, the performance efficiency is an important criterion for a practical use of a pseudorandom number generator.

We may envisage the following method of using of MSTg. Assume we have a fixed pseudorandom number generator (or a truly random number generator), say GEN.

- Use a secret random seed s_0 for GEN to set up the covers to build a version of MSTg, just as described in Section 7.2.
- Use a secret random seed s_1 to generate output bit sequences using MSTg.

Note that the GEN could be a given version of MSTg, which we call $\mathsf{MasterMSTg}$. After each generation of a generator MSTg, the $\mathsf{MasterMSTg}$ may, in turn, be updated in which we partially or entirely replace the group elements in its random covers by new random elements. The whole process of generating MSTg, therefore, does not rely on any fixed GEN or any fixed set of data.

This method of using MSTg is similar to the concept of one-time-pad. Namely, to generate a sequence of random bits of a certain length, we randomly generate an MSTg, which is then used once to generate the bit sequence. For generating a new bit sequence, a new MSTg will then be created. Also the $\mathsf{MasterMSTg}$ is changed each time.

Table 7.6 shows data of performance of MSTg with several parameter settings.

Table 7.6: Data of performance of MSTg.

version [b]	#c	C	$s\|r_i$	Speed [MB/s]	Memory [kB]
256/128	1	p_1	$32\|2^8$	125.4	128
256/128	2	1	$32\|2^8$	48.3	384
256/128	2	p_1	$32\|2^8$	45.7	384
256/128	3	1	$32\|2^8$	24.4	640
256/128	3	p_1	$32\|2^8$	23.8	640
192/64	1	p_2	$24\|2^8$	152.0	48
64/64	1	p_3	$8\|2^8$	622.4	16
64/64	2	p_3	$8\|2^8$	409.8	48

p_i[3]

7.6 Conclusions

This chapter introduces a new approach to designing pseudorandom number generators, called MSTg, by using random covers of finite groups. In particular, elementary abelian 2-groups proposed for the actual implementation turn out to be highly efficient and produce high-quality random bit sequences. Results from a very extensive sequence of tests for randomness using the NIST Statistical Test Suite and Diehard Battery of Tests show the excellent properties of the method. More importantly, we provide evidence that this class of generators is suitable for cryptographic applications. Finally, we discuss the implementation issues and include data of its performance obtained from an experimental implementation. We propose a method how to use MSTg in practice.

[3] $p_1 = 409386857537320638088244859975864581991121904039577858498073646525974530529 01$, $p_2 = 384418465689213814520715754733436629680286711031893386921 7$, $p_3 = 11895088228400396813$. Measurement has been made on 64-bit Linux Xeon machine with 2.4 GHz, 128 GB RAM and 16 MB Cache, and using NTL C++ Library (see [Ntl]).

Chapter 8

Summary and Further Research

In this thesis, we investigate the use of logarithmic signatures and random covers for finite groups in cryptography. Our primary objective is to show that the cryptosystem MST_3 can be realized with Suzuki 2-groups.

To set-up the cryptosystem MST_3, we require efficient methods of constructing covers and logarithmic signatures of the underlying group, both forming the basis of the system. We start with the investigating the problem of generating random covers for large finite groups. By revealing the connection of this problem with the classical occupancy problem, we determine a bound for the probability for which randomly chosen collection of group elements compose a cover. Thus, we solve the problem of generating random covers for arbitrary large groups and provide a method of constructing them in a very natural and highly efficient way, i.e. the elements of a cover can be chosen randomly. The crucial point that makes these structures useful for group based cryptography is that factorization with respect to a cover is presumedly intractable, i.e. the covers essentially induce one-way functions. The proof of this hypothesis appears to be difficult and remains a challenge. This problem is of course not only significant regarding cryptographic purposes but also interesting from the group-theoretic point of view; and it is worth pursuing further investigations.

For a possible realization of MST_3, the Suzuki 2-groups have been suggested. Due to their simple structure, the groups enable us to study the security of the system and also provide an efficient implementation. For the first realization, a special class of canonical logarithmic signatures for elementary abelian 2-groups has been proposed as a basis for the key generation. These are easy to construct and allow highly efficient factorization. We develop an attack showing that canonical signatures cannot be used to build a secure realization of MST_3 with Suzuki 2-groups.

We continue to investigate the realization of MST_3 and propose a new variant with significant improvement, strengthening the security of the scheme. For that purpose, we re-design the set-up of the scheme and introduce a class of fused transversal logarithmic signatures. These allow an efficient factorization if we keep track of the transformations used to generate them, i.e. this information provide a trap-door for the factorization with respect to fused transversal logarithmic signatures. Recently, some effort has been given to the study of factorization with respect to fused transversal logarithmic signatures (if the trap-door information is not known), see [BCM09]. The problem however, in general, remains unsolved and should be targeted in future.

We provide a thorough study of the security of the new MST_3 scheme, and prove lower bounds for the work effort required to determine an equivalent private key. We develop a chosen plaintext attack which enables to rule out the use of non-fused transversal logarithmic signatures. In addition, we show that fused transversal logarithmic signatures withstand this attack and thus to our knowledge they could be used in MST_3.We discuss implementation issues and provide data of performance from an experimental implementation with various parameters. These can be used as examples for the suitable choice of parameters for set-up of MST_3 in practice.

It is an interesting problem to investigate further classes of factorizable logarithmic signatures which may be used for MST_3, for example, the promising class of nonperiodic logarithmic signatures [Sza04] and their transformations. Another challenging problem regarding the realization of the scheme is the question of how to use the class of non-transversal signatures or random covers instead of the transversal ones to build a trap-door for the scheme.

Up to now the Suzuki 2-groups are the only groups for which the security of the realization of MST_3 has been studied and analysed. It is, therefore, worthwhile studying other classes of groups which are suitable for a realization of MST_3. Another possible direction for future research is the problem of constructing digital signature schemes on the basis of logarithmic signatures and random covers.

Appart from the main research objective, we use the insight about random covers gained during the investigation of MST_3 and use their induced mappings as random functions to build pseudorandom number generators. We introduce a new, extremely promising method for constructing PRNGs based on random covers for large finite groups, called MSTg. In particular, a realization of MSTg with the class of elementary abelian 2-groups allows a simple presentation and a highly efficient implementation. An extensive sequence of tests for randomness using the NIST Statistical Test Suite and Diehard Battery of Tests show extremely strong properties for the new generators. The results show that any randomly generated MSTg passes the NIST and Diehard tests under chosen stringent criterion. We discuss the security of the generators with respect to cryptographic applications. Although a more rigorous and mathematical proof, that however appears to be difficult, is still not available yet, we have given argumentation proving that the generators are suitable for cryptographic applications. We include data of performance of several MSTg's and propose a method of using them in practice. We think it is worth pursuing further investigation of these very simple, efficient and strong generators.

As this approach is recently introduced, many interesting open questions related to MSTg's need further investigation. For example, the question of reducing cover sizes by decreasing their block lengths, or the idea of using output (and/or intermediate) sequences of the generator as a feedback to dynamically update elements in covers. We will treat these questions in an upcomming research.

Appendix A

An Example

Here we present a toy example of the set-up for MST_3 with Suzuki 2-group $\mathsf{A}(12, \theta)$, with fused transversal β of type $(16, 16, 16)$ and Frobenius automorphism θ.

We start with the generating a factorizable logarithmic signature β for $\mathcal{Z}$ using Algorithm 9 from Chapter 5. Note that in practice, the size of $\mathcal{Z}$ and length of blocks of ε and β are much larger (see e.g. Chapter 6). As the $.\mathsf{a}$ part of the elements of $\mathcal{Z}$ is equal to zero (zero vector), we show only $.\mathsf{b}$ parts of the elements of the transformed signatures. For a signature $\varepsilon := (e_{i,j})$ of $\mathcal{Z}$ we define $\varepsilon_{.\mathsf{b}} := (e_{(i,j).\mathsf{b}})$.

Let ρ be a randomly chosen matrix in $\mathsf{GL}(12, 2)$

$$\rho = \begin{pmatrix} 011011001011 \\ 111100001010 \\ 100011010100 \\ 010010000101 \\ 001101101110 \\ 110001000000 \\ 111001010101 \\ 001111101000 \\ 111011000101 \\ 100011111011 \\ 011011110100 \\ 100011100101 \end{pmatrix}.$$

Create the canonical logarithmic signature $\varepsilon = [\mathsf{E}_1, \ldots, \mathsf{E}_6]$ of type $(4, 4, 4, 4, 4, 4)$. Filling ε with random bits, construct randomized canonical signature $\varepsilon^* := (e^*_{i,j})$.

Apply ρ on elements of ε^*. To make it easier for the reader to follow the movement of the bits after applying the fusion and shuffle transformations, we will apply the transformation with ρ in the last step.

	$\mathcal{E}_{.b}$			$\mathcal{E}^*_{.b}$
E_1	**00** 00 00 00 00 00		E^*_1	**00** 00 00 00 00 00
	10 00 00 00 00 00			**10** 00 00 00 00 00
	01 00 00 00 00 00			**01** 00 00 00 00 00
	11 00 00 00 00 00			**11** 00 00 00 00 00
E_2	00 **00** 00 00 00 00		E^*_2	**11 00** 00 00 00 00
	00 **10** 00 00 00 00			**01 10** 00 00 00 00
	00 **01** 00 00 00 00			**11 01** 00 00 00 00
	00 **11** 00 00 00 00			**01 11** 00 00 00 00
E_3	00 00 **00** 00 00 00		E^*_3	**11 00 00** 00 00 00
	00 00 **10** 00 00 00			**10 10 10** 00 00 00
	00 00 **01** 00 00 00			**01 00 01** 00 00 00
	00 00 **11** 00 00 00	$\longrightarrow$		**00 10 11** 00 00 00
E_4	00 00 00 **00** 00 00		E^*_4	**10 00 01 00** 00 00
	00 00 00 **10** 00 00			**11 00 11 10** 00 00
	00 00 00 **01** 00 00			**10 01 01 01** 00 00
	00 00 00 **11** 00 00			**00 01 11 11** 00 00
E_5	00 00 00 00 **00** 00		E^*_5	**01 01 01 11 00** 00
	00 00 00 00 **10** 00			**00 11 10 11 10** 00
	00 00 00 00 **01** 00			**10 01 10 10 01** 00
	00 00 00 00 **11** 00			**11 00 01 00 11** 00
E_6	00 00 00 00 00 **00**		E^*_6	**01 11 11 01 00 00**
	00 00 00 00 00 **10**			**11 11 00 00 11 10**
	00 00 00 00 00 **01**			**00 01 01 11 00 01**
	00 00 00 00 00 **11**			**00 00 00 10 01 11**

Choose a partition $\mathcal{P} = \big\{\{1,4\},\{2,6\},\{3,5\}\big\}$ and fuse blocks E^*_1 with E^*_4, E^*_2 with E^*_6, and E^*_3 with E^*_5. Let $\chi = [C_1, C_2, C_3] := (c_{i,j})$ be the resulting logarithmic signature of type $(16, 16, 16)$ obtained after this step.

As next step choose permutations $\pi_1, \pi_2, \pi_3 \in \mathfrak{S}_{16}$

$$\pi_1 = (1\ 16\ 9\ 14\ 11\ 15\ 10\ 2\ 5\ 8\ 4)(3\ 12\ 13\ 6\ 7)$$
$$\pi_2 = (1\ 3\ 13\ 5\ 15\ 12\ 7\ 14\ 2\ 8\ 6\ 10)(4\ 16\ 9\ 11)$$
$$\pi_3 = (1\ 14\ 9\ 16\ 3\ 10\ 2\ 4\ 11\ 15)(5\ 12\ 6\ 13)$$

and shuffle the elements in each block. Denote by $\chi' = [C'_1, C'_2, C'_3]$, with $C'_i := C_i^{\pi_i} = [c_{i,\pi_i(1)}, \ldots, c_{i,\pi_i(r_i)}]$, for $i = 1, 2, 3$.

Choose a permutation $\xi = (1\ 3) \in \mathfrak{S}_3$ and shuffle the blocks of χ'. Let $\chi'' = [C''_1, C''_2, C''_3] := [C'_{\xi(1)}, C'_{\xi(2)}, C'_{\xi(3)}]$.

Finally, apply matrix ρ to elements of $\chi'' := (c''_{i,j})$ to get $\beta := (b_{i,j})$ where $b_{i,j} = {c''_{i,j}}^{\rho}$.

The transformations used:

$$\mathcal{E}_{.b} \xrightarrow{\text{randomize}} \mathcal{E}^*_{.b} \xrightarrow[\text{T2}]{\text{fuse}} \chi_{.b} \xrightarrow[\text{T3}]{\pi_1,\pi_2,\pi_3} \chi'_{.b} \xrightarrow[\text{T4}]{\xi} \chi''_{.b} \xrightarrow[\text{T1}]{\rho} \beta_{.b}$$

	$\chi_{.b}$			$\chi'_{.b}$
	1000 0100 0000			**1101 1111** 0000
C_1	**1100 1110** 0000		C'_1	**0000 0100** 0000
	1001 0101 0000			**0101 1111** 0000
	0001 1111 0000			**1000 0100** 0000
	0000 0100 0000			**1001 1111** 0000
	0100 1110 0000			**0001 0101** 0000
	0001 0101 0000			**1001 0101** 0000
	1001 1111 0000			**0001 1111** 0000
	1100 0100 0000			**0000 1110** 0000
	1000 1110 0000			**1100 1110** 0000
	1101 0101 0000			**0101 0101** 0000
	0101 1111 0000			**0100 0100** 0000
	0100 0100 0000			**0100 1110** 0000
	0000 1110 0000			**1101 0101** 0000
	0101 0101 0000			**1000 1110** 0000
	1101 1111 0000			**1100 0100** 0000
	1011 1101 0000			**1101 0111 0001**
	0011 0000 1110			**0110 0010 0111**
	1101 0111 0001			**0000 1101 0000**
	1100 0010 0111			**0111 0010 0111**
C_2	**0001 1101 0000**		C'_2	**0110 0111 0001**
	1001 0000 1110			**0010 0000 1110**
	0111 0111 0001			**1000 0000 1110**
	0110 0010 0111			**1001 0000 1110**
	1010 1101 0000			**1100 0111 0001**
	0010 0000 1110	$\longrightarrow$		**1011 1101 0000**
	1100 0111 0001			**1100 0010 0111**
	1101 0010 0111			**0111 0111 0001**
	0000 1101 0000			**0001 1101 0000**
	1000 0000 1110			**0011 0000 1110**
	0110 0111 0001			**1101 0010 0111**
	0111 0010 0111			**1010 1101 0000**
	1001 0111 0000			**0001 0111 10**00
	1111 1011 1000			**0000 0100 11**00
	0101 1010 0100			**0111 1111 10**00
	0000 0100 1100			**1101 1110 01**00
	1111 1111 0000			**1000 0000 11**00
	1001 0011 1000			**0111 1011 00**00
C_3	**0011 0010 01**00		C'_3	**0011 0010 01**00
	0110 1100 1100			**0110 1100 11**00
	0001 0011 0000			**1110 1000 11**00
	0111 1111 1000			**1111 1011 10**00
	1101 1110 0100			**1011 0110 01**00
	1000 0000 1100			**1001 0011 10**00
	0111 1011 0000			**1111 1111 00**00
	0001 0111 1000			**0001 0011 00**00
	1011 0110 0100			**1001 0111 00**00
	1110 1000 1100			**0101 1010 01**00

	$\chi''_{.b}$			$\beta_{.b}$
	0001 0111 1000			1011 1011 1101
C''_1	0000 0100 1100		B_1	1010 0111 1110
	0111 1111 1000			1111 0000 1101
	1101 1110 0100			0100 1100 0100
	1000 0000 1100			0000 1111 0101
	0111 1011 0000			1101 1000 1000
	0011 0010 0100			1010 1111 1111
	0110 1100 1100			1110 1100 1110
	1110 1000 1100			0100 0100 0101
	1111 1011 1000			0101 1000 0110
	1011 0110 0100			0000 0111 0100
	1001 0011 1000			0001 0011 0110
	1111 1111 0000			0111 0000 0011
	0001 0011 0000			1001 0011 1000
	1001 0111 0000			0011 1011 0011
	0101 1010 0100			1110 0100 1111
	1101 0111 0001			0100 0101 1100
	0110 0010 0111			1111 0110 0001
	0000 1101 0000			1100 1100 0110
C''_2	0111 0010 0111		B_2	1011 1110 0100
	0110 0111 0001			1110 1100 0110
	0010 0000 1110			1000 0001 1110
	1000 0000 1110			0110 0000 0001
	1001 0000 1110			0010 1000 0100
	1100 0111 0001			0000 1101 1001
	1011 1101 0000	$\longrightarrow$		0110 0101 1100
	1100 0010 0111			0001 0111 1110
	0111 0111 0001			1010 0100 0011
	0001 1101 0000			1000 0100 0011
	0011 0000 1110			1100 1001 1011
	1101 0010 0111			0101 1111 1011
	1010 1101 0000			0010 1101 1001
	1101 1111 0000			1111 1101 0111
	0000 0100 0000			1100 0100 0000
	0101 1111 0000			1001 0001 1100
	1000 0100 0000			1010 1000 1011
	1001 1111 0000			0000 1101 1101
	0001 0101 0000			1011 0010 1101
C''_3	1001 0101 0000		B_3	1101 1110 0110
	0001 1111 0000			0110 0001 0110
	0000 1110 0000			0001 0111 1011
	1100 1110 0000			1000 1011 1010
	0101 0101 0000			0100 0010 0111
	0100 0100 0000			0011 0100 1010
	0100 1110 0000			1110 0111 0001
	1101 0101 0000			0010 1110 1100
	1000 1110 0000			0111 1011 0000
	1100 0100 0000			0101 1000 0001

Thus we obtain a factorizable logarithmic signature $\beta = [B_1, B_2, B_3] := (b_{i,j})$ for $\mathcal{Z}$

	0000 0000 0000	,	1011 1011 1101
B_1	0000 0000 0000	,	1010 0111 1110
	0000 0000 0000	,	1111 0000 1101
	0000 0000 0000	,	0100 1100 0100
	0000 0000 0000	,	0000 1111 0101
	0000 0000 0000	,	1101 1000 1000
	0000 0000 0000	,	1010 1111 1111
	0000 0000 0000	,	1110 1100 1110
	0000 0000 0000	,	0100 0100 0101
	0000 0000 0000	,	0101 1000 0110
	0000 0000 0000	,	0000 0111 0100
	0000 0000 0000	,	0001 0011 0110
	0000 0000 0000	,	0111 0000 0011
	0000 0000 0000	,	1001 0011 1000
	0000 0000 0000	,	0011 1011 0011
	0000 0000 0000	,	1110 0100 1111
	0000 0000 0000	,	0100 0101 1100
	0000 0000 0000	,	1111 0110 0001
	0000 0000 0000	,	1100 1100 0110
B_2	0000 0000 0000	,	1011 1110 0100
	0000 0000 0000	,	1110 1100 0110
	0000 0000 0000	,	1000 0001 1110
	0000 0000 0000	,	0110 0000 0001
	0000 0000 0000	,	0010 1000 0100
	0000 0000 0000	,	0000 1101 1001
	0000 0000 0000	,	0110 0101 1100
	0000 0000 0000	,	0001 0111 1110
	0000 0000 0000	,	1010 0100 0011
	0000 0000 0000	,	1000 0100 0011
	0000 0000 0000	,	1100 1001 1011
	0000 0000 0000	,	0101 1111 1011
	0000 0000 0000	,	0010 1101 1001
	0000 0000 0000	,	1111 1101 0111
	0000 0000 0000	,	1100 0100 0000
	0000 0000 0000	,	1001 0001 1100
	0000 0000 0000	,	1010 1000 1011
	0000 0000 0000	,	0000 1101 1101
	0000 0000 0000	,	1011 0010 1101
B_3	0000 0000 0000	,	1101 1110 0110
	0000 0000 0000	,	0110 0001 0110
	0000 0000 0000	,	0001 0111 1011
	0000 0000 0000	,	1000 1011 1010
	0000 0000 0000	,	0100 0010 0111
	0000 0000 0000	,	0011 0100 1010
	0000 0000 0000	,	1110 0111 0001
	0000 0000 0000	,	0010 1110 1100
	0000 0000 0000	,	0111 1011 0000
	0000 0000 0000	,	0101 1000 0001

Generate a random cover $\alpha = [A_1, A_2, A_3] := (a_{i,j})$ of the same type as β for $\mathcal{G}$

	1000 1100 0010	,	1010 1010 0011
A_1	0100 1110 0011	,	1011 0100 1111
	0011 1000 1110	,	1110 1110 0010
	1101 1001 1101	,	1000 0111 0101
	0101 0010 1001	,	0011 0101 0000
	0101 1011 1010	,	1110 1010 0000
	0010 1100 0111	,	0000 0011 0110
	1101 1010 1011	,	1011 0011 1110
	1110 0100 0000	,	0101 0101 0101
	1010 1011 1100	,	1001 1100 0110
	1000 0011 0001	,	0010 1001 1010
	0100 1100 0011	,	0100 0010 0001
	0100 1100 1101	,	1110 1000 1100
	1000 0000 0011	,	1000 1111 1000
	0110 0101 0110	,	0010 0100 1111
	1111 0101 1011	,	0100 0010 0010
	0100 1011 1010	,	0011 1000 0111
	1101 0001 0000	,	1001 0110 1110
	1001 0100 1001	,	0101 0101 1010
A_2	0110 1101 1101	,	1001 1010 1010
	0001 1110 0001	,	0110 1010 1100
	1110 1100 0011	,	0011 0010 1111
	0111 1001 0110	,	1000 0110 1011
	0110 1110 0001	,	1111 0011 1110
	1101 0011 0010	,	1011 0111 0110
	1100 1011 1111	,	0101 0010 0111
	1000 0010 0011	,	0001 0111 1111
	1100 1001 1110	,	0011 0001 0010
	0001 0000 1111	,	0110 0011 1001
	1100 0011 1011	,	0010 0001 1000
	1000 1110 0111	,	0110 1101 0001
	1000 1000 1000	,	1001 0011 1001
	1100 1011 0101	,	1011 0101 1101
	0100 0011 0000	,	0010 0110 0110
	0100 0101 0111	,	0010 0100 1000
	1111 1000 1100	,	0000 1100 0101
	0011 1101 0010	,	1100 1101 1101
	1000 0000 1010	,	1010 1011 0000
A_3	0100 1100 1110	,	0011 0110 0001
	0111 1111 0001	,	0001 1101 1100
	1110 0011 1110	,	1001 0101 1101
	0011 0100 0101	,	0001 0111 1101
	1111 0010 1111	,	0000 0000 0111
	1111 1100 1100	,	1111 0100 1010
	1001 1011 1010	,	1010 1100 0100
	0001 0011 0010	,	0100 0000 0001
	1101 0010 1011	,	0110 0111 1010
	0011 1000 0010	,	0000 0001 0001

And compute the cover $\gamma = [H_1, H_2, H_3] := (h_{i,j})$, where $h_{i,j} = t_{i-1}^{-1} \cdot a_{i,j} \cdot t_i \cdot b_{i,j} \cdot a_{i,j}^{\sigma}$

	1011 0010 1010	,	1001 1001 0010
H_1	0111 0000 1011	,	1110 0100 0001
	0000 0110 0110	,	1100 0100 1110
	1110 0111 0101	,	0010 0011 0111
	0110 1100 0001	,	0010 0101 1101
	0110 0101 0010	,	0000 1100 1100
	0001 0010 1111	,	0001 0011 1100
	1110 0100 0011	,	1000 0000 0111
	1101 1010 1000	,	1101 1000 0010
	1001 0101 0100	,	1110 1100 0001
	1011 1101 1001	,	1010 0110 0110
	0111 0010 1011	,	0010 0011 1111
	0111 0010 0101	,	1100 1110 1001
	1011 1110 1011	,	1101 0000 1000
	0101 1011 1110	,	0101 1011 1010
	1100 1011 0011	,	0110 1111 0011
	0111 0011 1001	,	1000 1110 0101
	1110 1001 0011	,	1100 1011 1110
	1010 1100 1010	,	1101 0100 0111
	0101 0101 1110	,	1110 0111 0000
H_2	0010 0110 0010	,	1011 1001 0101
	1101 0100 0000	,	1001 1110 0111
	0100 0001 0101	,	0100 0100 1000
	0101 0110 0010	,	1110 0110 0010
	1110 1011 0001	,	0100 0010 0110
	1111 0011 1100	,	1110 1110 0011
	1011 1010 0000	,	0101 1011 1101
	1111 0001 1101	,	0011 1011 0010
	0010 1000 1100	,	1111 1100 0001
	1111 1011 1000	,	0101 1010 1110
	1011 0110 0100	,	0101 1010 0110
	1011 0000 1011	,	1111 0000 1101
	1100 1100 0011	,	1111 0111 1000
	0100 0100 0110	,	1100 0010 1000
	0100 0010 0001	,	0110 1110 0011
	1111 1111 1010	,	0110 1110 1111
	0011 1010 0100	,	0111 0001 1001
	1000 0111 1100	,	1001 0101 1011
H_3	0100 1011 1000	,	1001 0011 1011
	0111 1000 0111	,	0010 1111 0010
	1110 0100 1000	,	1011 0101 0001
	0011 0011 0011	,	1100 0101 1000
	1111 0101 1001	,	0101 1101 0011
	1111 1011 1010	,	0100 1100 0010
	1001 1100 1100	,	0010 1100 0010
	0001 0100 0100	,	1111 0010 0010
	1101 0101 1101	,	1101 1010 0111
	0011 1111 0100	,	0000 0010 1101

where

$$\begin{aligned}
t_0 &= && (0000\,1101\,0011\ ,\ 1111\,1111\,0010)\\
t_1 &= && (0011\,0011\,1011\ ,\ 1110\,1111\,1101)\\
t_2 &= && (0000\,1011\,1000\ ,\ 0111\,1111\,0001)\\
t_3 &= && (0000\,1100\,1110\ ,\ 1101\,0111\,1100)
\end{aligned}$$

$$\sigma = \begin{pmatrix}
1&1&0&0&1&1&0&1&0&1&0&1\\
1&0&0&0&0&0&1&1&1&1&1&0\\
1&1&0&1&1&0&0&0&1&0&0&0\\
1&0&1&1&1&1&1&1&1&0&0&1\\
1&1&1&1&0&0&1&0&0&0&1&1\\
1&1&0&0&0&0&1&0&1&0&1&0\\
0&0&0&1&0&1&1&1&0&1&1&0\\
0&0&0&1&0&1&0&1&0&0&0&1\\
0&1&0&1&1&0&1&0&1&1&1&0\\
1&0&1&1&0&0&0&0&0&1&0&1\\
0&0&0&1&0&1&0&0&1&1&1&0\\
0&1&0&0&1&0&0&0&0&1&1&0
\end{pmatrix}$$

Keep the knowledge about generating β, i.e. $\varepsilon^*, \mathcal{P}, \pi_1, \pi_2, \pi_3, \xi, \rho$, and elements t_0, t_3 secret. Publish $[\alpha, \gamma]$. To reduce the public key size, it is sufficient to send $[\alpha, \gamma_{.b}]$ and values $t_{(i-1).a} \oplus t_{(i).a}$, for $i = 1, 2, 3$.

Encryption

Let $M = (0001\,0111\,1011)$ be a message to be send encrypted. Create element $x = (0 \ldots 0, M) \in \mathcal{Z}$. Further, choose $R \in \mathbb{Z}_{|\mathcal{Z}|}$. Let $R = 1302$, $(1302 = (0101\,0001\,0110)$ in Radix 2), i.e. $\tau^{-1}(R) = (5, 1, 6)$. Compute the ciphertext (y_1, y_2) as $y_1 = \breve{\alpha}(R) \cdot x$ and $y_2 = \breve{\gamma}(R) \cdot x$. Use the triple

$$(y_{1.a}, y_{1.b}, y_{2.b}) = (1100\,0110\,0100\ ,\ 0010\,0011\,0000\ ,\ 1011\,0100\,0111)$$

as ciphertext.

Decryption

To decrypt the secret message, first compute

$$\breve{\beta}(R) = f(y_1)^{-1} \cdot y_1^{-1} \cdot t_0 \cdot y_2 \cdot t_s^{-1} = (0 \ldots 0\ ,\ 1111\,0000\,1111)$$

and

$$z = \breve{\beta}(R)^{\rho^{-1}} = (0\dots0\ ,\ 1000\,1100\,0111)$$

then factorize z with respect to ε^*, i.e. recover the 6-tuple $(j_1, \dots, j_6)$ such that $z = e^*_{j_1} \cdots e^*_{j_6}$. This is done as follows. Divide $.\mathfrak{b}$ part of z in parts of $k_i = \log_2 r_i$ bit length. (Type of ε^* in our toy example is $(4, 4, 4, 4, 4, 4)$, therefore $k_1 = \dots = k_6 = 2$.)

$$z_{.\mathfrak{b}} = (10 \parallel 00 \parallel 11 \parallel 00 \parallel 01 \parallel 11)$$

Starting from the last part, determine the pointer j_6 as the index of element e^*_{6,j_6} with last two bits, i.e. bits 11. and 12., equal to 11.

$$e^*_{6,3} = (00\,00\,00\,10\,01\,\mathbf{11}) \quad \text{and hence} \quad j_6 = 3$$

Add $e^*_{6,3}$ to $z_{.\mathfrak{b}}$ to get

$$z'_{.\mathfrak{b}} = (10\,00\,11\,10\,00\,00)$$

Continue with the part before the last part and determine the pointer j_5 as the index of element e^*_{5,j_5} with bits 9. and 10. equal to 00.

$$e^*_{5,0} = (01\,01\,01\,11\,\mathbf{00}\,00) \quad \text{and therefore} \quad j_5 = 0$$

Add $e^*_{5,0}$ to $z'_{.\mathfrak{b}}$ to get

$$z''_{.\mathfrak{b}} = (11\,01\,10\,01\,00\,00)$$

Continue in the same way to determine $j_4 = 2$, $j_3 = 3$, $j_2 = 1$ and $j_1 = 0$. From the knowledge of partition $\mathcal{P}$ reconstruct permutation μ (see Remark 5.2.1). As $\mathcal{P} = \big\{\{1, 4\}, \{2, 6\}, \{3, 5\}\big\}$

$$\mu = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 4 & 2 & 6 & 3 & 5 \end{pmatrix}$$

Transform the 6-tuple of indices $(j_1, \dots, j_6)$ with μ into $(j_{\mu^{-1}(1)}, \dots, j_{\mu^{-1}(6)}) = (0, 2, 1, 3, 3, 0)$. Concatenate consecutive pairs of indices in Radix 2 representation

$$(00,\ 10,\ 01,\ 11,\ 11,\ 00) \longrightarrow (00\|10,\ 01\|11,\ 11\|00)$$

to get the 3-tuple $(i_1, i_2, i_3) = (2, 7, 12)$. Use permutations π_1, π_2, π_3 to transform all i_ℓ into $i'_\ell = \pi_\ell^{-1}(i_\ell)$ (note that as the indices have values starting from 0, one has to substitute symbols in permutations π_1, π_2, π_3 correspondingly). Then $(i'_1, i'_2, i'_3) = (6, 1, 5)$. Finally, use $\xi^{-1} = (1\ 3)$ to swapping the first and the last index to get $\bar{R} = (5, 1, 6)$ and $R = \tau(\bar{R}) = 1302$. Compute

$$\breve{\alpha}(R) = (1100\,0110\,0100\ ,\ 0011\,0100\,1011)$$

Recover M from

$$M = y_{1.b} \oplus \breve{\alpha}(R)_{.b} = (0001\,0111\,1011)$$

Bibliography

[Fel57] W. Feller, *An Introduction to Probability, Theory and Its Applications*, John Wiley & Sons, Volume 1, (1957).

[Hig63] G. Higman, Suzuki 2-groups, *Illinois J. Math.,* **7** (1963), 79–96.

[Hup67] B. Huppert, *Endliche Gruppen I,* Springer-Verlag Berlin Heidelberg, New York (1967).

[Ber70] E. Berlekamp, Factoring polynomials over large finite fields, *Mathematics of Computation,* **24**(111) (1970), 713–735.

[Sim70] C. C. Sims, Computational methods in the study of permutation groups, in John Leech, ed., *Computational Problems in Abstract Algebra,* Proceedings of a conference held at Oxford University in 1967 (Oxford: Pergamon, 1970), 169–183.

[Sim71] C. C. Sims, Computation with permutation groups, in S. R. Petrick, ed., *Proc. Symposium on Symbolic and Algebraic Manipulation,* Los Angeles, California (New York: ACM, 1971), 23–28.

[DH76] W. Diffie and M. E. Hellman, New Directions in Cryptography, *IEEE Trans. on Inform. Theory,* **IT-22**(6) (1976), 644–654.

[Sim78] C. C. Sims, Some group-theoretic algorithms, *Topics in Algebra,* M. F. Newman, editor, Springer-Verlag Lecture notes in Math, **679** (1978), 108–124.

[FHL80] M. Furst, J. E. Hopcroft and E. Luks, Polynomial-time algorithms for permutation groups, in *Proceedings of the 21'st IEEE Symposium on Foundations of Computation of Computer Science* (1980), 36–41.

[HB82] B. Huppert and N. Blackburn, *Finite Groups II,* Springer-Verlag Berlin Heidelberg, New York (1982).

[MOS84] S. S. Magliveras, B. A. Oberg and A. J. Surkan, A New Random Number Generator from Permutation Groups, In *Rend. del Sem. Matemat. e Fis. di Milano,* **LIV** (1984), 203–223.

[EG85] T. ElGamal, A public key cryptosystem and a signature scheme based on discrete logarithms, *IEEE Transactions on Information Theory,* **31** (1985), 469–472.

[WM85] N. R. Wagner and M. R. Magyarik, A public-key cryptosystem based on the word problem, *Proc. Advances in Cryptology – CRYPTO 1984, LNCS 196,* Springer-Verlag (1985), 19–36.

[BBS86] L. Blum, M. Blum and M. Shub, A simple unpredictable pseudo-random number generator, *SIAM Journal on Computing,* **15** (1986), 364–383.

[Mag86] S. S. Magliveras, A cryptosystem from logarithmic signatures of finite groups, in *Proceedings of the 29'th Midwest Symposium on Circuits and Systems,* Elsevier Publ. Co. (1986), 972–975.

[Mem89] N. D. Memon, *On logarithmic signatures and applications,* M.S. thesis, University of Nebraska USA (1989).

[MM89a] S. S. Magliveras and N. D. Memon, Properties of Cryptosystem PGM, *Advances in Cryptology,* Lecture Notes in Comp. Sc., Springer-Verlag, **435** (1989), 447–460.

[MM89b] S. S. Magliveras and N. D. Memon, Random Permutations from Logarithmic Signatures, *Computing in the 90's, First Great Lakes Comp. Sc. Conf., Lecture Notes in Computer Science,* Springer-Verlag, **507** (1989), 91–97.

[Sho90] V. Shoup, On the deterministic complexity of factoring polynomials over finite fields, *Information Processing Letters,* **33** (1990), 261–267.

[MM92] S. S. Magliveras and N. D. Memon, Algebraic properties of cryptosystem PGM, *J. of Cryptology,* **5** (1992), 167–183.

[Riv92] R. L. Rivest, *The RC4 Encryption Algorithm,* RAS Data Security, Inc., (1992), unpublished.

[Mit96] M. D. Mitzenmacher, *The Power of Two Choices in Randomized Load Balancing,* Ph.D. thesis, Computer Science Department, University of California at Berkeley, (1996).

[MOV97] A. Menezes, P. van Oorschot and S. Vanstone, *Handbook of Applied Cryptography,* CRC Press, (1997).

[Sho97] P. Shor, Polynomial time algorithms for prime factorization and discrete logarithms on quantum computers, *SIAM Journal on Computing,* **26**(5) (1997), 1484–1509.

[Knu98] D. E. Knuth, *The Art of Computer Programming, Volume 2: Seminumerical Algorithms,* Addison-Wesley, 3$^{\mathrm{rd}}$ Edition, Reading, Massachusetts (1998).

[MN98] M. Matsumoto and T. Nishimura, Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator, *ACM Transactions on Modeling and Computer Simulation,*, **8**(1) (1998), 3–30.

[Cus00] C. A. Cusack, *Group Factorizations in Cryptography,* Ph.D. thesis, University of Nebraska USA (2000).

[Sza04] S. Szabó, *Topics in Factorization of Abelian Groups,* Birkhäuser Verlag, Basel-Boston-Berlin (2004).

[MST02] S. S. Magliveras, Tran van Trung and D. R. Stinson, New approaches to designing public key cryptosystems using one-way functions and trap-doors in finite groups, *J. of Cryptology,* **15** (2002), 285–297.

[Mag02] S. S. Magliveras, Secret- and Public-key Cryptosystems form Group Factorizations, *Tatra Mt. Math. Publ.,* **25** (2002), 11–22.

[Ngu02] P. Nguyen, Editor, *New Trends in Cryptology,* European project "STORK – Strategic Roadmap for Crypto" – IST-2002-38273. (http://www.di.ens.fr/~pnguyen/pub.html#Ng03)

[VS02] M. I. G. Vasco and R. Steinwandt, Obstacles in two public key cryptosystems based on group factorizations, *Tatra Mt. Math. Publ.,* **25** (2002), 23–37.

[Wag02] D. Wagner, A Generalized Birthday Problem, *Proc. Advances in Cryptology – CRYPTO 2002 LNCS 2442,* Springer-Verlag (2002), 288–303.

[VRS03] M. I. G. Vasco, M. Rötteler and R. Steinwandt, On Minimal Length Factorizations of Finite Groups, *Experimental Mathematics,* **12** (2003), 1–12.

[VMS04] M. I. G. Vasco, C. Martínez and R. Steinwandt, Towards a Uniform Description of Several Group Based Cryptographic Primitives, *Designs, Codes and Cryptography,* **33** (2004), 215–226.

[LT05] W. Lempken, and Tran van Trung, On minimal logarithmic signatures of finite groups, *Experimental Mathematics,* **14** (2005), 257–269.

[BSVM05] J. M. Bohli, R. Steinwandt, M. I. G. Vasco and C. Martínez. Weak keys in MST_1, *Designs, Codes and Cryptography,* **37** (2005), 509–524.

[CV06] A. Caranti and F. Dala Volta, The Round Functions of Cryptosystem PGM Generate the Symmetric Group, *Designs, Codes and Cryptography,* **38** (2006), 147–155.

[ST07] P. Svaba and Tran van Trung, On generation of random covers for finite groups, *Tatra Mt. Math. Publ.,* **37** (2007), 105–112.

[BJ08] Y. Berkovich and Z. Janko, *Groups of Prime Power Order,* Walter de Gruyter, Volume 2, Berlin, New York (2008).

[MSTZ08] S. S. Magliveras, P. Svaba, Tran van Trung and P. Zajac, On the security of a realization of cryptosystem MST_3, *Tatra Mt. Math. Publ.,* **41** (2008), 1–13.

[BCM09] S. R. Blackburn, C. Cid and C. Mullan, Cryptanalysis of the MST_3 Public Key Cryptosystem, *J. Math. Crypt.,* **3** (2009), 321–338.

[LMTW09] W. Lempken, S. S. Magliveras, Tran van Trung and W. Wei, A public key cryptosystem based on non-abelian finite groups, *J. of Cryptology,* **22** (2009), 62–74.

[VPD10] M. I. G. Vasco, A. L. P. del Pozo and P. T. Duarte, A note on the security of MST_3, *Designs, Codes and Cryptography,* **55** (2010), 189–200.

[ST10] P. Svaba and Tran van Trung, Public key cryptosystem MST_3: cryptanalysis and realization, *J. Math. Cryptol.,* **4** (2010), 271–315.

[MST11] P. Marquardt, P. Svaba and Tran van Trung, Pseudorandom number generators based on random covers for finite groups, *Design, Codes and Cryptography*, (in press). DOI:10.1007/s10623-011-9485-1

[Diehard] G. Marsaglia, DIEHARD: a battery of test of randomness, (1995). (http://stat.fsu.edu/~geo/diehard.html)

[Gap] The GAP Group, GAP – Groups, Algorithms, and Programming, Version 4.4.12, (2008). (http://www.gap-system.org)

[Nist] A. Rukhin, et. al. *Statistical test suite for random and pseudorandom number generators for cryptographic applications,* NIST Special Publication 800-22, Revised April 2010, National Institute of Standards and Technology, (2010). (http://csrc.nist.gov/rng)

[Ntl] V. Shoup, NTL: A Library for doing Number Theory, Version 5.5.2, (2009). (http://www.shoup.net/ntl)