Secret Sharing Schemes and Noncommutative Polynomial Interpolation

Daniel Johnson

A Thesis submitted to the Faculty of Graduate Studies of

The University of Manitoba

in partial fulfillment of the requirements of the degree of

MASTER OF SCIENCE

Department of Mathematics

University of Manitoba

Winnipeg

Abstract

Secret sharing schemes and polynomial interpolation - their link is by now historic. This thesis uses and develops noncommutative models of interpolation and subsequently extends the original Shamir scheme of 1979 [**32**] in several instances.

Using first left-oriented quaternion polynomials, also present are two novel schemes which use free quaternion polynomials.

The schemes exhibited, though making no use of the numerous advancements of secret sharing schemes, may be themselves conceivably advanced in much the same way that the Shamir scheme has. Thus the groundwork for schemes with noncommutative polynomials is founded.

Acknowledgements

Thanks go firstly to Dr. Yang Zhang, my supervisor, an eminently charitable fellow who seems fully incapable of being unreasonable.

Thanks secondly to those who've funded my studies, including Dr. Zhang and the Faculty of Graduate Studies at the University of Manitoba.

Lastly I'd like to give a respectful nod to those mathematicians whose work I've made use of - most particularly Vladimir Bolotnikov and T.Y. Lam - by whose help my understanding of matters has been considerably fast-forwarded.

Contents

Chapter 1

Introduction

The concept of a secret sharing scheme, however clear, is without strict mathematical formalism. A secret sharing scheme is a means of mathematically separating a piece of confidential information among a group of participants such that only valid subsets may, in unison, reconstruct this secret information in its entirety.

Almost all schemes involve one privileged individual, the dealer, who stands outside the bounds of confidentiality and knows all information to begin with, the secret included. The dealer is responsible for generating and then distributing to each participant their own derived fragment of the secret, known as a share.

The exact form according to which the secret is rebuilt varies between schemes.

But that being said, there is a common preliminary specification which will be our focus: the (n, t)-threshold scheme. Having a total number of participants t, it allows any subset of size equal to or greater than the threshold, n, to pool their shares and reconstruct the secret. Threshold schemes exemplify a type of access structure - a system of determining which subsets may reconstruct the secret.

Threshold schemes were the first developed, with both Shamir [**32**] and Blakley [**6**] in 1979. The former uses Lagrange interpolation to reconstruct a secret polynomial of degree $(n - 1)$ from n points. The latter uses the intersection of n hyperplanes of dimension $(n - 1)$ in n-dimensional space to discover their unique, hidden common point.

The applications of secret sharing schemes are primarily within the sphere of cryptography. One is secure key management, wherein a cryptographic key is separated as a secret among many servers to reduce the vulnerability of any individual to attack, as well as providing backup in case any servers are compromised [**17**]. Such is useful for both military and large-scale finance operations. Medical as well as military purposes also exist for confidential image sharing [**5**].

Another application is secure multi-party computation, wherein a group wishes to compute a common result without each participant revealing the full extent of their own input information [**21**], [**14**], [**13**]. For instance, in

a particular model of multi-party computation, each participant shares their own input via a scheme to the other participants, who then all compute using their secret shares, which then may be recombined into a complete computed value via helpful algebraic properties.

From this example we see secret sharing schemes having found themselves among the list of essential cryptographic primitives - like cryptosystems, hash functions, digital signatures - those basic blocks used in the construction of other, more complex cryptographic protocols. Secret sharing has likewise received a great deal of attention, with hundreds of results and schemes being published since its beginnings [**33**].

Understandably, for while the schemes of 1979 may have initiated and allowed certain protocols, they nevertheless had their shortcomings. Among advancements are the possibility of sharing multiple secrets at once [**34**],[**30**], shares that are reusable [**11**], [**22**], and different access structures [**4**], to name but a few.

Of the many ways of constructing secret sharing schemes - like using vector spaces and matrices, the Chinese Remainder Theorem, or systematic block codes [**34**] - perhaps the most common means is through polynomial interpolation. Indeed, the original Shamir scheme set something of a precedent.

But polynomial interpolation is itself a worthy topic of interest, having a fairly long-standing history - its manifest inception being the work of Edward

Waring in 1779 [**37**].

And so it is that with interpolation we rebuild a polynomial based only on something incomplete: finitely many of its points or values in some precise relation to it. We rely upon the forceful context of the theorems and axioms these polynomials exist within to ensure the rest. The essence of interpolation is thus in determining a whole from its parts.

Given this broad appeal, the applications of interpolation have become far too numerous to list exhaustively. Apart from secret sharing schemes, we've computational uses, like Karatsuba's algorithm which allows the fast multiplication of two natural numbers [**24**]. Yet more uses abound in engineering, e.g., in determining the performance of a diesel engine [**2**].

The advancements of polynomial interpolation are likewise numerous. As far as the present thesis is concerned, however, those relevant are where interpolation takes place in more general circumstances than polynomials over fields, finite or otherwise.

Given our aims, it's first necessary to fully articulate the 1979 Shamir scheme.

1.1 The Shamir Scheme

As in [**32**] we see the scheme has two phases, though outlined perhaps vaguely. We presently formalize it.

1.1.1 Setup

For this (n, t)-scheme, where $2 \leq n \leq t$, let $U_1, ..., U_t$ be the participants, and let $S \in \mathbb{N}$ be the secret information. (Where $\mathbb{N}$ denotes the set of natural numbers, including 0.)

The scheme actually works with polynomials over the finite field $\mathbb{F}_p$, where p is some prime larger than both n and S.

The dealer defines an $(n-1)$ degree polynomial $f(x) \in \mathbb{F}_p[x]$, with coefficients chosen randomly and uniformly, except that the constant term is taken to be S. Thus, $f(x) = a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$, and $a_0 = S$.

The dealer then evaluates $y_r = f(r)$ for $r = 1, ..., t$, and privately sends (r, y_r) to participant U_r, for every r.

1.1.2 Reconstruction

Supposing that for some $\{x_1, ..., x_n\} \subseteq \{1, ..., t\}$, that $U_{x_1}, ..., U_{x_n}$ wish to reconstruct S, they then pool their shares, giving each U_{x_i}, $1 \leq i \leq n$, the complete sequence $(x_1, y_{x_1}), ..., (x_n, y_{x_n})$ from which he obtains $f(x)$ by the well-known Lagrange interpolation model:

$$f(x) = \sum_{j=1}^{n} y_{x_j} \prod_{\substack{i=1 \\ i \neq j}}^{n} \frac{(x - x_i)}{(x_j - x_i)}.$$

Lastly, U_{x_i} evaluates $f(0) = S$.

1.1.3 Lagrange Interpolation Proof

We must, of course, demonstrate that the Lagrange interpolation formula actually works. We will do so for any field $\mathbb{F}$, and not merely $\mathbb{F}_p$; the notation will be modified slightly to deal with the more general context.

Lemma 1.1.1 For $f(x) \in \mathbb{F}[x]$ of degree $(n-1)$ such that $f(x_i) = y_i \in \mathbb{F}$, with the x_i all distinct $(1 \leq i \leq t)$, we have that $f(x) = l_\Lambda(x)$ where $\Lambda = \{i_1, ..., i_n\} \subseteq \{1, ..., t\}$ with $|\Lambda| = n$, and

$$l_\Lambda(x) = \sum_{j=1}^{n} y_{i_j} \prod_{\substack{k=1 \\ k \neq j}}^{n} \frac{(x - x_{i_k})}{(x_{i_j} - x_{i_k})}$$

Proof: Evidently, $l_\Lambda(x_{i_r}) = y_{i_r}$ for $1 \leq r \leq n$ as

$$\begin{aligned} l_\Lambda(x_{i_r}) &= \sum_{j=1}^{n} y_{i_j} \prod_{\substack{k=1 \\ k \neq j}}^{n} \frac{(x_{i_r} - x_{i_k})}{(x_{i_j} - x_{i_k})} \\ &= y_{i_r} \prod_{\substack{k=1 \\ k \neq r}}^{n} \frac{(x_{i_r} - x_{i_k})}{(x_{i_r} - x_{i_k})} \\ &= y_{i_r} \cdot 1 \\ &= y_{i_r}. \end{aligned}$$

Now, bound to each coefficient y_{i_j} of $l_\Lambda(x)$ is a product of linear polynomials.

As each such linear product contains $(n-1)$ factors it is of degree $(n-1)$; and multiplying such factors by the coefficients y_{i_j} then adding them all together to complete $l_\Lambda(x)$ cannot increase the degree. Thus $l_\Lambda(x)$ is of maximum degree $(n-1)$.

Hence, for $1 \leq k \leq n$,

$$f(x_{i_k}) = y_{i_k} = l_\Lambda(x_{i_k}) \iff f(x_{i_k}) - l_\Lambda(x_{i_k}) = 0 \iff (f - l_\Lambda)(x_{i_k}) = 0.$$

But this means that $(f - l_\Lambda)(x)$, a polynomial of degree no greater than $(n-1)$, has n distinct zeros, which could imply it was composed of n distinct linear factors. Thus $(f - l_\Lambda)(x)$ could be of degree n, in fact; a contradiction, unless $(f - l_\Lambda)(x) \equiv 0$. That is, $(f - l_\Lambda)(x) = 0$, for every $x \in \mathbb{F}$; i.e., $f(x) = l_\Lambda(x)$. □

1.2 Computational Complexity

Abstractly, it's helpful to use a strictly mathematical way of measuring the computational costs of an algorithm or process, apart from the hardware and software specificities used to implement them. In this thesis complexity will be measured in the number of operations necessary to perform a given algorithm, considered as a function of a single input, $f : \mathbb{N} \to \mathbb{N}$.

With this in mind, using [**28**] we define the following.

Definition 1.2.1 For $f, g : \mathbb{N} \to \mathbb{N}$ we have $f(n) = O(g(n))$ if there exist $k, c \in \mathbb{N}$ such that for every $n \geq k$, we have $f(n) \leq cg(n)$.

This notation allows us to state the complexity in asymptotic terms, as to avoid being lost in the superfluous detail which would invariably arise from multi-step processes.

Moreover, as computational costs become significant, those smaller, suppressed terms will contribute less and less to the actual outcome; and these would be the situations in which one was actually concerned with efficiency to begin with.

In this thesis, the function f is always left implicit and we speak of an algorithm being of $O(g(n))$ complexity without further reference.

Remark 1.2.2 For $f(n)$ and $g(n)$ of complexities $O(g(n))$ and $O(h(n))$, respectively, we have that the complexity of $(f + g)(n)$ is $O(h(n))$.

Proof: By definition we have $f = O(g)$ if there exist $c, k \in \mathbb{N}$ such that for every $n \geq k$, $f(n) \leq cg(n)$. Similarly, having $g = O(h)$ means there exist c' and k' such that for every $n \geq k'$, $g(n) \leq c'h(n)$. Hence, taking $k'' = \max\{k, k'\}$ we have, for $n \geq k''$, $(f + g)(n) = f(n) + g(n) \leq cg(n) + c'h(n) \leq cc'h(n) + c'h(n) \leq cc'h(n) + cc'h(n) = 2cc'h(n)$. Thus $2cc'$ and k'' satisfy the requirements to have $(f + g)(n) = O(h(n))$. □

The import of this remark is that for distinct sequential processes the sum-

mative asymptotic complexity will actually equal that of the most asymptotically complex process of the sequence.

Remark 1.2.3 For $f(n)$ and $g(n)$ of complexities $O(h_1(n))$ and $O(h_2(n))$, respectively, we have that the complexity of $(fg)(n)$ is $O((h_1h_2)(n))$.

Proof: By assumption, for $f = O(h_1)$ we have $c, k \in \mathbb{N}$ and for $g = O(h_2)$ we have $c', k' \in \mathbb{N}$ satisfying the definition. Hence, taking $c'' = cc'$ and $k'' = \max\{k, k'\}$, we have, for $n \geq k''$, $(fg)(n) = f(n)g(n) \leq ch_1(n)c'h_2(n) = c''h_1h_2(n)$, as required. □

This remark tells us that the complexity of products is the product of complexities.

1.3 Thesis Overview

This thesis is organized as follows:

In **Chapter 2** we introduce the fundamentals of left-oriented quaternion polynomials §2.1 for Lagrange §2.2 and Newton §2.3 interpolation models, as well as the basic notions of free quaternion polynomials §2.4.

In **Chapter 3** we exhibit the basics of left-oriented skew polynomials §3.1, the relevant details of the left backward shift operator for Hermite interpolation §3.2. Afterwards the actual conditions for the Hermite interpolation of

skew polynomials are given in §3.3, with a brief example of the quaternion-conjugation skew polynomial ring §3.4.

In **Chapter 4** we present the quaternion Lagrange extension of the Shamir secret sharing scheme with setup §4.1 and reconstruction §4.2 phases identified. Computational complexity analysis is broken into setup complexity §4.3.1 and reconstruction §4.3.2. Lastly, a brief security analysis is given §4.4.

In **Chapter 5** is an analogous scheme for the quaternion Newton extension of the Shamir scheme with setup §5.1 and reconstruction §5.2 phases identified. Computational complexity §5.3 and security analyses §5.4 follow.

In **Chapter 6** we present a new secret sharing scheme using free quaternion polynomials, with setup §6.1, reconstruction §6.2, complexity analysis of the setup §6.3.1 and reconstruction §6.3.2. Lastly, a security analysis §6.4.

In **Chapter 7** we present a variation of the previous secret sharing scheme using free quaternion polynomials, with setup §7.1, reconstruction §7.2, complexity analysis of the setup §7.3.1 and reconstruction §7.3.2. Lastly, a security analysis §7.4.

In **Chapter 8** we detail prospects of future work in mathematics from the major questions raised in this thesis.

Chapter 2

Quaternion Polynomials

All definitions and basic results on quaternion polynomials and interpolation have been drawn from [**8**], [**23**], and [**38**].

If one were to begin by asking why we have given the (real) quaternions our exceptional attention in this thesis, we would answer that they are perhaps the most accessible and concrete noncommutative division ring. So, one may gain the appreciation of the challenges and appeals of noncommutativity but without forfeiting the greater part of intellectual intuition.

And, for their part, the real quaternions are of a venerable lineage, being the first genuine extension of the complex numbers. Discovered and publicized in 1843 by William Rowan Hamilton [**19**] the algebra of quaternions has since found application in engineering and the physical sciences especially, e.g., in quantum mechanics [**20**]. More recent examples being signal processing [**27**],

flight simulation [**12**], and computer graphic software [**36**].

The subsequent interest in quaternion polynomials was natural, here also much progress being made since their inception, as evidenced in [**29**], [**8**], and [**9**].

Several non-equivalent definitions of quaternion polynomials do exist - as highlighted in [**39**] - from which there remain important unsolved questions. For instance, given an arbitrary quaternion polynomial (of any definition, actually) there's no extant algorithm for finding all of its roots. Knowing this, we will merely be choosing whatever variety of quaternion polynomial best suits our purpose.

2.1 Elements of Quaternion Polynomials

In order to speak cogently of polynomials over the quaternions, we must first, if not simultaneously, define and investigate the quaternions themselves.

The set of (real) quaternions $\mathbb{H}$ is a division ring consisting of elements

$$a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k} \text{ where } a, b, c, d \in \mathbb{R} \text{ and } \mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1.$$

Note that the center of $\mathbb{H}$ is $\mathbb{R}$. Also note, for $\alpha = a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k}$, the conjugate of α is $\overline{\alpha} = a - b\mathbf{i} - c\mathbf{j} - d\mathbf{k}$ and the modulus of α is defined as $\|\alpha\| = \sqrt{a^2 + b^2 + c^2 + d^2} = \sqrt{\alpha\overline{\alpha}} = \sqrt{\overline{\alpha}\alpha} \in \mathbb{R}$. We define $Re(\alpha) = a \in \mathbb{R}$.

Let $\mathbb{H}[z]$ denote the ring of polynomials in z, a commutative variable, with quaternion coefficients. Due to the noncommutativity of $\mathbb{H}$ an issue immediately arises in defining polynomial evaluation. For, even though the variable z commutes with all quaternions, if we substitute a particular quaternion for z this commutativity would not, in general, be sustained.

Hence, to ensure evaluation is well-defined we must establish a convention of arranging the variables in our formulae before substituting or otherwise involving our values. Here polynomial evaluation is always done with all given powers of the variable collected to the left-hand side of their respective terms, preceding substitution.

Thus we define left evaluation as follows: for $\alpha \in \mathbb{H}$ and $\sum_{k=0}^{n} z^k f_k = f(z) \in \mathbb{H}[z]$ as

$$f(\alpha) := \sum_{k=0}^{n} \alpha^k f_k.$$

Note first that, by the mere distributivity of $\mathbb{H}$, left evaluation splits over polynomial addition. Note secondly that we can as easily define right evaluation - but only one convention is necessary for our desired results. The next definition is of an elementary progression.

Definition 2.1.1 A quaternion α is called a *(left) zero* of $f(z) = \sum_{k=0}^{n} z^k f_k$ if

$$f(\alpha) = \sum_{k=0}^{n} \alpha^k f_k = 0.$$

The zeros (or roots) of a polynomial will play their role in providing a tangible, often concretely specific, object through which we may dissect, classify, compare, and even reconstruct polynomials.

As simple a beginning, and as similar to complex analysis, there are yet further ramifications of having a commutative variable with (perhaps) noncommutative coefficients. Namely that, in general, $f(z) = g(z)h(z) \not\Rightarrow f(\alpha) = g(\alpha)h(\alpha)$ for $\alpha \in \mathbb{H}$.

As, in the statement "$f(z) = g(z)h(z)$", while formally the left- and right-hand sides represent the same polynomial, the variables and coefficients will be collected differently on either side of the equation. For "$f(z)$" represents a single polynomial of the appropriate left-oriented form, while "$g(z)h(z)$" represents a product of *two*. But this situation follows from our convention which was taken up (nearly) of necessity.

As intimated in the introduction, there are indeed other definitions of quaternion polynomials and evaluation - one of which will be seen later in Chapters 7 and 8, where evaluation *is* mere value-for-variable substitution. However, and in fact, each of these definitions comes with its own set of restrictions and problematics [**39**]. The left-oriented polynomials as we've here taken afford us entirely tractable conditions for interpolation, both abstractly and computationally.

It is in this light that the following concepts will be important, not only in

understanding the intricacies of quaternion polynomial evaluation, but also in giving the necessary conditions for the interpolation problem itself.

Definition 2.1.2 Two quaternions α and β are called *equivalent*, or conjugate to each other, if there exists a nonzero $h \in \mathbb{H}$ such that $\alpha = h\beta h^{-1}$.

This is denoted $\alpha \sim \beta$, and is easily seen to be an equivalence relation. The equivalence class containing α is denoted $[\alpha]$.

We designate a class of special polynomials, associated directly to a particular quaternion and, as we will see, its entire conjugacy class. These polynomials will prove necessary in the interpolation process.

Definition 2.1.3 For $\alpha \in \mathbb{H} \setminus \mathbb{R}$ the *characteristic polynomial* of $[\alpha]$ is

$$X_{[\alpha]}(z) = (z-\alpha)(z-\overline{\alpha}) = (z-\overline{\alpha})(z-\alpha) = z^2 - 2z \cdot Re(\alpha) + \|\alpha\|^2 \in \mathbb{R}[z].$$

The next theorem, while demonstrating the connection between conjugate elements and their characteristic polynomial, is also of importance in giving alternate criteria for deciding when two quaternions are conjugate to begin with. No longer relegating matters to the nebulous question of existence, these criteria allow the check to be mere computation.

Theorem 2.1.4 For $\alpha, \beta \in \mathbb{H}$ the following are equivalent:

(1) $X_{[\alpha]}(\beta) = 0$

(2) $X_{[\beta]}(\alpha) = 0$

(3) $\alpha \sim \beta$

(4) $Re(\alpha) = Re(\beta)$ and $\|\alpha\| = \|\beta\|$

Proof: Extending from [**31**], pp.13, 16;

(3)⇒(1): So then, $\beta = h\alpha h^{-1}$ for some nonzero $h \in \mathbb{H}$. Which allows

$$X_{[\alpha]}(\beta) = (h\alpha h^{-1})(h\alpha h^{-1}) - 2(h\alpha h^{-1})Re(\alpha) - \|\alpha\|^2 \cdot hh^{-1}.$$

Though, as $\|\alpha\|^2$ and $Re(\alpha)$ are real, they will commute with h, h^{-1}, so

$$\begin{aligned} X_{[\alpha]}(\beta) &= h\alpha^2 h^{-1} - h2\alpha Re(\alpha)h^{-1} - h\|\alpha\|^2 h^{-1} \\ &= h(X_{[\alpha]}(\alpha))h^{-1} \\ &= 0. \end{aligned}$$

(3)⇒(2): Similar.

(1)⇒(4): Subtracting the equality $\beta^2 - 2\beta Re(\beta) + \|\beta\|^2 = 0$ from $X_{[\alpha]}(\beta) = 0$ gives $2(Re(\alpha) - Re(\beta))\beta = \|\alpha\|^2 - \|\beta\|^2$. If $Re(\alpha) \neq Re(\beta)$ then β must be real. Subtracting $\alpha^2 - 2\alpha Re(\alpha) + \|\alpha\|^2 = 0$ from $X_{[\alpha]}(\beta) = 0$ gives $\alpha = \beta$. Otherwise $Re(\alpha) = Re(\beta)$ and hence $\|\alpha\|^2 - \|\beta\|^2 = 0$; the result follows.

(2)⇒(4): Analogous.

(4)⇒(3): Let $\alpha = x_0 + x_1\mathbf{i} + x_2\mathbf{j} + x_3\mathbf{k}$ and $\beta = y_0 + y_1\mathbf{i} + y_2\mathbf{j} + y_3\mathbf{k}$.

By hypothesis, $x_0 = y_0$ and a quick verification from $\|\alpha\| = \|\beta\|$ we see that $x_1^2 + x_2^2 + x_3^2 = y_1^2 + y_2^2 + y_3^2$; call this latter statement (ϕ). Consider now the equation

$$(z_0 + z_1\mathbf{i} + z_2\mathbf{j} + z_3\mathbf{k})\beta = \alpha(z_0 + z_1\mathbf{i} + z_2\mathbf{j} + z_3\mathbf{k}); z_0, z_1, z_2, z_3 \in \mathbb{R}$$

Equating coefficients in the left- and right-hand sides of each of 1, $\mathbf{i}$, $\mathbf{j}$, $\mathbf{k}$, after multiplying through, gives the system of equations

$$\begin{bmatrix} 0 & x_1 - y_1 & x_2 - y_2 & x_3 - y_3 \\ -x_1 + y_1 & 0 & x_3 + y_3 & -x_2 - y_2 \\ -x_2 + y_2 & -x_3 - y_3 & 0 & x_1 + y_1 \\ -x_3 + y_3 & x_2 + y_2 & -x_1 - y_1 & 0 \end{bmatrix} \begin{bmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \end{bmatrix} = 0.$$

The claim is that the matrix on the left-hand side, call it X, is singular.

Indeed, knowing (ϕ) we see that $X \cdot (0, x_1 + y_1, x_2 + y_2, x_3 + y_3)^T = 0$. So, unless, $x_1 + y_1 = x_2 + y_2 = x_3 + y_3 = 0$ the matrix X is singular. Though, in this case also $y_i = -x_i$, for $i = 1, 2, 3$, and so the system becomes

$$\begin{bmatrix} 0 & 2x_1 & 2x_2 & 2x_3 \\ -2x_1 & 0 & 0 & 0 \\ -2x_2 & 0 & 0 & 0 \\ -2x_3 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \end{bmatrix} = 0$$

which is evidently singular in all cases of x_1, x_2, x_3 being zero or nonzero.

Hence our original equation of quaternions with these coefficients z_0, z_1, z_2, z_3 has a solution, i.e. there exists $z \in \mathbb{H}$ such that $z \neq 0$ and $z\beta = \alpha z$, i.e., $\alpha = z\beta z^{-1}$. $\square$

Now we will return to polynomials proper and derive the subsequent statement, which gives us a way to factor a quaternion polynomial so soon as we know one of its roots.

Remark 2.1.5 For $f \in \mathbb{H}[z]$, α is a left root of $f \iff f(z) = (z - \alpha)h(z)$ for some $h(z) \in \mathbb{H}[z]$.

Proof: First recognize that there is a left Euclidean algorithm for $\mathbb{H}[z]$ as $\mathbb{H}$ is a division ring. Suppose $f(\alpha) = 0$ and $\deg(f) \geq 1$, then upon division $f(z) = (z - \alpha)q(z) + r(z)$ for some $q(z) = z^n q_n + \cdots + zq_1 + q_0$. Thus,

$$
\begin{aligned}
& f(z) = (z - \alpha)(z^n q_n + \cdots + zq_1 + q_0) + r(z) \\
\implies & f(z) = z^{n+1} q_n + \cdots + z^2 q_1 + zq_0 - (z^n \alpha q_n + \cdots + z\alpha q_1 + \alpha q_0) + r(z) \\
\implies & f(\alpha) = \alpha^{n+1} q_n + \cdots + \alpha^2 q_1 + \alpha q_0 - (\alpha^n \alpha q_n + \cdots + \alpha\alpha q_1 + \alpha q_0) + r(\alpha) \\
\implies & 0 = 0 + r(\alpha) \\
\implies & r(\alpha) = 0
\end{aligned}
$$

but from this point it must be that $r(z) \equiv 0$ as $r(z)$ is actually a constant, the divisor being $(z - \alpha)$, of degree 1. Hence, $f(z) = (z - \alpha)q(z)$ and so

choose $h(z) = q(z)$.

Now assume that $f(z) = (z-\alpha)h(z)$. If we take $h(z) = z^m h_m + \cdots + zh_1 + h_0$ and left multiply by $(z-\alpha)$, then evaluate at $z = \alpha$ we will, as in the previous set of implications, get $f(\alpha) = 0$, as required. $\square$

It follows similarly that if $g(\alpha) = 0$ then $(gf)(\alpha) = 0$, for every $f \in \mathbb{H}[z]$. But if $g(\alpha) \neq 0$, then, by commutativity of z,

$$(gf)(z) = \sum_k z^k g(z) f_k = \sum_k z^k [\sum_j z^j g_j] f_k = \sum_k \sum_j z^{j+k} g_j f_k.$$

Now, evaluating gf at (α) we have

$$\begin{aligned}
(gf)(\alpha) &= \sum_k \sum_j \alpha^{j+k} g_j f_k \\
&= \sum_k \alpha^k [\sum_j \alpha^j g_j] f_k \\
&= \sum_k \alpha^k [g(\alpha)] f_k \\
&= g(\alpha) g(\alpha)^{-1} \sum_k \alpha^k g(\alpha) f_k \\
&= g(\alpha) \sum_k [g(\alpha)^{-1} \alpha g(\alpha)]^k f_k \\
&= g(\alpha) f(g(\alpha)^{-1} \alpha g(\alpha)).
\end{aligned}$$

So we see that evaluating products of two functions requires us to evaluate the rightmost function according to a *conjugated* value. From the above two

statements also comes the following:

Definition 2.1.6 For $f(z), g(z) \in \mathbb{H}[z]$, we define

$$(gf)(\alpha) := g(\alpha)f(g(\alpha)^{-1}\alpha g(\alpha)), \text{ if } g(\alpha) \neq 0;$$

$$(gf)(\alpha) := 0, \text{ if } g(\alpha) = 0.$$

We will also need a lemma. Its result is unsurprising; for conjugate elements are all inherently linked by their unique characteristic polynomial.

Lemma 2.1.7 Let $f \in \mathbb{H}[z]$, with α and β as distinct conjugates (i.e., $\beta \in [\alpha] \setminus \{\alpha\}$). The following two statements are equivalent:

(A) The quaternions α and β are left zeros of f.

(B) There exists $g(z) \in \mathbb{H}[z]$ such that $f(z) = X_{[\alpha]}(z)g(z) = g(z)X_{[\alpha]}(z)$.

Proof: ($\Leftarrow$) Trivial.

($\Rightarrow$) Suppose $f(\alpha) = f(\beta) = 0$, then since $\beta \in [\alpha]$ we have

$$\begin{aligned} & \beta^2 - \beta(\alpha + \overline{\alpha}) + \|\alpha\|^2 = X_{[\alpha]}(\beta) = 0 \\ \implies & \beta(\beta - \alpha) = \beta^2 - \beta\alpha = \beta\overline{\alpha} - \|\alpha\|^2 = (\beta - \alpha)\overline{\alpha} \\ \implies & (\beta - \alpha)^{-1}\beta(\beta - \alpha) = \overline{\alpha}. \end{aligned}$$

Since $f(\alpha) = 0$, f can be factored as in Remark 2.1.5 to evaluate $f(\beta)$.

According to Definition 2.1.6 and the above statement,

$$f(\beta) = (\beta - \alpha) \cdot h((\beta - \alpha)^{-1}\beta(\beta - \alpha)) = (\beta - \alpha) \cdot h(\overline{\alpha}).$$

Which, since $f(\beta) = 0$ and $\mathbb{H}$ is a division ring gives $h(\overline{\alpha}) = 0$. So, again, by Remark 2.1.5 the polynomial h can be factored as $h(z) = (z - \overline{\alpha})g(z)$, for some $g(z) \in \mathbb{H}[z]$. Which, given the previous factorization of f, implies

$$f(z) = (z - \alpha)h(z) = (z - \alpha)(z - \overline{\alpha})g(z) = X_{[\alpha]}(z)g(z)$$

as required. □

2.2 Quaternion Lagrange Interpolation

This sub-chapter shows how to construct a quaternion polynomial with target values $(\alpha_1, c_1), ..., (\alpha_n, c_n)$, analogous to the Lagrange interpolation method for polynomials over fields.

Definition 2.2.1 The *left minimal polynomial* (lmp) of $\Lambda \subseteq \mathbb{H}$ is the unique monic polynomial, denoted $P_{\Lambda,l}$, of least degree such that Λ is its zero set. By convention we take $P_{\emptyset,l} \equiv 1$.

The above definition is fundamental to the project of interpolation; minimal polynomials will be used as the building blocks of the interpolating polyno-

mial.

We also stated that the lmp must be unique; this stands to reason as it is monic, of least degree, and zero on Λ, whereas all polynomials zero on Λ form an ideal in $\mathbb{H}[z]$, which is a principal ideal domain (PID). It's not difficult to see that $P_{\Lambda,l}$ must be the unique generator of that formed ideal.

This uniqueness condition will ultimately play an important role in ensuring that the interpolating polynomial, as reconstructed, will be identical to the initial polynomial which generates the target values we aim for.

The next lemma and theorem tell us explicitly the form of the lmp for an arbitrary set of quaternions, taking fully into account the relations of conjugacy between the elements of this set.

Lemma 2.2.2 Let $\Lambda \subset \mathbb{H}$ be contained in a finite union of conjugacy classes. If V is a conjugacy class distinct from Λ and if $U \subset V$ contains at least two elements, then

$$P_{\Lambda \cup U,l} = X_V(z) \cdot P_{\Lambda,l}$$

Proof: A consequence of Lemma 2.1.7. □

Theorem 2.2.3 Let $\Lambda = \{\alpha_1, ..., \alpha_s\} \cup U_1 \cup \cdots \cup U_k$, where $U_1, ..., U_k$ are subsets of disjoint conjugacy classes $V_1, ..., V_k$, respectively, containing at least two elements, and $\alpha_1, ..., \alpha_s \in \mathbb{H} \setminus (V_1 \cup \cdots \cup V_k)$ are all pairwise nonconjugate

quaternions. Then

$$P_{\Lambda,l} = p_s(z) \cdot \prod_{j=1}^{k} X_{V_j}(z),$$

where $p_s(z)$ is a monic polynomial of degree s defined recursively, starting with $p_0(z) \equiv 1$ and, for $0 \leq k < s$,

$$p_{k+1}(z) = p_k(z)(z - p_k(\alpha_{k+1})^{-1}\alpha_{k+1}p_k(\alpha_{k+1})).$$

Proof: As all the α_j's belong to distinct conjugacy classes, induction shows that for each $j = 1, ..., s$ the polynomial $p_j(z)$ is the lmp of the set $\{\alpha_1, ..., \alpha_j\}$ and also that $p_j(\alpha_{j+1}) \neq 0$. Hence p_s is the lmp of $\{\alpha_1, ..., \alpha_s\}$ and repeated application of Lemma 2.2.2 yields

$$P_{\Lambda,l} = P_{\{\alpha_1,...,\alpha_s\},l}(z) \cdot \prod_{j=1}^{k} X_{V_j}(z) = p_s(z) \cdot \prod_{j=1}^{k} X_{V_j}(z)$$

as required. □

Now we approach the problem of interpolation proper. First, for quaternions $\{\alpha_1, ..., \alpha_n\} = \Lambda$, let $\Lambda_k = \Lambda \setminus \{\alpha_k\}$, where $1 \leq k \leq n$.

Theorem 2.2.4 Assume that no three of the interpolation nodes, i.e., no three α_i's of target values $(\alpha_1, c_1), ..., (\alpha_n, c_n)$, are conjugate. Then $g(z)$ will be the unique polynomial of degree less than n such that $g(\alpha_j) = c_j$, for

every j, where

$$g(z) = \sum_{k=1}^{n} P_{\Lambda_k,l}(z) \cdot P_{\Lambda_k,l}(\alpha_k)^{-1} \cdot c_k$$

Proof: The polynomial $P_{\Lambda_k,l}(z)$ vanishes on Λ_k, i.e., $P_{\Lambda_k,l}(\alpha_i) = 0$ for $i \neq k$. But also $P_{\Lambda_k,l}(\alpha_k) \neq 0$ since otherwise the set Λ_k contains at least two elements equivalent to α_k, which would contradict the assumption.

To understand this we look to the construction of $P_{\Lambda_k,l}(z)$. Its leftmost factor is some "$p_s(z)$" based on those elements which have no conjugate in Λ_k. If nevertheless $p_s(\alpha_k) = 0$ then by Definition 2.1.6 and $p_s(z)$ being a product of linear factors based on elements of Λ_k, we must have some $\alpha_j \in \Lambda_k$ such that $\alpha_k \sim \alpha_j$ and $p_s(\alpha_j) = 0$. (So we can write, by Remark 2.1.5, $p_s = (z - \alpha_j)p'(z)$, for some $p' \in \mathbb{H}[z]$.)

Though, these conjugates being distinct, certainly $(\alpha_k - \alpha_j) \neq 0$. This means $p'(\widetilde{\alpha_k}) = 0$, where $\widetilde{\alpha_k} = (\alpha_k - \alpha_j)\alpha_k(\alpha_k - \alpha_j)^{-1}$.

But then $p_s(z) = (z - \alpha_j)(z - \widetilde{\alpha_k})p''(z)$ which violates the form of $p_s(z)$, since by definition this polynomial is based on elements of Λ_k which have no conjugate in Λ_k. Unless, $\widetilde{\alpha_k} \notin \Lambda_k$, which is impossible as $p_s(z)$ is a factor of $P_{\Lambda_k,l}$, which is the *minimal* polynomial associated to Λ_k; there are no superfluous roots or factors.

If $\left(\prod_{j=1}^{m} X_{V_j}\right)(\alpha_k) = 0$ - where m is the number of distinct conjugacy classes

in Λ_k containing two conjugate elements - then we must have $\alpha_k \sim \alpha_j \in V_j$ for some $1 \leq j \leq m$. Though as the factors of this product function exist only for elements of Λ_k which have two conjugates already, having a total of three in Λ is excluded.

With $P_{\Lambda_k,l}(\alpha_k) \neq 0$ being proven, evaluation at the individual α_j verifies that $g(\alpha_j) = c_j$ for every j. And as $\deg(P_{\Lambda_k,l}) = (n-1)$ we know that $g(z)$, which is the sum of such polynomials multiplied by constants, will be at most of degree $(n-1)$.

For uniqueness, suppose we have $f \in \mathbb{H}[z]$ such that $f(\alpha_j) = c_j$, for every j. Then we have $(f-g)(\alpha_j) = f(\alpha_j) - g(\alpha_j) = c_j - c_j = 0$, for every j, so that $f - g$ has a zero set which contains Λ. Hence, by Theorem 2.2.3 we have $f - g = P_{\Lambda,l} \cdot h$ for some $h \in \mathbb{H}[z]$, i.e. $f = g + P_{\Lambda,l} \cdot h$. Though if h is not identically 0 then f, and hence g, must have at least degree n, contradicting the assumption.

□

2.3 Quaternion Newton Interpolation

Again suppose we have target values $(\alpha_1, c_1), ..., (\alpha_n, c_n)$ where all the α_i are nonconjugate and wish to devise a polynomial $g(z) \in \mathbb{H}[z]$ of degree $(n-1)$ such that $g(\alpha_i) = c_i$, for every i. The modified Newton method of

interpolation will be to compute $g(z)$ as follows:

First, adapting the notation of Theorem 2.2.4 slightly, but continuing as in Theorem 2.2.3, we define $\Lambda_j = \{\alpha_1, ..., \alpha_{j-1}\}$, with $2 \leq j \leq n-1$ and $\Lambda_1 = \emptyset$. Thus $P_{\Lambda_j,l} = p_j(z)$ where $p_j(z)$ is the monic polynomial of degree $(j-1)$ defined recursively, for $1 \leq k \leq (n-1)$, as $p_1(z) \equiv 1$ and

$$p_{k+1}(z) = p_k(z)(z - p_k(\alpha_{k+1})^{-1}\alpha_{k+1}p_k(\alpha_{k+1})).$$

Now define $g(z) = p_1(z)b_1 + p_2(z)b_2 + \cdots + p_n(z)b_n$ where the b_j's are quaternions computed recursively as $b_1 = c_1$ and, for $1 \leq i \leq n-1$,

$$b_{i+1} = p_{i+1}(\alpha_{i+1})^{-1}(c_{i+1} - b_1 - p_2(\alpha_{i+1})b_2 - \cdots - p_i(\alpha_{i+1})b_i) \in \mathbb{H}.$$

Noting that $p_{i+1}(\alpha_{i+1}) \neq 0$ as α_{i+1} is nonconjugate to all elements of Λ_i.

Lemma 2.3.1 The above construction gives $g(z)$ as required, i.e., gives us $g(\alpha_i) = c_i, 1 \leq i \leq n$, where g is the unique polynomial of degree $(n-1)$ satisfying these equalities.

Proof: By induction on j; the index of the α_j's.

Base case, $j = 1$:

Note that for $1 < i \leq (n-1)$ that $p_i(\alpha_1) = 0$, and hence

$$\begin{aligned} g(\alpha_1) &= p_1(\alpha_1)b_1 + p_2(\alpha_1)b_2 + \cdots + p_n(\alpha_1)b_n \\ &= 1 \cdot b_1 + 0 \cdot b_2 + \cdots + 0 \cdot b_n \\ &= c_1. \end{aligned}$$

The case of $(j+1)$ from j, for $1 \leq j \leq (n-1)$; as $p_k(\alpha_j) = 0$ for $k > j$:

$$\begin{aligned} g(\alpha_{j+1}) &= b_1 + p_2(\alpha_{j+1})b_2 + \cdots + p_n(\alpha_{j+1})b_n \\ &= b_1 + p_2(\alpha_{j+1})b_2 + \cdots + p_{j+1}(\alpha_{j+1})b_{j+1} \\ &= b_1 + \cdots + p_{j+1}(\alpha_{j+1})[p_{j+1}(\alpha_{j+1})^{-1}(c_{j+1} - b_1 - \cdots - p_j(\alpha_{j+1})b_j)] \\ &= c_{j+1}. \end{aligned}$$

As required. Lastly, as our interpolation nodes are all assumed nonconjugate, uniqueness holds as Theorem 2.2.4 asserts there is but one polynomial having these nodes and of degree no more than $(n-1)$, however it be expressed. □

2.4 Free Quaternion Polynomials

Let $\mathbb{H}\langle z \rangle$ denote the free algebra with strictly noncommutative indeterminate z over $\mathbb{H}$. We call elements of this set free quaternion polynomials.

Elements of $\mathbb{H}\langle z \rangle$ are finite sums of finite "words" of quaternions and the

variable z.

Certainly if one likes, one may consider $\mathbb{H}\langle z\rangle$ as a polynomial ring, with addition and multiplication mostly as expected. The exception is that for addition terms of the same "degree" in z - by which we mean terms with the same number of occurrences of z - typically can't be collected into a single term. Verily, as the noncommuting variables of respective terms may themselves be entrapped by distinct quaternion coefficients on the left-hand and right-hand sides.

Nevertheless, for its universally vexing complications, in $\mathbb{H}\langle z\rangle$ we gain a straightforward (and unambiguous) definition of polynomial evaluation: simple substitution. I.e., for $f \in \mathbb{H}\langle z\rangle$ and $\alpha \in \mathbb{H}$, define $f(\alpha)$ as the quaternion obtained by replacing each instance of z in f by α.

Comparatively little is yet known of free quaternion polynomials, with only a number of basic and partial results. As, in **[16]**, which tells us that such polynomials with a single highest degree term must have a root - but speak nothing of what polynomial factorization means if this root is even concretely determined.

However, the requisites of our secret sharing schemes won't go beyond this scanty knowledge, and so we may proceed.

Chapter 3

Skew Polynomial Rings

The following is an adapted collation from [**7**], [**18**], [**25**], and [**26**]. Modifications, syntheses, and amending absences of proofs were done by the present author.

Skew polynomials are a relatively new addition to the mathematical scene. They have found applications in, e.g., cryptanalysis [**1**] and linear codes [**10**]. More universal than the quaternion polynomials as hitherto defined, which are but one example of many.

Significant to mention is that skew polynomials are typically of a more general cast than found in this thesis, though the more specific variety, sometimes called twisted polynomials, are nevertheless of interest. They yet have explicit formulae and depend somewhat less on recursive analysis.

3.1 Elements of Skew Polynomial Rings

Skew polynomial rings exist in the context of division rings, here always denoted by $\mathbf{D}$. Now, differing but symmetric to their typical presentation, we give the following definition.

Definition 3.1.1 A *(left) skew polynomial ring* over a division ring $\mathbf{D}$ with automorphism σ is the set of elements of the form $\sum_{i=0}^{r} z^i a_i$ $(r \geq 0, a_i \in \mathbf{D})$ together with ring addition and multiplication as usual, except multiplication is now modified by the twist rule $az = z\sigma^{-1}(a)$. This ring is denoted $\mathbf{D}[z;\sigma]$.

Remark 3.1.2 $\mathbf{D}[z;\sigma]$ is a left Euclidean domain.

Proof: We demonstrate a left division algorithm for $\mathbf{D}[z;\sigma]$. That is, for every $f, g \in \mathbf{D}[z;\sigma]$ with $\deg(f) > 0$ and $\deg(g) > 0$, there exist unique $q, r \in \mathbf{D}[z;\sigma]$ with $\deg(r) < \deg(g)$ and with $f = gq + r$.

Suppose now $f(z) = z^n a_n + \cdots + za_1 + a_0$ and $g(z) = z^m b_m + \cdots + zb_1 + b_0$ with $b_m \neq 0$. If $n < m$ then $f = g0 + f$. If $n \geq m$ then

$$f(z) - g(z)z^{n-m}(\sigma^{m-n}(b_m))^{-1}a_n = z^{n-1}c_{n-1} + \cdots + c_0.$$

Induction on n gives the existence of q and r and uniqueness may be proven by assuming the existence of another q', r' such that $f = gq' + r'$. For, in this case, $0 \equiv g(q - q') + (r - r')$, but as $\deg(g)$ is strictly greater than $0, \deg(r)$

and $\deg(r')$, we must have $q = q'$ and thus $r = r'$. $\square$

Similar to the case of quaternion polynomials there are difficulties in defining an unambiguous method of polynomial evaluation, though now further complicated by the noncommutativity of the variable z with elements of $\mathbf{D}$. There are two equivalent ways of dealing with this.

Definition 3.1.3 For $a \in \mathbf{D}, f(z) \in \mathbf{D}[z;\sigma]$, if $f(z) = (z-a)q(z) + r$, for some $r \in \mathbf{D}$, the evaluation of $f(z)$ at a is $f(a) := r$.

This definition works because $\mathbf{D}[z;\sigma]$ is a left Euclidean domain, so that such division by a linear factor always produces a unique constant of $\mathbf{D}$; this evaluation map is well-defined.

There is, however, a second definition of evaluation in the case of skew polynomial rings. Define a sequence of functions $M_i : \mathbf{D} \to \mathbf{D}$ recursively as

$$M_0(a) = 1 \text{ and } M_{i+1}(a) = a \cdot \sigma^{-1}(M_i(a)).$$

How these new functions relate to the previous definition of polynomial evaluation is the content of the following theorem.

Theorem 3.1.4 For $f(z) = \Sigma_i z^i b_i \in \mathbf{D}[z;\sigma]$ and $a \in \mathbf{D}$, $f(a) = \Sigma_i M_i(a) b_i$.

Proof: First observe the special case:

For every $n \geq 0$, we have $z^n - M_n(a) \in (z-a) \cdot \mathbf{D}[z;\sigma]$.

Call the above statement (θ), which is proven by induction on n as follows. By definition of $M_i(a)$,

$$\begin{aligned} z^{n+1} - M_{n+1}(a) &= z^{n+1} - a\sigma^{-1}(M_n(a)) \\ &= z^{n+1} - a\sigma^{-1}(M_n(a)) + z\sigma^{-1}(M_n(a)) - z\sigma^{-1}(M_n(a)) \\ &= (z-a)\sigma^{-1}(M_n(a)) - M_n(a)z + z^{n+1} \\ &= (z-a)\sigma^{-1}(M_n(a)) + (z^n - M_n(a))z \in (z-a) \cdot \mathbf{D}[z;\sigma] \end{aligned}$$

Using the induction hypothesis for the final containment. Thus, with (θ),

$$f(z) - \Sigma_i M_i(a) b_i = \Sigma_i (z^i - M_i(a)) b_i \in (z-a) \cdot \mathbf{D}[z;\sigma]$$

and so $r = \Sigma_i M_i(a) b_i$ by the uniqueness of the Euclidean algorithm. $\square$

Similar to the previous quaternion case the noncommutativity of the skew polynomial ring causes issues with evaluating products of polynomials. The next definition, which actually will define an equivalence relation, is useful for this.

Definition 3.1.5 For any $a \in \mathbf{D}$ and $c \in \mathbf{D} \setminus \{0\}$ let ${}^c a := c^{-1} a \sigma^{-1}(c)$. Two elements $a, b \in \mathbf{D}$ are called (left) *σ-conjugates* if there exists $c \in \mathbf{D} \setminus \{0\}$ such that ${}^c a = b$. Two elements not σ-conjugate are called (left) *σ-distinct*.

How to evaluate products of functions is now given.

Theorem 3.1.6 Let $f(z), g(z) \in \mathbf{D}[z;\sigma]$ and $a \in \mathbf{D}$. If $f(a) = 0$ then $(fg)(a) = 0$. If $f(a) \neq 0$ then $(fg)(a) = f(a)g({}^{f(a)}a)$.

Proof: If $f(a) = 0$ then $(z - a)$ divides $f(z)$ on the left and so also divides $f(z)g(z)$ on the left, thus giving, by our "division" definition of polynomial evaluation, $(fg)(a) = 0$. Now assume $f(a) \neq 0$; for brevity let $c = f(a)$. Left dividing $f(z)$ by $(z - a)$, obtaining $f(z) = (z - a)q_1(z) + c$ and thus writing $b = {}^{c}a$, we have

$$c(z - b) = c(z - c^{-1}a\sigma^{-1}(c)) = (z - a)\sigma^{-1}(c).$$

Also writing $g(z) = (z - b)q_2(z) + g(b)$ we have

$$\begin{aligned} f(z)g(z) &= (z - a)q_1(z)g(z) + c((z - b)q_2(z) + g(b)) \\ &= (z - a)q_1(z)g(z) + (z - a)\sigma^{-1}(c)q_2(z) + cg(b) \\ &= (z - a)[q_1(z)g(z) + \sigma^{-1}(c)q_2(z)] + cg(b). \end{aligned}$$

All of which means that $cg(b) = f(a)g({}^{f(a)}a)$ is the remainder of $f(z)g(z)$ when the latter is divided on the left by $(z - a)$. From Definition 3.1.3 it follows that $(fg)(a) = f(a)g({}^{f(a)}a)$. □

The next theorem and corollary while, strictly speaking, unnecessary for our

further results, do offer insight into the theory of skew polynomials as a whole and are included for the sake of completion.

Theorem 3.1.7 A skew polynomial $f(z) \in \mathbf{D}[z;\sigma]$ of degree n has roots in at most n σ-conjugacy classes.

Proof: By induction on n. The base case of $n = 1$ gives $f(z) = (z - \alpha)$, for some $\alpha \in \mathbf{D}$, and only α itself is a root of f which belongs to a single conjugacy class. So assume $n > 1$.

Suppose that $f(z)$ has a root $\beta \in \mathbf{D}$; by the division algorithm (Remark 3.1.2) we have that $f(z) = (z - \beta)g(z)$ where $g(z)$ is of degree $(n - 1)$. Then the induction hypothesis applies and g has its roots in at most $(n-1)$ conjugacy classes.

Now supposing there exists $\gamma \in \mathbf{D}$ such that $f(\gamma) = 0$, then by the product rule of evaluation (Theorem 3.1.6) we have that $(\gamma - \beta)g({}^{\gamma-\beta}\gamma) = 0$. But this means, as $\mathbf{D}$ is a division ring, that $\gamma - \beta = 0$ or $g({}^{\gamma-\beta}\gamma) = 0$.

In the latter case this means that γ belongs to one of the $(n - 1)$ conjugacy classes associated to g. In the former case this means that $\gamma = \beta$ and so we need only add one more conjugacy class, that of β, to account for all, giving us n conjugacy classes in total. □

Corollary 3.1.8 Let $\Omega = \{\alpha_1, ..., \alpha_n\}$ be a set of points of $\mathbf{D}$ such that every α_i belongs to a different conjugacy class. A skew polynomial of min-

imal degree that vanishes on Ω has degree n; there is only one such monic polynomial.

Proof: By induction on n. For the base case, only the degree 1 monic polynomial $f_1(z) = (z - \alpha_1)$ vanishes on $\{\alpha_1\}$. Supposing now that there exists $f_i(z)$ monic that vanishes on $\{\alpha_1, ..., \alpha_i\}$, we can construct the polynomial $f_{i+1}(x)$ of degree $(i+1)$ as $f_{i+1}(z) = f_i(z)(z - {}^{f_i(\alpha_{i+1})}\alpha_{i+1})$.

Apparently, f_{i+1} is monic and vanishes on $\{\alpha_1, ..., \alpha_i\}$; it also vanishes at α_{i+1} by Theorem 3.1.6. So, we can construct f_n monic of degree n that vanishes on Ω, and by Theorem 3.1.7 we know we cannot find a polynomial of less degree that also vanishes on Ω, since all its elements are nonconjugate. $\square$

3.2 Skew Polynomial Hermite Interpolation Preliminaries

A generalization on the Lagrange method, the purpose of Hermite interpolation is to find a polynomial which not only has target values for the function itself, but for its derivatives as well. We intend on adapting it to the skew polynomial case. The problem is stated thus:

Formally speaking, for target pairs

$$\begin{gathered}(\alpha_1, \beta_1^1), (\alpha_2, \beta_2^1), ..., (\alpha_n, \beta_n^1) \\ (\alpha_1, \beta_1^2), (\alpha_2, \beta_2^2), ..., (\alpha_n, \beta_n^2) \\ \vdots \\ (\alpha_1, \beta_1^k), (\alpha_2, \beta_2^k), ..., (\alpha_n, \beta_n^k)\end{gathered}$$

all in $\mathbf{D}^2$, we desire $f \in \mathbf{D}[z;\sigma]$ such that $f^{(j-1)}(\alpha_i) = \beta_i^j$, for $1 \leq i \leq n$ and $1 \leq j \leq k$.

Here, $f^{(m)}(\alpha)$, for $m \in \mathbb{N}$ and $\alpha \in \mathbf{D}$, is the m-th derivative of $f(z)$ evaluated at α - which will be defined precisely and later this section as the theory becomes sufficiently developed to support it.

Though, indeed, before actually solving the problem of Hermite interpolation we must first bend our discourse on skew polynomials towards the intricacies of a certain tool, whose properties are critical in all subsequent developments of chapters 3 and 6.

Definition 3.2.1 Taking $f(z) = \sum_{i=0}^{n} z^i f_i \in \mathbf{D}[z, \sigma]$ and $\alpha \in \mathbf{D}$ the *left backward shift operator*, denoted L_α, is given as

$$(L_\alpha f)(z) := \sum_{k=0}^{n-1} z^k \left(\sum_{j=0}^{n-k-1} \left[\prod_{0 \leq l < j} \sigma^{-(l+n-j)}(\alpha) \right] f_{j+k+1} \right).$$

Ultimately, this operator will allow us to deconstruct skew polynomials into

reduced components whose properties we can effectively gauge and manipulate.

Note, to this end, and in advance of proof, the equation

$$f(z) = f(\alpha) + (z - \alpha)(L_\alpha f)(z) \tag{3.2.1}$$

which will be seen to hold as $(L_\alpha f)(z)$ is the unique quotient obtained from dividing $f(z)$ on the left by $(z - \alpha)$, keeping in mind Definition 3.1.3.

Applying induction with equation (3.2.1) we have the following remark; we may always write a skew polynomial in a particular form, which will be all the more convenient once we have knowledge of the polynomial's zeros.

Remark 3.2.2 Define $\rho_\alpha = (z - \alpha)$, for $\alpha \in \mathbf{D}$. Now, for $\alpha_1, ..., \alpha_n \in \mathbf{D}$,

$$f = f(\alpha_1) + \sum_{k=1}^{n-1} \rho_{\alpha_1} \cdots \rho_{\alpha_k} \cdot (L_{\alpha_k} \cdots L_{\alpha_1} f)(\alpha_{k+1}) + \rho_{\alpha_1} \cdots \rho_{\alpha_n} \cdot (L_{\alpha_n} \cdots L_{\alpha_1} f).$$

The next lemma, however simple of itself, will not only render certain future propositions provable, but also greatly simplify all other proofs involving the backward shift operator.

Lemma 3.2.3 L_α is a right linear operator.

Proof: For $f, g \in \mathbf{D}[z; \sigma]$, without loss of generality assume that f and g are of the same degree (include zero terms to compensate in the lower degree

polynomial, if necessary). Hence, letting $\beta_j = \prod_{0 \le l < j} \sigma^{-(l+n-j)}(\alpha)$, we have

$$\begin{aligned}
L_\alpha(f+g) &= \sum_{k=0}^{n-1} z^k \left(\sum_{j=0}^{n-k-1} \beta_j (f+g)_{j+k+1} \right) \\
&= \sum_{k=0}^{n-1} z^k \left(\sum_{j=0}^{n-k-1} \beta_j (f_{j+k+1} + g_{j+k+1}) \right) \\
&= \sum_{k=0}^{n-1} z^k \left(\sum_{j=0}^{n-k-1} \beta_j f_{j+k+1} + \beta_j g_{j+k+1} \right) \\
&= \sum_{k=0}^{n-1} z^k \left(\sum_{j=0}^{n-k-1} \beta_j f_{j+k+1} \right) + \sum_{k=0}^{n-1} z^k \left(\sum_{j=0}^{n-k-1} \beta_j g_{j+k+1} \right) \\
&= L_\alpha f + L_\alpha g
\end{aligned}$$

as needed. Now, for $\gamma \in \mathbf{D}$,

$$\begin{aligned}
L_\alpha(f\gamma) &= \sum_{k=0}^{n-1} z^k \left(\sum_{j=0}^{n-k-1} \beta_j (f\gamma)_{j+k+1} \right) \\
&= \sum_{k=0}^{n-1} z^k \left(\sum_{j=0}^{n-k-1} \beta_j f_{j+k+1} \gamma \right) \\
&= \left[\sum_{k=0}^{n-1} z^k \left(\sum_{j=0}^{n-k-1} \beta_j f_{j+k+1} \right) \right] \gamma \\
&= (L_\alpha f)\gamma
\end{aligned}$$

as required. □

Another result gives some insight into how the linear shift operator behaves

with products of polynomials.

Lemma 3.2.4 For $\alpha, \beta \in \mathbf{D}$ and $f \in \mathbf{D}[z;\sigma]$,

$$L_\alpha[f(z)(z-\beta)] = L_\alpha[f(z)](z-\beta) + c_\alpha$$

for some $c_\alpha \in \mathbf{D}$.

Proof: To begin with,

$$L_\alpha[f(z)(z-\beta)] = L_\alpha[f(z)z - f(z)\beta)] = L_\alpha[f(z)z] - L_\alpha[f(z)]\beta$$

with the last equality justified by the linearity of the shift operator. Though attending to $L_\alpha[f(z)z]$; without loss of generality, $f(z) = z^n f_n + \cdots + z f_1 + f_0$,

so then let $g(z) = f(z)z = z^{n+1}\sigma^{-1}(f_n) + \cdots + z^2\sigma^{-1}(f_1) + z\sigma^{-1}(f_0)$ and thus

$$\begin{aligned}
L_\alpha[f(z)z] &= L_\alpha g(z) \\
&= \sum_{k=0}^{n} z^k \left(\sum_{j=0}^{n-k} \left[\prod_{0\le l<j} \sigma^{-(l+n+1-j)}(\alpha) \right] g_{j+k+1} \right) \\
&= \sum_{k=0}^{n} z^k \left(\sum_{j=0}^{n-k} \left[\prod_{0\le l<j} \sigma^{-(l+n+1-j)}(\alpha) \right] \sigma^{-1}(f_{j+k}) \right) \\
&= \sum_{k=1}^{n} z^k \left(\sum_{j=0}^{n-k} \left[\prod_{0\le l<j} \sigma^{-(l+n+1-j)}(\alpha) \right] \sigma^{-1}(f_{j+k}) \right) + c_\alpha \\
&= \sum_{k=0}^{n-1} z^k \cdot z \left(\sum_{j=0}^{n-k-1} \left[\prod_{0\le l<j} \sigma^{-(l+n+1-j)}(\alpha) \right] \sigma^{-1}(f_{j+k+1}) \right) + c_\alpha \\
&= \sum_{k=0}^{n-1} z^k \left(\sum_{j=0}^{n-k-1} \left[\prod_{0\le l<j} \sigma^{-(l+n-j)}(\alpha) \right] f_{j+k+1} \right) z + c_\alpha \\
&= L_\alpha[f(z)]z + c_\alpha
\end{aligned}$$

where $c_\alpha = \sum_{j=0}^{n} \left[\prod_{0\le l<j} \sigma^{-(l+n+1-j)}(\alpha) \right] \sigma^{-1}(f_j)$.

Hence,

$$\begin{aligned}
L_\alpha[f(z)z] - L_\alpha[f(z)]\beta &= L_\alpha[f(z)]z + c_\alpha + L_\alpha[f(z)]\beta \\
&= L_\alpha[f(z)](z-\beta) + c_\alpha
\end{aligned}$$

as required. □

But now we play catch up, and prove that wherein we have claimed.

Lemma 3.2.5 The equation (3.2.1) holds.

Proof: By induction on $n = \deg(f)$. Now, the base case holds trivially; so assume $n \geq 1$ and that $f(z) = z^n f_n + z^{n-1} f_{n-1} + \cdots + f_0 = z^n f_n + g(z)$; where $\deg(g) = n - 1$. Without loss of generality, assume $f_n \neq 0$. Now,

$$\begin{aligned} & f = f(\alpha) + (z - \alpha)(L_\alpha f) \\ \iff & f - f(\alpha) = (z - \alpha)(L_\alpha f) \\ \iff & z^n f_n - M_n(\alpha) f_n + g - g(\alpha) = (z - \alpha)(L_\alpha(z^n f_n + g)) \\ \iff & LHS = (z - \alpha) L_\alpha(z^n) f_n + (z - \alpha)(L_\alpha g) \end{aligned}$$

though, invoking the induction hypothesis, this is equivalent to

$$\begin{aligned} & LHS = (z - \alpha) L_\alpha(z^n) f_n + g - g(\alpha) \\ \iff & z^n f_n - M_n(\alpha) f_n = (z - \alpha) L_\alpha(z^n) f_n \\ \iff & z^n - M_n(\alpha) = (z - \alpha) L_\alpha(z^n). \end{aligned}$$

To look more closely at $(z - \alpha)L_\alpha(z^n)$, by definition of L_α, and writing $z^n = h(z)$ in its full polynomial form (including zero terms), we have

$$(z - \alpha) L_\alpha(z^n) = (z - \alpha) \sum_{k=0}^{n-1} z^k \left(\sum_{j=0}^{n-k-1} \left[\prod_{0 \leq l < j} \sigma^{-(l+n-j)}(\alpha) \right] h_{j+k+1} \right).$$

However, $h_{j+k+1} = 1$ when $j + k + 1 = n$, i.e., when $j = n - k - 1$ and 0

otherwise; hence

$$
\begin{aligned}
(z-\alpha)L_\alpha(z^n) &= (z-\alpha)\sum_{k=0}^{n-1} z^k \left[\prod_{0\le l<n-k-1} \sigma^{-(l+k+1)}(\alpha)\right] \\
&= \sum_{k=0}^{n-1} z^{k+1} \left[\prod_{0\le l<n-k-1} \sigma^{-(l+k+1)}(\alpha)\right] \\
&\quad - \sum_{k=0}^{n-1} z^k \sigma^{-k}(\alpha) \left[\prod_{0\le l<n-k-1} \sigma^{-(l+k+1)}(\alpha)\right] \\
&= \sum_{k=1}^{n} z^k \left[\prod_{0\le l<n-k} \sigma^{-(l+k)}(\alpha)\right] - \sum_{k=0}^{n-1} z^k \left[\prod_{0\le l<n-k} \sigma^{-(l+k)}(\alpha)\right] \\
&= z^n - z^0 \left[\prod_{0\le l<n-0} \sigma^{-(l+0)}(\alpha)\right] \\
&= z^n - M_n(\alpha)
\end{aligned}
$$

as required. □

Now that we have the L_α operators which are proven to behave as desired we define divided differences - a compact way of referring to the action of multiple shift operators on and subsequent evaluation of a given polynomial. This object, the divided difference, is used in conceptually relating the shift operator to the evaluation of skew polynomial derivatives.

Definition 3.2.6 The *left divided difference* of the skew polynomial f based

on a collection $(\alpha_1, ..., \alpha_k)$ is $[\alpha_1; f] := f(\alpha_1)$ for $k = 1$, and for $k \geq 2$

$$[\alpha_1, ..., \alpha_k; f] := (L_{\alpha_{k-1}} \cdots L_{\alpha_1} f)(\alpha_k).$$

The automorphism σ is tacitly involved in the divided difference formula. Indeed, for σ is used in the definition of polynomial evaluation and the linear shift operator.

We have another remark and two lemmas which describe the particularities of L_0; this operator essentially knocks down the degree of a polynomial by one, leaving all coefficients of nonconstant terms otherwise intact.

The involvement of L_0 is not arbitrary as it may seem; it will occur naturally in relating and moreover computing the entries of the confluent σ-Vandermonde matrix.

Remark 3.2.7 For $f = z^n f_n + \cdots + z f_1 + f_0 \in \mathbf{D}[z; \sigma]$, $n \geq 1$,

$$L_0 f = z^{n-1} f_n + \cdots + z f_2 + f_1.$$

Proof: By Lemma 3.2.5, we have $f = f(0) + z \cdot L_0 f \iff f - f(0) = z \cdot L_0 f$. The result follows by dividing on the left by z. □

Lemma 3.2.8 $L_0 L_\alpha = L_\alpha L_0$, for every $\alpha \in \mathbf{D}$.

Proof: For f of degree n, using Remark 3.2.7, the definition of L_α, and

writing $\beta_j = \prod_{0 \leq l < j} \sigma^{-(l+n-j)}(\alpha)$,

$$\begin{aligned} L_\alpha L_0 f &= L_\alpha \left[\sum_{k=0}^{n-1} z^k f_{k+1} \right] \\ &= \sum_{k=0}^{n-2} z^k \left(\sum_{j=0}^{n-k-2} \beta_j f_{j+k+2} \right) \\ &= L_0 \left[\sum_{k=0}^{n-1} z^k \left(\sum_{j=0}^{n-k-1} \beta_j f_{j+k+1} \right) \right] \\ &= L_0 L_\alpha f \end{aligned}$$

as required. □

Lemma 3.2.9 For $\alpha \in \mathbf{D}$ and $n \geq 2$,

$$(L_\alpha - L_0)(z^n) = (L_\alpha L_0 z^n)\sigma^{-(n-1)}(\alpha).$$

Proof: First, using the implicit linear splitting of $L_\alpha - L_0$,

$$\begin{aligned}
(L_\alpha - L_0)(z^n) &= L_\alpha(z^n) - L_0(z^n) \\
&= \left(\sum_{k=0}^{n-1} z^k \left[\prod_{0\le l<n-k-1} \sigma^{-(l+k+1)}(\alpha)\right]\right) - z^{n-1} \\
&= \left(\sum_{k=0}^{n-2} z^k \prod_{0\le l<n-k-1} \sigma^{-(l+k+1)}(\alpha)\right) \\
&+ \left(z^{n-1} \prod_{0\le l<0} \sigma^{-(l+n)}(\alpha)\right) - z^{n-1} \\
&= \sum_{k=0}^{n-2} z^k \left[\prod_{0\le l<n-k-1} \sigma^{-(l+k+1)}(\alpha)\right] \\
&= \left(\sum_{k=0}^{n-2} z^k \left[\prod_{0\le l<n-k-2} \sigma^{-(l+k+1)}(\alpha)\right]\right) \sigma^{-(n-1)}(\alpha) \\
&= (L_\alpha z^{n-1})\sigma^{-(n-1)}(\alpha) \\
&= (L_\alpha L_0 z^n)\sigma^{-(n-1)}(\alpha)
\end{aligned}$$

as required. □

Lemma 3.2.10 For $\alpha_1 = \cdots = \alpha_k = \alpha \in \mathbf{D}$, with $k \ge 2$ and $j \ge 1$,

$$[\alpha_1, ..., \alpha_k; z^j] = [\alpha_1, ...\alpha_k; z^{j-1}]\sigma^{-(j-1)}(\alpha_k) + [\alpha_1, ..., \alpha_{k-1}; z^{j-1}].$$

Proof: Using Lemma 3.2.9 we immediately have

$$L_{\alpha_1} z^j = (L_{\alpha_1} L_0 z^j)\sigma^{-(j-1)}(\alpha_1) + L_0 z^j.$$

And so, by the linearity of L_{α_i}, for every i,

$$\begin{aligned} L_{\alpha_{k-1}} \cdots L_{\alpha_1} z^j &= L_{\alpha_{k-1}} \cdots L_{\alpha_2}[(L_{\alpha_1} L_0 z^j)\sigma^{-(j-1)}(\alpha_1) + L_0 z^j] \\ &= (L_{\alpha_{k-1}} \cdots L_{\alpha_1} z^{j-1})\sigma^{-(j-1)}(\alpha_1) + L_{\alpha_{k-1}} \cdots L_{\alpha_2} z^{j-1}. \end{aligned}$$

Now, evaluating both the left- and right-hand sides at α_k, knowing that skew polynomial evaluation is right linear and using the definition of left divided differences,

$$\begin{aligned} & (L_{\alpha_{k-1}} \cdots L_{\alpha_1} z^j)(\alpha_k) = (L_{\alpha_{k-1}} \cdots L_{\alpha_1} z^{j-1})(\alpha_k) \cdot \sigma^{-(j-1)}(\alpha_1) \\ & \qquad\qquad + (L_{\alpha_{k-1}} \cdots L_{\alpha_2} z^{j-1})(\alpha_k) \\ \iff & [\alpha_k, ..., \alpha_1; z^j] = [\alpha_k, ..., \alpha_1; z^{j-1}]\sigma^{-(j-1)}(\alpha_1) + [\alpha_k, ..., \alpha_2; z^{j-1}] \\ \iff & [\overbrace{\alpha, ..., \alpha}^{k\times}; z^j] = [\overbrace{\alpha, ..., \alpha}^{k\times}; z^{j-1}]\sigma^{-(j-1)}(\alpha) + [\overbrace{\alpha, ..., \alpha}^{(k-1)\times}; z^{j-1}] \end{aligned}$$

where the last statement, itself equivalent to the initial statement of the lemma, holds by the identity of α_1 through α_k. □

The notion of derivatives for skew polynomials is not as is typically. Derivatives will be defined only through values at particular points, as seen in **[18]**. The peculiarity of this circumstance will ultimately force us to seek other means of demonstrating the conditions of interpolation (in fact using the operators L_α).

Definition 3.2.11 The *k-th derivative of* $f \in \mathbf{D}[z;\sigma]$ (of degree n) *evaluated*

at $\alpha \in \mathbf{D}$ is

$$f^{(k)}(\alpha) := \sum_{j=1}^{n} N_j^k(\alpha) f_j$$

where, for $1 \leq k < j$,

$$N_j^k(\alpha) := k! \sum_{0 \leq r_{j-k} < \cdots < r_2 < r_1 \leq j-1} \sigma^{-r_{j-k}}(\alpha) \cdots \sigma^{-r_2}(\alpha) \sigma^{-r_1}(\alpha)$$

with $N_j^j(\alpha) = j!$ and $N_j^k(\alpha) = 0$ for $k > j \geq 0$.

By convention $f^{(0)}(\alpha) := f(\alpha)$.

While reducing to the canonical case for formal derivatives of polynomials over division rings, there is a marked difference. We can no longer speak of the "derivative" as though it were itself another polynomial, and then, e.g., determine by degree how many zeros such a polynomial must have, and then use minimal polynomials, etc. We shall nevertheless get by without such luxury.

Of needful properties, the evaluation of derivatives at a point is right linear; this is the substance of the following lemma.

Lemma 3.2.12 For $f, g \in \mathbf{D}[z; \sigma]$, without loss of generality of same degree, and $\gamma, \alpha \in \mathbf{D}$,

$$(f + g\gamma)^{(k)}(\alpha) = f^{(k)}(\alpha) + g^{(k)}(\alpha) \cdot \gamma.$$

Proof:

$$\begin{aligned}(f+g\gamma)^{(k)}(\alpha) &= \sum_{j=1}^{n} N_j^k(\alpha)(f+g\gamma)_j \\ &= \sum_{j=1}^{n} N_j^k(\alpha)(f_j + g_j\gamma) \\ &= \sum_{j=1}^{n} (N_j^k(\alpha)f_j + N_j^k(\alpha)g_j\gamma) \\ &= \sum_{j=1}^{n} N_j^k(\alpha)f_j + (\sum_{j=1}^{n}(N_j^k(\alpha)g_j)\gamma \\ &= f^{(k)}(\alpha) + g^{(k)}(\alpha)\cdot\gamma\end{aligned}$$

□

Significantly, we have the following formula; it allows us to relate a polynomial to its evaluation of derivatives at points.

Theorem 3.2.13 For $f \in \mathbf{D}[z;\sigma]$ such that $\deg(f) = n$,

$$f = \sum_{k=0}^{n} (z-\alpha)^k \cdot \frac{f^{(k)}(\alpha)}{k!}$$

Proof: We first prove the statement for $f(z) = z^n$; since the derivation operation is right linear, by extension the lemma will hold for every skew polynomial. The proof proceeds by induction on polynomial degree.

Beginning with Lemma 3.2.5 and knowing that $\deg(L_\alpha f) = n - 1 < n$, we

have that $f(z) = (z-\alpha)(L_\alpha f) + f(\alpha)$ which by the induction hypothesis is equivalent to

$$\begin{aligned} & f(z) = (z-\alpha)\sum_{k=0}^{n-1}(z-\alpha)^k \cdot \frac{(L_\alpha f)^{(k)}(\alpha)}{k!} + f(\alpha) \\ \iff & f(z) = \sum_{k=1}^{n}(z-\alpha)^k \cdot \frac{(L_\alpha f)^{(k-1)}(\alpha)}{(k-1)!} + f(\alpha). \end{aligned}$$

Hence, by the last equality above, to prove the theorem we must show that

$$\begin{aligned} & \sum_{k=1}^{n}(z-\alpha)^k \cdot \frac{(L_\alpha f)^{(k-1)}(\alpha)}{(k-1)!} + f(\alpha) = \sum_{k=0}^{n}(z-\alpha)^k \cdot \frac{f^{(k)}(\alpha)}{k!} \\ \iff & \sum_{k=1}^{n}(z-\alpha)^k \cdot (\frac{f^{(k)}(\alpha)}{k!} - \frac{(L_\alpha f)^{(k-1)}(\alpha)}{(k-1)!}) \equiv 0 \\ \iff & \frac{f^{(k)}(\alpha)}{k!} = \frac{(L_\alpha f)^{(k-1)}(\alpha)}{(k-1)!}, \forall k \leq n. \end{aligned}$$

However, given that $f(z) = z^n$, we have for $k = n$ that as $N_j^j(\alpha) = j!$, for every $j \in \mathbb{N}$, then $\frac{n!}{n!} = \frac{(n-1)!}{(n-1)!}$, which is true, and for $k < n$,

$$\frac{f^{(k)}(\alpha)}{k!} = \sum_{0 \leq r_{n-k} < \cdots < r_1 \leq n-1} \sigma^{-r_{n-k}}(\alpha) \cdots \sigma^{-r_1}(\alpha). \tag{3.2.2}$$

Also, recalling the proof of Lemma 3.2.5, using the polynomial z^n we have

$L_\alpha z^n = \sum_{j=0}^{n-1} z^j \left[\prod_{0 \le l < n-j-1} \sigma^{-(l+j+1)}(\alpha) \right]$, which implies that

$$\frac{(L_\alpha z^n)^{(k-1)}(\alpha)}{(k-1)!} = \frac{1}{(k-1)!} \sum_{j=1}^{n-1} N_j^{k-1}(\alpha) \left[\prod_{0 \le l < n-j-1} \sigma^{-(l+j+1)}(\alpha) \right]$$

$$= \sum_{j=k}^{n-1} \sum_{0 \le r_{j-k+1} < \cdots < r_1 \le j-1} \sigma^{-r_{j-k+1}}(\alpha) \cdots \sigma^{-r_1}(\alpha) \left[\prod_{j+1 \le l < n} \sigma^{-l}(\alpha) \right].$$

(The case where $\alpha = 0$ may now be seen as trivial; assume otherwise.) So then, comparing this result to equation (3.2.2) which it must equal, we have, iff,

$$\sum_{j=k}^{n-1} \sum_{0 \le r_{j-k+1} < \cdots < r_1 \le j-1} \sigma^{-r_{j-k+1}}(\alpha) \cdots \sigma^{-r_1}(\alpha) \left[\prod_{j+1 \le l < n} \sigma^{-l}(\alpha) \right]$$
$$= \sum_{0 \le r_{n-k} < \cdots < r_1 \le n-1} \sigma^{-r_{n-k}}(\alpha) \cdots \sigma^{-r_1}(\alpha).$$

We prove this last statement by using induction on n. With the base cases trivial, we proceed as follows:

$$RHS = \sum_{0 \le r_{n-k} < \cdots < r_1 \le n-1} \sigma^{-r_{n-k}}(\alpha) \cdots \sigma^{-r_1}(\alpha)$$
$$= \sum_{0 \le r_{n-k} < \cdots < r_1 = n-1} \sigma^{-r_{n-k}}(\alpha) \cdots \sigma^{-r_1}(\alpha)$$
$$+ \sum_{0 \le r_{n-k+1} < \cdots < r_1 \le n-2} \sigma^{-r_{n-k+1}}(\alpha) \cdots \sigma^{-r_1}(\alpha).$$

Also,

$$\begin{aligned} LHS &= \sum_{j=k}^{n-1} \sum_{0\le r_{j-k+1}<\cdots<r_1\le j-1} \sigma^{-r_{j-k+1}}(\alpha)\cdots\sigma^{-r_1}(\alpha)\left[\prod_{j+1\le l<n}\sigma^{-l}(\alpha)\right] \\ &= \sum_{j=k}^{n-2} \sum_{0\le r_{j-k+1}<\cdots<r_1\le j-1} \sigma^{-r_{j-k+1}}(\alpha)\cdots\sigma^{-r_1}(\alpha)\left[\prod_{j+1\le l<n}\sigma^{-l}(\alpha)\right] \\ &\quad + \sum_{0\le r_{n-k+1}<\cdots<r_1\le n-2} \sigma^{-r_{n-k+1}}(\alpha)\cdots\sigma^{-r_1}(\alpha)\left[\prod_{n\le l<n}\sigma^{-l}(\alpha)\right]. \end{aligned}$$

Hence, setting the right-hand side equal to the left we have that, the two latter terms of each sum canceling, the result is

$$\begin{aligned} &\sum_{0\le r_{n-k}<\cdots<r_1=n-1} \sigma^{-r_{n-k}}(\alpha)\cdots\sigma^{-r_1}(\alpha) \\ &= \sum_{j=k}^{n-2} \sum_{0\le r_{j-k+1}<\cdots<r_1\le j-1} \sigma^{-r_{j-k+1}}(\alpha)\cdots\sigma^{-r_1}(\alpha)\left[\prod_{j+1\le l<n}\sigma^{-l}(\alpha)\right] \end{aligned}$$

which is equivalent to

$$\begin{aligned} &\left[\sum_{0\le r_{n-k+1}<\cdots<r_1\le n-2} \sigma^{-r_{n-k+1}}(\alpha)\cdots\sigma^{-r_1}(\alpha)\right]\sigma^{-(n-1)}(\alpha) \\ &= \left[\sum_{j=k}^{n-2} \sum_{0\le r_{j-k+1}<\cdots<r_1\le j-1} \sigma^{-r_{j-k+1}}(\alpha)\cdots\sigma^{-r_1}(\alpha)\left[\prod_{j+1\le l<n-1}\sigma^{-l}(\alpha)\right]\right]\sigma^{-(n-1)}(\alpha). \end{aligned}$$

Now divide both sides by $\sigma^{-(n-1)}(\alpha)$, which value is not 0 as $\alpha \neq 0$ by assumption and σ^r is an automorphism, for all $r \in \mathbb{Z}$. But what remains is simply the induction hypothesis. □

As simple as it is to be stated, and now essentially proven, the next remark establishes the all-important connection between our backward shift opera-

tors L_α and the evaluation of derivatives.

Remark 3.2.14 For $k \geq 0$, $[\overbrace{\alpha, ..., \alpha}^{(k+1)\times}; f] = \dfrac{f^{(k)}(\alpha)}{k!}$.

Proof: Compare Remark 3.2.2 with Theorem 3.2.13. □

3.3 Skew Polynomial Hermite Interpolation

We will actually work with a more general form for the Hermite interpolation problem. Each α_i is thus related to an independently chosen number k_i, specifying the next-to-highest derivative f takes values for at this point. I.e., we have target values $(\alpha_i, \beta_i^1), ..., (\alpha_i, \beta_i^{k_i})$, where $1 \leq i \leq n$. This generalization will cost only some little extra effort.

We'll see in the inclusion and synthesis of matrices with our already developed skew polynomial theory the means to allow for the interpolating function f to be constructed as desired. Hence we introduce the σ-Vandermonde matrix for motivation, and then the confluent σ-Vandermonde matrix, our true object of interest.

Definition 3.3.1 For $\alpha_1, ..., \alpha_n \in \mathbf{D}$, the *left σ-Vandermonde matrix* is

$$V_n^\sigma(\alpha_1, ..., \alpha_n) = \begin{bmatrix} 1 & M_1(\alpha_1) & M_2(\alpha_1) & \cdots & M_{n-1}(\alpha_1) \\ 1 & M_1(\alpha_2) & M_2(\alpha_2) & \cdots & M_{n-1}(\alpha_2) \\ 1 & M_1(\alpha_3) & M_2(\alpha_3) & \cdots & M_{n-1}(\alpha_3) \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & M_1(\alpha_n) & M_2(\alpha_n) & \cdots & M_{n-1}(\alpha_n) \end{bmatrix}.$$

What is known merely as a Vandermonde matrix is the above, but with $\sigma = Id$, in which case, there is no distinction between left or right. The next two definitions build into a generalization of the previous.

Definition 3.3.2 Taking $\alpha \in \mathbf{D}$, the *left confluent σ-Vandermonde matrix* is, for $1 \leq i \leq k$ and $1 \leq j \leq m$,

$$V_{k,m}^\sigma(\alpha) := ([\overbrace{\alpha, ..., \alpha}^{i\times}; z^{j-1}])_{i,j} \in \mathbf{D}^{k\times m}.$$

Definition 3.3.3 For $\alpha_0, ..., \alpha_n \in \mathbf{D}$, also called the *left confluent σ-Vandermonde matrix* is, for $\mathbf{k} = (k_1, ..., k_n) \in \mathbb{N}^n$,

$$V_{\mathbf{k},m}^\sigma(\alpha_1, ..., \alpha_n) := \begin{bmatrix} V_{k_1,m}^\sigma(\alpha_1) \\ \vdots \\ V_{k_n,m}^\sigma(\alpha_n) \end{bmatrix} \in \mathbf{D}^{(k_1+\cdots+k_n)\times m}.$$

It can be seen that the confluent σ-Vandermonde matrix of Definition 3.3.3

with parameters $k_1 = \cdots = k_n = 1$ is exactly the σ-Vandermonde matrix of Definition 3.3.1.

Our next remark provides a way of recursively computing the matrix entries of Definition 3.3.2, once we've computed the first row. Its method may, of course, be easily extended to the more complex form of matrix as given in Definition 3.3.3.

Remark 3.3.4 Define $V = V^{\sigma}_{k,m}(\alpha)$. For $1 \leq i < k$ and $1 \leq j < m$

$$V_{i+1,j+1} = V_{i+1,j} \cdot \sigma^{-j}(\alpha) + V_{i,j}.$$

Proof: An immediate consequence of Lemma 3.2.10. □

Observe further that by combining Definition 3.3.2, Definition 3.2.11 and Remark 3.2.14,

$$V^{\sigma}_{k,m}(\alpha) = (N^{(i-1)}_{j-1}(\alpha))_{i,j} \in \mathbf{D}^{k \times m},$$

which is to show that the left confluent σ-Vandermonde matrix is populated simply by the values of our derivative function N evaluated at α. But this is the key to relating our desired interpolating function f; the following remarks are thusly immediate.

Remark 3.3.5 The interpolating function $f(z) = \sum_{i=0}^{m-1} z^i f_i$, for a given l,

$1 \leq l \leq n$, must satisfy

$$V^{\sigma}_{k_l,m}(\alpha_l) \begin{bmatrix} f_0 \\ \vdots \\ f_{m-1} \end{bmatrix} = \begin{bmatrix} \beta_l^1 \\ \vdots \\ \beta_l^{k_l} \end{bmatrix}$$

.

where $f^{(j-1)}(\alpha_l) = \beta_l^j$, for $1 \leq j \leq k_l$.

Remark 3.3.6 The interpolating function $f(z) = \sum_{i=0}^{m-1} z^i f_i$ must satisfy

$$V^{\sigma}_{\mathbf{k},m}(\alpha_1, ..., \alpha_n) \begin{bmatrix} f_0 \\ \vdots \\ f_{m-1} \end{bmatrix} = \begin{bmatrix} B_1 \\ \vdots \\ B_n \end{bmatrix}$$

where $B_l = (\beta_l^1, ..., \beta_l^{k_l})^T$.

Observe also that we have yet to specify precisely which $m \in \mathbb{N}$ we use. Indeed, at the very least, by considerations of linear dependence for vector spaces over division rings, the left confluent σ-Vandermonde matrix may guarantee solutions for f so long as $m \geq k_1 + \cdots + k_n$.

The solution for f could be uniquely existent precisely when equality holds, and then the matrix $V^{\sigma}_{\mathbf{k},m}(\alpha_1, ..., \alpha_n)$ would be invertible, allowing for a direct computation of $f_0, ..., f_{m-1}$. This is the case we shall concern ourselves with

primarily.

Discovering then more precise conditions upon the entries of $V^{\sigma}_{\mathbf{k},m}(\alpha_1, ..., \alpha_n)$ - and hence of the relations on $\alpha_1, ..., \alpha_n$ - which allow matrix invertibility, and so determining this f, will be our present purpose.

The next lemma is for the "commutation" of linear factors in our setting.

Lemma 3.3.7 For nonzero $\alpha, \beta \in \mathbf{D}$, there exist $c_1, d_1, c_2, d_2 \in \mathbf{D}$ such that

$$(z - \alpha)(z - {}^{c_1}\beta) = (z - \beta)(z - {}^{d_1}\alpha) \tag{3.3.1}$$

$$(z - \alpha)(z - \beta) = (z - {}^{c_2}\beta)(z - {}^{d_2}\alpha). \tag{3.3.2}$$

Proof: We first prove equation (3.3.1); to this end, note that

$$\begin{aligned} & (z - \alpha)(z - {}^{c_1}\beta) = (z - \beta)(z - {}^{d_1}\alpha) \\ \iff & z^2 - z(\sigma^{-1}(\alpha) + {}^{c_1}\beta) + \alpha \cdot {}^{c_1}\beta = z^2 - z(\sigma^{-1}(\beta) + {}^{d_1}\alpha) + \beta \cdot {}^{d_1}\alpha \\ \iff & [\sigma^{-1}(\alpha) + {}^{c_1}\beta = \sigma^{-1}(\beta) + {}^{d_1}\alpha] \;\&\; [\alpha \cdot {}^{c_1}\beta = \beta \cdot {}^{d_1}\alpha]. \end{aligned} \tag{3.3.3}$$

As the case $\alpha = \beta$ is trivial assume otherwise. Setting $c_1 = \beta - \alpha$ and $d_1 = \alpha - \beta$, knowing that $\alpha \neq \beta$ allows $(\alpha - \beta)^{-1}$ etc. to exist, we get, for

the first equation of (3.3.3) using the definition of σ-conjugacy,

$$\begin{aligned}
& \sigma^{-1}(\alpha) + {}^{c_1}\beta = \sigma^{-1}(\beta) + {}^{d_1}\alpha \\
\iff & \sigma^{-1}(\alpha) + {}^{\beta-\alpha}\beta = \sigma^{-1}(\beta) + {}^{\alpha-\beta}\alpha \\
\iff & \sigma^{-1}(\alpha) + (\beta-\alpha)^{-1}\beta\sigma^{-1}(\beta-\alpha) = \sigma^{-1}(\beta) + (\alpha-\beta)^{-1}\alpha\sigma^{-1}(\alpha-\beta).
\end{aligned}$$

We next make some minor algebraic modifications (shifting negatives), then left-multiply both sides of the equation by $(\alpha - \beta)$:

$$\begin{aligned}
& \sigma^{-1}(\alpha) + [-(\alpha-\beta)^{-1}]\beta\sigma^{-1}(\beta-\alpha) = RHS \\
\iff & \sigma^{-1}(\alpha) + (\alpha-\beta)^{-1}\beta\sigma^{-1}(\alpha-\beta) = RHS \\
\iff & (\alpha-\beta)\sigma^{-1}(\alpha) + \beta\sigma^{-1}(\alpha-\beta) = (\alpha-\beta)\sigma^{-1}(\beta) + \alpha\sigma^{-1}(\alpha-\beta)
\end{aligned}$$

and, lastly, using the properties of σ^{-1} as an automorphism,

$$\begin{aligned}
& \alpha\sigma^{-1}(\alpha) - \beta\sigma^{-1}(\alpha) + \beta\sigma^{-1}(\alpha) - \beta\sigma^{-1}(\beta) \\
& = \alpha\sigma^{-1}(\beta) - \beta\sigma^{-1}(\beta) + \alpha\sigma^{-1}(\alpha) - \alpha\sigma^{-1}(\beta)
\end{aligned}$$

which holds iff $0 = 0$, a tautology.

Now, for the second equation of (3.3.3), we expand according to the defini-

tions and shift negatives to obtain

$$\begin{aligned}
& \alpha \cdot {}^{c_1}\beta = \beta \cdot {}^{d_1}\alpha \\
\iff & \alpha \cdot {}^{\beta-\alpha}\beta = \beta \cdot {}^{\alpha-\beta}\alpha \\
\iff & \alpha(\beta-\alpha)^{-1}\beta\sigma^{-1}(\beta-\alpha) = \beta(\alpha-\beta)^{-1}\alpha\sigma^{-1}(\alpha-\beta) \\
\iff & \alpha(\alpha-\beta)^{-1}\beta\sigma^{-1}(\alpha-\beta) = \beta(\alpha-\beta)^{-1}\alpha\sigma^{-1}(\alpha-\beta)
\end{aligned}$$

whence we right multiply by $[\sigma^{-1}(\alpha-\beta)]^{-1}$ (which exists as σ^{-1} is an automorphism with $\alpha \neq \beta$) and factor out constants, to get

$$\begin{aligned}
& \alpha(\alpha-\beta)^{-1}\beta = \beta(\alpha-\beta)^{-1}\alpha \\
\iff & \alpha[\beta(\beta^{-1}-\alpha^{-1})\alpha]^{-1}\beta = \beta[\alpha(\beta^{-1}-\alpha^{-1})\beta]^{-1}\alpha \\
\iff & (\beta^{-1}-\alpha^{-1})^{-1} = (\beta^{-1}-\alpha^{-1})^{-1}.
\end{aligned}$$

Now to prove equation (3.3.2), use equation (3.3.1) replacing β by ${}^{c_1^{-1}}\beta$, with c_1 as before. Knowing that ${}^{c_1}({}^{c_1^{-1}}\beta) = \beta$ take $c_2 = {c_1}^{-1}$, $d_2 = d_1$ and the result is immediate. □

The next theorem is key to Hermite interpolation in the skew polynomial case.

Theorem 3.3.8 For $\alpha_1, ..., \alpha_n \in \mathbf{D}$ all nonconjugate and nonzero, target values $\beta_i^j \in \mathbf{D}$ with $1 \leq i \leq n$ and $1 \leq j \leq k_i$, where $m = k_1 + \cdots + k_n$ and $\mathbf{k} = (k_1, ..., k_n)$, we have that:

(A) $V^{\sigma}_{\mathbf{k},m}(\alpha_1, ..., \alpha_n)$ is invertible.

(B) There exists a unique $f \in \mathbf{D}[z;\sigma]$ such that $\deg(f) \leq m-1$ and $f^{(j-1)}(\alpha_i) = \beta_i^j$.

Proof: Apparently, by Remark 3.3.6, if $V^{\sigma}_{\mathbf{k},m}(\alpha_1, ..., \alpha_n)$ is in fact invertible then (B) will be an immediate consequence of using matrix multiplication by an inverse, thus computing unique coefficients $f_0, ..., f_{m-1}$. So, prove (A); the outset is as the proof of Theorem 4.4 in **[18]** but takes matters further with the methods of **[7]**.

Oddly enough, we don't begin directly with the matrix $V^{\sigma}_{\mathbf{k},m}(\alpha_1, ..., \alpha_n)$, which only appears at the proof's end. We actually require a more general, but closely related, linear transformation. Define $\Phi : \mathbf{D}[z;\sigma] \to \mathbf{D}^m$ by

$$\Phi(h) = (h^{(0)}(\alpha_1), ..., h^{(k_1-1)}(\alpha_1), ..., h^{(0)}(\alpha_n), ..., h^{(k_n-1)}(\alpha_n))$$

It is evident that Φ is right-linear, as skew polynomial evaluation at derivatives is itself right-linear. This implies that $Ker(\Phi)$ is a right ideal of $\mathbf{D}[z;\sigma]$, and as this ring is a left PID by virtue of its left division algorithm, $Ker(\Phi)$ will be principal. (Henceforth, for $f \in \mathbf{D}[z;\sigma]$, let $\langle f \rangle$ denote the right ideal generated by f.)

Defining $\Delta = ((\alpha_1, k_1), ..., (\alpha_n, k_n))$, let f_Δ denote the (monic) generator of $Ker(\Phi)$; as is categorical for this setting, f_Δ will be unique and of minimal

degree. Also, observe that $Ker(\Phi) \neq \langle 0 \rangle$ as Φ by definition has a domain of necessarily higher vector space dimension than the codomain. I.e., $f_\Delta \neq 0$.

By the conditions attendant on $f_\Delta \in Ker(\Phi)$ we have, for $1 \leq i \leq n$ and $0 \leq j \leq (k_i - 1)$, that

$$f_\Delta^{(j)}(\alpha_i) = 0 \iff [\overbrace{\alpha_i, ..., \alpha_i}^{(j+1)\times}; f_\Delta] = 0 \iff (L_{\alpha_i}^j f_\Delta)(\alpha_i) = 0$$

where $L_{\alpha_i}^j$ denotes the application of L_{α_i} j times.

Now, for $P_i^j = (z - \alpha_i)^j$, $j \in \mathbb{N}$, by Remark 3.2.2 we have that $h \in \langle P_i^j \rangle_r \iff h^{(l)}(\alpha_i) = 0$ for $0 \leq l < j$. Hence $f_\Delta \in \langle P_i^{k_i} \rangle_r$, for $1 \leq i \leq n$.

Beginning with k_1 we have that $f_\Delta = P_1^{k_1} \cdot g_1(z)$ for some $g_1 \in \mathbf{D}[z; \sigma]$, possibly of degree 0, i.e., $g_1(z) \equiv 1$. Hence $\deg(f_\Delta) \geq k_1$; but if we can prove that $\deg(f_\Delta) \geq k_1 + k_2$ then, by continuing inductively, we will have that $\deg(f_\Delta) \geq k_1 + \cdots + k_n$ and moreover equality will hold as f_Δ is the minimal polynomial to satisfy the given criteria of the evaluation of derivatives.

So suppose $g_1 \equiv 1$; as $f_\Delta(\alpha_2) = 0$ we have then, using Theorem 3.1.6 on product evaluations,

$$\begin{aligned} 0 &= P_1^{k_1}(\alpha_2) \\ &= ({}^{c_1}\alpha_2 - \alpha_1)({}^{c_2}\alpha_2 - \alpha_1) \cdots ({}^{c_{k_1}}\alpha_2 - \alpha_1) \end{aligned}$$

where $c_1 = 1$, and for $1 \leq l < k_1$, $c_{l+1} = (\alpha_2 - \alpha_1) \cdots ({}^{c_l}\alpha_2 - \alpha_1)$. Though as $\mathbf{D}$ is a division ring it must be that one of the factors of the product $({}^{c_1}\alpha_2 - \alpha_1)({}^{c_2}\alpha_2 - \alpha_1) \cdots ({}^{c_{k_1}}\alpha_2 - \alpha_1)$ is itself 0. But this requires, for some l, that ${}^{c_l}\alpha_2 = \alpha_1$, i.e., $\alpha_2 \sim \alpha_1$, contrary to supposition.

Now assume that $\deg(g_1) \geq 1$. Again, $f_\Delta(\alpha_2) = 0$; yet we know $P_1^{k_1}(\alpha_2) = d_1 \neq 0$ for some $d_1 \in \mathbf{D}$, so it must be that $g_1({}^{d_1}\alpha_2) = 0$. Now, by use of the division algorithm on $\mathbf{D}[z;\sigma]$ this gives us that $g_1 = (z - {}^{d_1}\alpha_2)g_2(z)$ for some $g_2(z)$ of degree greater than or equal to 0.

If we apply Lemma 3.3.7 repeatedly we get that

$$\begin{aligned} f_\Delta &= P_1^{k_1}(z - {}^{d_1}\alpha_2)g_2 \\ &= (z - {}^{\widetilde{d}_1}\alpha_2)\widetilde{P}_1^{k_1} g_2 \end{aligned}$$

where $\widetilde{P}_1^{k_1}$ is whatever polynomial, a product of linear factors, results from using Lemma 3.3.7 k_1 times, and $\widetilde{d}_1$ whatever conjugating element. Since we also know that, by assumption, $f_\Delta(\alpha_2) = 0$ it must be that $\widetilde{d}_1 = 1$, as the factor $(z - {}^{\widetilde{d}_1}\alpha_2)$ was that which, before permuting, evaluated f_Δ to 0 to begin with.

But now, using our assumption on f_Δ and the "division" property of L_{α_2}, $(L_{\alpha_2} f_\Delta)(\alpha_2) = (\widetilde{P}_1^{k_1} g_2)(\alpha_2) = 0$.

And so, by the same argument as before, given that $\widetilde{P}_1^{k_1}$ is composed only

of linear factors in ${}^{c}\alpha_1$, varying $c \in \mathbf{D}$ per factor, we can prove $\deg(g_2) \geq 1$. However, our assumptions on f_Δ only guarantee this until k_2 applications of L_{α_2}. Hence $\deg(f_\Delta) \geq k_1 + k_2$, as required, and so indeed $\deg(f_\Delta) = k_1 + \cdots + k_n$.

Now suppose that $V^{\sigma}_{\mathbf{k},m}(\alpha_1, ..., \alpha_n)$ is not invertible; this is equivalent to stating that there exists $h \in \mathbf{D}[z;\sigma]$ such that $h \neq 0$ and $\Phi(h) = V^{\sigma}_{\mathbf{k},m}(\alpha_1, ..., \alpha_n) \cdot (h_0, ..., h_{m-1})^T = 0$. But this is impossible as f_Δ is the minimal polynomial associated to Φ, and is itself of degree $m > (m-1) \geq \deg(h)$. □

It's worth noting that in the abstract sense there's no saying precisely what conditions elements must satisfy to be σ-conjugate, other than blankly repeating the definition. In this way we have a priori no knowledge of the appearance or form of equivalence classes. As well, our secret sharing schemes must first specify concrete division rings and automorphisms to be effectively computable.

3.4 The Quaternion-Conjugation Skew Polynomial Ring

Though there are a number of possibilities, in keeping with our focus on quaternions our skew polynomial ring of choice is $\mathbb{H}[z;\bar{}]$. I.e., take the real

quaternions as our division ring with conjugation as the automorphism.

The next lemma gives us some criteria for determining when elements are σ-conjugate.

Lemma 3.4.1 If $\sigma(\alpha) = \overline{\alpha}$ for every $\alpha \in \mathbb{H}$, then $\beta, \gamma \in \mathbb{H}$ are left σ-conjugates only if $\|\gamma\| = \|\beta\|$.

Proof: $\beta \sim \gamma$ means there exists nonzero $c \in \mathbb{H}$ such that ${}^c\beta = \gamma$. But then $c^{-1}\beta\overline{c} = \gamma$ which means

$$\begin{aligned} \|\gamma\| &= \|c^{-1}\beta\overline{c}\| \\ &= \|c^{-1}\| \cdot \|\beta\| \cdot \|\overline{c}\| \\ &= \|c\|^{-1} \cdot \|\beta\| \cdot \|c\| \\ &= \|\beta\|. \end{aligned}$$

□

In this way, we can more methodically find acceptable base elements to use when interpolating polynomials: simply choose elements that all have different moduli.

Chapter 4

The Quaternion Lagrange Shamir Scheme

The following secret sharing scheme is intended as an extension of the original Shamir scheme of [**32**] to polynomials over quaternions.

However, Shamir's scheme chooses its polynomials uniformly over the field $\mathbb{F}_p$ (with p prime), though the analogous set of quaternions, denoted by $\mathbb{F}_p^{\mathbb{H}}$, being all elements

$$a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k} \text{ where } a, b, c, d \in \mathbb{F}_p, \text{ with } \mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1,$$

is not even a division ring, much less a field [**3**]. So, a given element of $\mathbb{F}_p^{\mathbb{H}}$ may be lacking the inverse necessary to allow reconstruction of the desired

polynomial via the above interpolation formula of Theorem 2.2.4.

Therefore, we must find another way of randomizing the polynomial coefficients.

4.1 Setup

For this (n, t) scheme let $U_1, ..., U_t$ be the participants, and let $S \in \mathbb{N}$ be the secret. The dealer must first define an $(n-1)$ degree polynomial $g(z) \in \mathbb{H}[z]$ of the form $g(z) = z^{n-1}a_{n-1} + \cdots + za_1 + a_0$, chosen randomly, except that $a_0 = S$.

The random choice of polynomial coefficients is as follows:

For $1 \leq m \leq n-1$, choose $a_m = a_{m0} + a_{m1}\mathbf{i} + a_{m2}\mathbf{j} + a_{m3}\mathbf{k}$, where each a_{ml}, $0 \leq l \leq 3$, is selected randomly and uniformly from the set $\{1, ..., N\}$ for some fixed $N \in \mathbb{N}$ taken as large as desired.

However, $a_0 = S + a_{01}\mathbf{i} + a_{02}\mathbf{j} + a_{03}\mathbf{k}$, with a_{0l} chosen randomly from $\{1, ..., N\}$, $1 \leq l \leq 3$.

He then evaluates $d_r = g(q_r)$ for $1 \leq r \leq t$, where $q_r = r + r^2\mathbf{i} + r^3\mathbf{j} + r^4\mathbf{k}$. Note that by Theorem 2.1.4 none of these q_r will be conjugate, since their real components all differ.

The dealer then sends (q_r, d_r) to participant U_r, for every r.

As an aside: The significance of choosing our quaternion polynomials with natural number coefficients is to ensure that a computer, with necessarily finite memory and processing power, may be able, after all involved operations, to return the secret S exactly.

It's true that inversion, as needed in the reconstruction phase, will inevitably result in *rational* quaternions, though symbolic computation is nevertheless able to work with rationals without loss of information.

4.2 Reconstruction

Supposing, without loss of generality, that $U_1, ..., U_n$ wish to reconstruct S, they then pool their shares, giving each U_i the sequence $(q_1, d_1), ..., (q_n, d_n)$ from which he uses the quaternion interpolation model as follows:

Using $\Lambda = \{q_1, ..., q_n\}$ and denoting by Λ_k the set $\Lambda \setminus \{q_k\}$, by Theorem 2.2.4, while knowing all q_k are pairwise nonconjugate, we have

$$h(z) = \sum_{k=1}^{n} P_{\Lambda_k,l}(z) \cdot P_{\Lambda_k,l}(q_k)^{-1} \cdot d_k$$

with $P_{\Lambda_k,l}$ computed recursively according to the formula of Theorem 2.2.3.

Participant U_i then computes $h(0) = a_0$ and then observes $Re(a_0) = S$ to retrieve the secret.

It may be questioned as to whether or not $h(0) = g(0) = a_0$, since $h(z)$ is

not thus obtained in the same form as $g(z)$ with all its variables collected.

Though, by the uniqueness condition of Theorem 2.2.4, we can be assured of its correctness, and that also $h(z) = g(z)$ itself, since both $h(z)$ and $g(z)$ are ultimately polynomials of degree $(n-1)$ with n targets. One is but a different way of expressing the other, and the recursive $h(z)$ was constructed so that this equality would hold given our meaning of polynomial evaluation (Definition 2.1.6).

4.3 Complexity Analysis

The two phases of Setup and Reconstruction will be evaluated separately, in terms of the total number of participants t and the threshold n, respectively. The complexity of multiplying and adding quaternions will be assumed as $O(1)$ for clarity.

4.3.1 Setup Complexity

First suppose we have a random oracle to supply the four components of each a_m, for every m (being $a_{m0}, a_{m1}, a_{m2}, a_{m3}$), so that each choice is made with a complexity of $O(1)$. There are $4(n-1)$ choices to be made in assembling $g(z)$, giving a complexity of $O(n)$. Though $t \geq n$ means this step is also of complexity $O(t)$.

All that remains is for the dealer to evaluate $g(z)$ at $q_1, ..., q_t$. Though according to [**39**] a quaternion polynomial of degree m may be evaluated at points $b_1, ..., b_m$ with complexity $O(m \log^2 m)$. So consider $g(z)$ as a polynomial of degree t (with however many zero coefficients above the n-th), giving a complexity of $O(t \log^2 t)$.

Hence, as we can always find some fixed $x \in \mathbb{N}$ such that $\log^2 x \geq 1$, in asymptotic terms the $O(t)$ is overshadowed by $O(t \log^2 t)$, which latter becomes the total setup complexity.

4.3.2 Reconstruction Complexity

For a given participant U_i they must first construct the function $g(z)$ before obtaining $S = g(0)$. Now, by definition,

$$g(z) = \sum_{k=1}^{n} P_{\Lambda_k,l}(z) \cdot P_{\Lambda_k,l}(q_k)^{-1} \cdot d_k.$$

However, since all the q_k's are pairwise nonconjugate we must use the recursive formula for each $P_{\Lambda_k,l}(z)$, and so the methods of [**39**] do not apply here - as in their case the polynomial was already given.

Hence, we have $P_{\Lambda_k,l}(z) = p_{n-1}(z)$ with $p_0(z) \equiv 1$ and

$$p_{s+1}(z) = p_s(z)(z - p_s(q_{s+1})^{-1} q_{s+1} p_s(q_{s+1}))$$

for $1 \leq s \leq n-2$. (We avoid overcomplicating the indices, though p_{n-1} depends on k; and for any given k we've relabeled the elements of $\Lambda_k = \{q_1, ..., q_{n-1}\}$.)

Suppose we're moving to step $s+1$, where $s \geq 1$. Assume that quaternion inversion also takes $O(1)$ steps; obtaining p_{s+1} from p_s depends mainly on computing $p_s(q_{s+1})$, since the multiplications, inversion, and subtraction are $O(1), O(1)$ and $O(1)$ complexity respectively.

Now, as we've already computed $p_s(z)$, which by the form of the induction equation will be a product of linear factors, we must go back to definition Definition 2.1.6 to decide the complexity of evaluating $p_s(q_{s+1})$.

So suppose that $p_s(z) = (z-a_1)\cdots(z-a_s)$ for some $a_1, ..., a_s \in \mathbb{H}$ which have been previously calculated. Let $f_i = (z-\alpha_i)$ for brevity, $1 \leq i \leq s$. We then compute $p_s(q_{s+1})$ as follows:

Define $v_j = f_j(v_{j-1})^{-1} v_{j-1} f_j(v_{j-1})$ where $v_0 = q_{s+1}$ and $1 \leq j \leq s$; to adjudge maximal complexity we've assumed that no j is such that $f_j(v_{j-1}) = 0$.

Then, by repeated use of Definition 2.1.6 and induction,

$$\begin{aligned} p_s(q_{s+1}) &= (f_1 \cdots f_s)(v_0) = f_1(v_0)[(f_2 \cdots f_s)(f_1(v_0)^{-1} v_0 f_1(v_0))] \\ &= f_1(v_0)(f_2 \cdots f_s)(v_1) = \cdots = f_1(v_0) f_2(v_1) \cdots f_s(v_{s-1}). \end{aligned}$$

So, begin with computing $f_1(v_0) = (v_0 - a_1)$, which takes $O(1)$ steps; computing $f_2(v_1) = (v_1 - a_2) = (f_1(v_0)^{-1}v_0 f_1(v_0) - a_2)$ will take $O(1)$ steps since $f_1(v_0)$ and v_0 are already known. This trend continues, by induction, for all $j \leq s$ so that computing $(f_1 \cdots f_s)(v_0) = p_s(q_{s+1})$ requires $O(s)$ steps.

Hence, as computing $p_{s+1}(z)$ only requires $O(1)$ further operations, it has total complexity $O(s)$. Also, as this must be done for each s, $(n-1)$ times in total, we have that computing a given $P_{\Lambda_k,l}(z)$ takes $O(1) + O(2) + \cdots + O(n-1) = O(n^2)$ steps.

Therefore, constructing $P_{\Lambda_{k,l}}(z)$ then evaluating $P_{\Lambda_k,l}(q_k)$ will take $O(n^2) + O(1) = O(n^2)$ steps. We can also now evaluate $P_{\Lambda_{k,l}}(0)$ - since even for quaternion polynomials $(f+g)(a) = f(a) + g(a), a \in \mathbb{H}$ - which will again take $O(1)$ steps. Whereafter computing $P_{\Lambda_k,l}(0) \cdot P_{\Lambda_k,l}(q_k)^{-1} \cdot d_k$ leaves the complexity at $O(n^2)$.

Hence, the summation to complete $g(0)$ takes $O(n^2)$ steps for each k, where $1 \leq k \leq n$, giving a final reconstruction complexity of $O(n^3)$.

4.4 Security Analysis

Suppose maximally, and without loss of generality, that participants $U_1, ..., U_{n-1}$ pool their shares to obtain S.

They may easily obtain $P_{\Lambda_k,l}(z)$, for $1 \leq k \leq n$, and set up

$$\begin{aligned} S = h(0) &= \sum_{k=1}^{n} P_{\Lambda_k,l}(0) \cdot P_{\Lambda_k,l}(q_k)^{-1} \cdot d_k \\ &= \left[\sum_{k=1}^{n-1} P_{\Lambda_k,l}(0) \cdot P_{\Lambda_k,l}(q_k)^{-1} \cdot d_k \right] + P_{\Lambda_n,l}(0) \cdot P_{\Lambda_n,l}(q_n)^{-1} \cdot d_n \\ &= c_{n-1} + c_n \cdot d_n \end{aligned}$$

where c_{n-1} and c_n are constant quantities known to $U_1, ..., U_{n-1}$.

However, at this point their resources fail them, as they know not a single coefficient from the original polynomial $h(z)$, so that their best guess at d_n would be entirely random. In this way, S is effectively shielded from them behind d_n; the adversaries would need some way to bound a quaternion polynomial, while yet having insufficiently many values to reconstruct it through interpolation.

Supposing, alternatively, they set up the matrix equation

$$\begin{bmatrix} 1 & q_1 & q_1^2 & \dots & q_1^n \\ 1 & q_2 & q_2^2 & \dots & q_2^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & q_n & q_n^2 & \dots & q_n^n \end{bmatrix} \begin{bmatrix} S \\ h_1 \\ \vdots \\ h_{n-1} \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix}$$

and so by inverting the left-hand matrix, Q, say, which is fully known to

$U_1, ..., U_{n-1}$, they obtain

$$\begin{bmatrix} S \\ h_1 \\ \vdots \\ h_{n-1} \end{bmatrix} = Q^{-1} \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix}$$

but as we can see all of $S, h_1, ..., h_{n-1}$ are incalculable as d_n yet remains unknown and so halts every such attempt at obtaining these coefficients.

While it is true that a correct choice of, say, h_1 would thus allow these participants to obtain d_n and so find S, the bound N may be chosen as large as desired. This would've been of value to attackers only if the number of secret polynomial coefficients were terribly deficient compared with the possible range of shares.

Also note, that for every participant less than $(n-1)$ the problem requires another share or coefficient guessed. The adversary must yet contend with somehow guessing N to begin with.

Chapter 5

The Quaternion Newton Shamir Scheme

Another iteration of the Shamir scheme with quaternion polynomials, the difference here being the model of interpolation used to recover the secret.

The Setup phase is thus identical to the Lagrange case; the significant differences are in the Reconstruction phase.

5.1 Setup

For this (n, t) scheme let $U_1, ..., U_t$ be the participants and let $S \in \mathbb{N}$ be the secret.

The dealer randomly chooses the polynomial $g(z) \in \mathbb{H}[z]$ of degree $(n-1)$, as in the Lagrange scheme, having constant term a_0 such that $Re(a_0) = S$. With q_r also as in **4.1**, $1 \leq r \leq n$, he then computes $d_r = g(q_r)$ and sends it secretly to participant U_r, for every r.

5.2 Reconstruction

Supposing, without loss of generality, that $U_1, ..., U_n$ wish to reconstruct S, they then pool their shares giving each U_i the sequence $(q_1, d_1), ..., (q_n, d_n)$ from which he uses the Newton interpolation method as follows:

Participant U_i obtains $h(z) = b_1 + p_2(z)b_2 + \cdots + p_n(z)b_n$ iteratively, first by constructing all the $p_j(z)$'s in order from 1 to n. Next he must compute the b_j's which are obtained, again from 1 to n, by the construction of 2.3 having $(q_1, d_1), ..., (q_n, d_n)$ as targets.

Lastly he computes $h(0) = g(0) = a_0$ and observes $Re(a_0) = S$.

5.3 Complexity Analysis

Evaluated in terms of total number of participants t for setup and the threshold value n for reconstruction. The same assumptions are held for the complexity of quaternion products, inversions and additions, all being $O(1)$.

The setup complexity is gain identical to the Lagrange case; thus being $O(t \log^2 t)$.

The reconstruction complexity is more involved. We must first adjudge the complexity of obtaining the sequence $p_1, ..., p_n$:

From the Lagrange case analysis (**2.3.2**) we already know the complexity of computing $p_{j+1}(z)$, assuming we have $p_j(z)$, to be $O(j)$; and doing this for each j, $(n-1)$ times in total, gives a running complexity of $O(n^2)$.

Now we require the sequence $b_1, ..., b_n$. Once more assuming that we already have $b_1, ..., b_j$ we find the complexity of computing b_{j+1}.

By definition, $b_{j+1} = p_{j+1}(q_{j+1})^{-1}(d_{j+1} - b_0 - p_1(q_{j+1})b_1 - \cdots - p_j(q_{j+1})b_j)$ so begin by having U_i compute the sequence $p_1, ..., p_n$ which from the Lagrange case of **2.3.2** is $O(n^2)$.

We also know that computing $p_j(q_k)$ is of complexity $O(j)$, for $j \leq k$, and as $p_j(q_k) = 0$ for $j > k$ we have complexity $O(1) + 2O(2) + 3O(3) + \cdots + nO(n)$ to account for every case of j from 1 to n. This results in a complexity of $O(n^3)$.

Hence, possessing all the $p_j(q_k)$, computing b_{j+1} requires further j multiplications followed by $(j+1)$ subtractions and one inversion followed by one last multiplication: this gives $O(j) + O(j) + O(1) + O(1) = O(j)$ complexity. As

this must be done for $1 \leq j \leq (n-1)$ we have $O(1)+O(2)+\cdots+O(n-1) = O(n^2)$ complexity.

Since computing all the $p_j(q_k)$'s is only additively followed by the rest of the b_js' computations we have running complexity $O(n^3) + O(n^2) = O(n^3)$.

Lastly, U_i computes $g(0) = S$, i.e., $S = p_1(0)b_1 + \cdots + p_n(0)b_n$. Having the sequence of $p_1, ..., p_n$ from before now allows a complexity of only $O(j)$ for computing $p_j(0)$; indeed by the j-th step we already have $p_{j-1}(0)$ and computing $p_j(0)$ only requires one more multiplication. Including the n subsequent multiplications and $(n-1)$ additions to complete S we have the total complexity of computing S as $O(1)+\cdots+O(n)+nO(1)+(n-1)O(1) = O(n^2)$.

As these operations of computing S, again, additively follow the previous, we have a final reconstruction complexity of $O(n^3) + O(n^2) = O(n^3)$.

5.4 Security Analysis

Suppose maximally, and without loss of generality, that participants $U_1, ..., U_{n-1}$ pool their shares to obtain the secret, S.

They may easily obtain the sequences $p_1(z), ..., p_n(z)$ and $b_1, ..., b_{n-1}$ then set

up the equation

$$\begin{aligned} S = g(0) &= p_1(0)b_1 + p_2(0)b_2 + \cdots + p_n(0)b_n \\ &= C_{n-1} + c_n \cdot b_n \end{aligned}$$

where C_{n-1} and c_n are known constants.

However, b_n remains unknown as we see in the equation

$$b_n = p_n(q_n)^{-1}(c_n - b_0 - p_1(q_n)b_1 - \cdots - p_{n-1}(q_n)b_{n-1})$$

noting that the (effectively) random value c_n is itself unknown to these participants, of indeterminate bounds, and so the secret S is shielded from discovery.

It is again possible to try and sidestep this difficulty by setting up and inverting the matrix equation

$$\begin{bmatrix} 1 & q_1 & q_1^2 & \dots & q_1^n \\ 1 & q_2 & q_2^2 & \dots & q_2^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & q_n & q_n^2 & \dots & q_n^n \end{bmatrix} \begin{bmatrix} S \\ h_1 \\ \vdots \\ h_{n-1} \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix}$$

like the Lagrange case, though just as there we know that S is safe behind d_n, and there are N^4 choices for any given h_j, where the bound N may be

chosen as large as desired.

One less participant colluding likewise increases the necessary choices of coefficients or shares by one. Again the attackers must contend with not knowing N precisely.

Chapter 6

The Free Quaternion Polynomial Scheme: First Form

This chapter details a novel secret sharing scheme and uses free quaternion polynomials.

6.1 Setup

For this $(n+1, t)$ secret sharing scheme let $S \in \mathbb{N}$ be the secret and $U_1, ..., U_t$ be the participants. The dealer must first define an n-th degree polynomial $L(z) \in \mathbb{H}\langle z \rangle$ of the form

$$L(z) = a_0 + a_1 z a_1 + a_2 z a_2 z a_2 + \cdots + a_n z a_n \cdots a_n z a_n$$

chosen randomly, except that $\|a_n\| = S$ is the secret. The random choice of polynomial coefficients is as follows: For $0 \leq m \leq n$ choose $a_m = a_{m0} + a_{m1}\mathbf{i} + a_{m2}\mathbf{j} + a_{m3}\mathbf{k}$ where each a_{ml}, $0 \leq l \leq 3$, is selected randomly and uniformly from the set $\{1, ..., N\}$ for some fixed $N \in \mathbb{N}$ taken as large as desired.

The dealer then generates the shares for participants $U_1, ..., U_t$ as follows: Selecting some $q \in \mathbb{H} \setminus \mathbb{R}$ such that $\|q\| = 1$, he computes, for $1 \leq j \leq t$, the values $y_j = L(jq)$. The dealer sends (j, y_j) to participant U_j, and all participants know the value n.

6.2 Reconstruction

Without loss of generality, suppose $U_1, ..., U_{n+1}$ wish to reconstruct the secret.

Each U_j, $1 \leq j \leq n+1$, possesses and pools

$$\begin{aligned} y_j &= a_0 + a_1 jqa_1 + a_2 jqa_2 jqa_2 + \cdots + a_n jqa_n \cdots a_n jqa_n \\ &= a_0 + ja_1 qa_1 + j^2 a_2 qa_2 qa_2 + \cdots + j^n a_n qa_n \cdots a_n qa_n. \end{aligned}$$

For simplicity denote $A_i = a_i qa_i \cdots a_i qa_i$ as the ith term of the above sum, which then becomes $y_j = A_0 + jA_1 + j^2 A_2 + \cdots + j^n A_n$.

With $U_1, ..., U_{n+1}$ contributing $y_1, ..., y_{n+1}$ they may assemble the matrix

equation

$$\begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & 2 & 2^2 & \dots & 2^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & (n+1) & (n+1)^2 & \dots & (n+1)^n \end{bmatrix} \begin{bmatrix} A_0 \\ A_1 \\ A_2 \\ \vdots \\ A_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n+1} \end{bmatrix}$$

.

Note that the $(n+1)$-by-$(n+1)$ matrix on the left is a Vandermonde matrix, which ensures its invertibility. So this equation of the form $VA = Y$ may be transformed to $A = V^{-1}Y$, and then U_j may determine $A_n = Q$ for some $Q \in \mathbb{H}$. As $\|A_n\| = \|a_n q a_n \cdots a_n q a_n\| = \|a_n\|^n \|q\|^n = \|a_n\|^n$, participant U_j can then compute $S = \|a_n\| = \|Q\|^{1/n}$.

As an important aside, though we chose participants $U_1, ..., U_{n+1}$ any $(n+1)$ participants would do just as well, since the only requirement for assembling a Vandermonde matrix is that each row progress by a different base, which it will by choice of shares.

6.3 Complexity Analysis

The two phases of setup and reconstruction will be evaluated separately, and in terms of the threshold value $(n+1)$ (actually, just n).

6.3.1 Setup Complexity

First suppose we have a random oracle to supply the four components of each a_m, for every m (being $a_{m0}, a_{m1}, a_{m2}, a_{m3}$), so that each choice is made with a complexity of $O(1)$. There are $4(n+1)$ choices to be made in assembling $L(z)$, giving a complexity of $O(n)$.

Again assuming that quaternion multiplication is of $O(1)$ complexity and addition $O(1)$, we must then find the computational complexity of calculating $y_j = L(jq)$.

Allow the dealer to find y_j as follows: He first computes the sequence $A_0, ..., A_n$ which, as $A_i = a_i q a_i \cdots a_i q a_i$ will require $2i$ multiplications, and so gives complexity $O(2)+O(4)+\cdots+O(2n)+O(n) = O(n^2)$, including the initial random choice of coefficients.

In parallel he obtains the sequence $j, j^2, ..., j^n$ which requires $(n-1)$ multiplication operations and so is of $O(n)$ complexity. Afterwards to complete the given $j^i A_i$ he must multiply j^i by A_i which occurs n times having complexity $O(n)$. Then these A_i must be added together, again having $O(n)$.

This gives us a running complexity of $O(n^2) + O(n) + O(n) + O(n) = O(n^2)$ to compute y_j, and as this must be done for every $1 \leq j \leq t$, we have a total of $tO(n^2) = O(n^3)$ complexity for the setup process, noting t that is fixed, and so $O(n)$.

6.3.2 Reconstruction Complexity

For a given participant U_i they begin by constructing and inverting the matrix V, which being $(n+1)$-by-$(n+1)$ will take $O(n^3)$ operations.

They then need only take the dot-product of the vector Y^T by the last row of V^{-1} to obtain Q, which takes $nO(1) + nO(1) = O(n)$ operations.

To find the complexity of computing the n-th root of Q, first see that $\|Q\|$ is $O(n)$ as Q is fixed. Hence there are $O(n)$ natural numbers less than Q, and even at its most inefficient, checking which is the least that, to the n-th power, returns an integer will yield Q, the secret. As raising to the n-th power takes n multiplications, the complexity of this n-th root computation is $nO(n) = O(n^2)$.

The final reconstruction complexity is then $O(n^3) + O(n) + O(n^2) = O(n^3)$.

6.4 Security Analysis

This scheme is secure because at least $(n+1)$ shares are required for the matrix equation to compute $\|a_n\|$. All that could otherwise be derived would be $\|a_n\|$ in terms of other, unknown and randomly chosen values, where the bound of these values' norms may be taken as large as desired.

As is the case with the previous schemes, attackers may set up the same

equation as legitimate reconstructors and make their best guess at the shares or secret polynomial coefficients, both random.

For the scheme in its first form, there is no dependence relation among the $A_0, ..., A_n$ the attackers might take advantage of. The price paid for this is in computational complexity.

Chapter 7

The Free Quaternion Polynomial Scheme: Second Form

This chapter details a variation upon the previous free quaternion polynomial scheme. Its construction is much the same, with the exception being how the secret polynomial coefficients are chosen.

7.1 Setup

For this $(n+1, t)$ secret sharing scheme let $S \in \mathbb{N}$ be the secret and $U_1, ..., U_t$ be the participants. The dealer must first define an n-th degree polynomial

$L(z) \in \mathbb{H}\langle z\rangle$ of the form

$$L(z) = a_0 + a_0 z a_1 + a_0 z a_1 z a_2 + \cdots + a_0 z a_1 \cdots a_{n-1} z a_n$$

chosen randomly, except that $\|a_n\| = S$ is the secret. The random choice of polynomial coefficients is as before: For $0 \leq m \leq n$ choose $a_m = a_{m0} + a_{m1}\mathbf{i} + a_{m2}\mathbf{j} + a_{m3}\mathbf{k}$ where each a_{ml}, $1 \leq l \leq 3$, is selected randomly and uniformly from the set $\{1, ..., N\}$ for some fixed $N \in \mathbb{N}$ taken as large as desired.

The dealer then generates the shares for participants $U_1, ..., U_t$: Selecting some $q \in \mathbb{H} \setminus \mathbb{R}$ such that $\|q\| = 1$, he computes, for $1 \leq j \leq t$, the values $y_j = L(jq)$. The dealer sends (j, y_j) to participant U_j, and all participants know the value n.

7.2 Reconstruction

Without loss of generality, suppose $U_1, ..., U_{n+1}$ wish to reconstruct the secret.

Each U_j possesses and pools

$$y_j = a_0 + j a_0 q a_1 + j^2 a_0 q a_1 q a_2 + \cdots + j^n a_0 q a_1 \cdots a_{n-1} q a_n.$$

For simplicity denote $A_i = a_0 q a_1 \cdots a_{i-1} q a_i$ as the ith term of the above sum, which then becomes $y_j = A_0 + jA_1 + j^2 A_2 + \cdots + j^n A_n$.

With $U_1, ..., U_{n+1}$ contributing $y_1, ..., y_{n+1}$ they may assemble the matrix equation

$$\begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & 2 & 2^2 & \dots & 2^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & (n+1) & (n+1)^2 & \dots & (n+1)^n \end{bmatrix} \begin{bmatrix} A_0 \\ A_1 \\ A_2 \\ \vdots \\ A_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n+1} \end{bmatrix}$$

noting once again that the $(n+1)$-by-$(n+1)$ matrix on the left is a Vandermonde matrix. Transforming $VA = Y$ to $A = V^{-1}Y$, each U_j may then determine $A_n = Q$ and $A_{n-1} = Q'$ for some $Q, Q' \in \mathbb{H}$. As it's also true that $\|A_n\| = \|a_0 q a_1 \cdots a_{n-1} q a_n\| = \|a_0\|\|a_1\| \cdots \|a_n\|$, and $\|A_{n-1}\| = \|a_0\|\|a_1\| \cdots \|a_{n-1}\|$ then U_j can compute $S = \|a_n\| = \|Q\|/\|Q'\|$.

7.3 Complexity Analysis

Again separated into setup and reconstruction phases, in terms of n.

7.3.1 Setup Complexity

First suppose we have a random oracle to supply the four components of each a_m for every m (being $a_{m0}, a_{m1}, a_{m2}, a_{m3}$), so that each choice is made with a complexity of $O(1)$. There are $4(n+1)$ choices to be made in assembling $L(z)$, giving a complexity of $O(n)$.

We must then find the computational complexity of calculating $y_j = L(jq)$.

Allow the dealer to find y_j as follows: He computes the sequence $A_0, A_1, ..., A_n$ which, as $A_i = a_0 q a_1 \cdots a_{i-1} q a_i$ will require only 2 more multiplications for a given i from $(i-1)$, since $A_i = A_{i-1} q a_i$. Including the initial random choice of coefficients, will give complexity $O(n) + 2nO(1) = O(n)$.

In parallel he obtains the sequence $j, j^2, ..., j^n$ which requires $(n-1)$ multiplication operations and so is of $O(n)$ complexity. Afterwards to complete the given $j^i A_i$ he must multiply j^i by A_i which occurs n times having complexity $O(n)$. Then these A_i must be added together, again having $O(n)$.

This gives us a running complexity of $O(n)+O(n)+O(n) = O(n)$ to compute y_j, and as this must be done for every $1 \leq j \leq t$, with t being $O(n)$, gives a total of $O(n^2)$ complexity for the setup process.

7.3.2 Reconstruction Complexity

For a given participant U_i they begin by constructing and inverting the matrix V, which being $(n+1)$-by-$(n+1)$ will take $O(n^3)$ operations.

They then take the dot-product of the vector Y^T by the last two rows of V^{-1} to obtain Q and Q', which takes $2nO(1) = O(n)$ operations.

Lastly U_i need only divide Q by Q' which takes $O(1)$ steps. Hence the total complexity of reconstruction is $O(n^3) + O(n) + O(1) = O(n^3)$.

7.4 Security Analysis

This scheme is secure because at least $(n+1)$ shares are required for the matrix equation to compute $\|a_n\|$. All that could otherwise be derived would be $\|a_n\|$ in terms of other, unknown and randomly chosen values, where the bound of these values' norms may be taken as large as desired.

As is the case with the previous schemes, attackers may set up the same equation as legitimate reconstructors and make their best guess at shares or secret polynomial coefficients, both random.

For the scheme in its second form, there is a limited dependence relation among the $A_0, ..., A_n$ the attackers might take advantage of. The benefit is in less computational complexity than the first form.

Chapter 8

Future Work

At this point we might ask what loose ends may be tied up in any subsequent labors in the various areas of this thesis.

To begin with, it was stated in Chapter 3 that our formulae were achieved for left skew polynomials; the right variation is perfectly parallel, but perhaps in the relation of both left and right something interesting may be found. (Right skew polynomials merely arrange the variables to the right of coefficients and use the equivalent twist rule of $za = \sigma(a)z$.)

There is also, and perhaps with greater significance, a more general form of skew polynomial than presented - in its "right" presentation this consists of extending $\mathbf{D}[z;\sigma]$ to $\mathbf{D}[z;\sigma,\delta]$ where $\delta : \mathbf{D} \to \mathbf{D}$ is called a "σ-derivation". This δ satisfies, for all $a, b \in \mathbf{D}$, $\delta(a+b) = \delta(a) + \delta(b)$ and also $\delta(ab) = \delta(a)b + \sigma(a)\delta(b)$.

The σ-derivation enters in by a further devolving of the (right) twist rule, which governs the commutation of variables with coefficients. Now we have, for all $a \in \mathbf{D}$, $za = \sigma(a)z + \delta(a)$.

The addition of the "$\delta(a)$" with no variable is a subtle but highly influential change, as one can no simply commute the variable in higher powers en masse with coefficients. Lower degree terms will be generated at every commutation, which themselves must then be dealt with no less.

E.g., $z^4a = z^3(\sigma(a)z + \delta(a)) = z^3\sigma(a)z + z^3\delta(a)$, etc.

Now, one can easily define the left backward shift operator in this context, though to attempt the same proofs as ours (by mere definitions, at least) will thus increase their symbolic load to an unwholesome degree. As if the formula wasn't bad enough for twisted polynomials!

Nevertheless, it may still be done, and even if one cannot discover half-pleasant equations, recursive analysis may prove sufficient to demonstrate similar results.

It was also stated that there are many extant developments in secret sharing schemes that the schemes of this thesis make no use of. It is much to the point to see how the former might translate to the noncommutative setting. Any scheme of a given paper could be analyzed as to its potential for this.

The quaternion Newton and Lagrange schemes could also each be varied to use different sorts of noncommutative polynomials, and attempts made to

gauge the computational and security benefits of such a variation.

Moving more specifically to questions of security, if we might find actual effective ways of bounding quaternion and free quaternion polynomials we could tell more realistically just how difficult predicting some participant's share could be, and with better accuracy indicate the schemes' strength to attack.

Lastly, this thesis in no real way involves the concepts and issues of quantum computers. It remains unknown to your present author whether or not these categorically superior machines render the presented secret sharing schemes obsolete. If they do, it could still be of worth seeing whether or not any modifications may circumvent the schemes' inadequacies.

Bibliography

[1] Abdelraheem M.A. & Beelen P. & Bogdanov A. & Tischhauser E. Twisted Polynomials and Forgery Attacks on GCM. Eurocrypt 2015, Springer-Verlag.

[2] Abramek K.F. & Stoeck T. Application of th Polynomial Interpolation Method for Determining the Performance Characteristics of a Diesel Engine. Metrology and Measurement Systems, 2014, Vol. 21, No. 1, pp.157-168.

[3] Aristidou M. & Demetre A. A Note on Quaternion Rings over $\mathbb{Z}_p$. International Journal of Algebra, 2009, Vol. 3, No. 15, pp.725-728

[4] Binu V.P. & Sreekumar A. Efficient Multi Secret Sharing Scheme with Generalized Access Structure. International Journal of Computer Applications, 2014, Vo. 90, No. 12.

[5] Binu V.P. & Sreekumar A. Lossless Secret Image Sharing Schemes. arXiv.org, Cornell University, 2015.

[6] Blakley G. Safeguarding cryptographic keys. Proceedings of AFIPS National Computer Conference, 1979.

[7] Bolotnikov V. Confluent Vandermonde matrices, divided differences, and Lagrange-Hermite interpolation over quaternions. Communications in Algebra, 2017, Vol. 45, Iss. 2, pp.574-599.

[8] Bolotnikov V. Polynomial Interpolation Over Quaternions. arXiv.org, Cornell University, May 2014.

[9] Bolotnikov V. Zeros and Factorizations of Quaternion Polynomials: The Algorithmic Approach. arXiv.org, Cornell University, May 2015.

[10] Boucher D. & Ulmer F. Linear codes using skew polynomials with automorphisms and derivations. Designs, Codes and Cryptography, Springer, 2014, Vol. 70, Iss. 3, pp.405-431.

[11] Chen L. & Gollmann D. & Mitchell C.J. & Wild P. Secret Sharing with Reusable Polynomials. Information Security and Privacy, Springer, 1997, pp.183-193.

[12] Cooke J.M. & McGhee R.B. & Pratt D.R. & Zyda M.J. NPSNET: Flight Simulation Dynamic Modeling Using Quaternions. Presence, Vol. 1, No. 4, 1994, pp.404-420.

[13] Cramer R. & Damgård I. & Maurer U. General Secure Multi-Party Computation from any Linear Secret-Sharing Scheme. International Association for Cryptologic Research, Eurocrypt 2000.

[14] Cramer R. & Damgård I. Multiparty Computation, an Introduction. (Chapter from) Contemporary Cryptology, 2006, pp.41-87.

[15] Davie A.M. & Stothers A.J. Improved bound for complexity of matrix multiplication. Proceedings of the Royal Society of Edinburgh, 2013, Vol. 143A, pp.351-370.

[16] Eilenberg S. & Niven I. The "Fundamental Theorem of Algebra" for Quaternions. Bulletin of the American Mathematical Society, 1944, Vol. 50, pp.246-248.

[17] Endurthi A. & Tentu A.N. & Venkaiah V.Ch. Reusable Multi-Stage Multi-Secret Sharing Scheme Based on Asmuth-Bloom Sequence. International Journal of Computer Applications, ICCCMIT, 2014.

[18] Erić A. Polynomial Interpolation Problem for Skew Polynomials. Applicable Analysis and Dicrete Mathematics, 2007, Vol. 1, pp.403-414.

[19] Familton J.C. Quaternions: A History of Complex Noncommutative Rotation Groups in Theoretical Physics. Columbia University, 2015.

[20] Finkelstein D. Foundations of Quaternion Quantum Mechanics. Journal of Mathematical Physics, Vol. 3, 1962.

[21] Franklin M. & Yung M. Communication complexity of secure computation. Proceedings of the 24th annual ACM symposium on the Theory of computing, pp.699-710.

[22] Hardjono T. & Seberry J. & Zheng Y. Reusing Shares in Secret Sharing Schemes. Computer Journal, 1994, Vol. 37, Iss. 3, pp.199-205.

[23] Huang L. & Wang Q-W. & Zhang Y. The Moore-Penrose inverses of matrices over quaternion polynomial rings. Linear Algebra and Its Applications, Elsevier, 2015, Vol. 475, pp.45-61.

[24] Karatsuba A. & Ofman Y. Multiplication of Many-Digital Numbers by Automatic Computers. Physics-Doklady, 1963, Vol. 7, pp.595-596.

[25] Kschischang F.R. & Liu S. & Mangianello F. Kötter interpolation in skew polynomial rings. Designs, Codes and Cryptography, Springer, 2014, Vol. 72, Iss. 3, pp.593-608.

[26] Lam T.Y. A general theory of Vandermonde. Expositiones Mathematicae, Bibliogaphisches-Institut, 1986, pp.193-215.

[27] Liu W. & Liu Z. & Xu Y. & Zhang X. Quaternion-based worst case Constrained beamformer based on electromagnetic vector-sensor arrays. ICASSP, IEEE, 2013.

[28] Menezes A.J. & van Oorschot P.C. & Vanstone S.A. Handbook of Applied Cryptography. CRC Press, 1996.

[29] Niven I. Equations in Quaternions. The American Mathematical Monthly, 1941, Vol. 48, No. 10, pp.654-651.

[30] Pang L. & Wang Y. A Secure and Efficient (t, n) Multi-Secret Sharing

Scheme. Wuhan University Journal of Natural Sciences, 2005, Vol. 10, No. 1, pp.191-194.

[31] Rodman L. Topics in Quaternion Linear Algebra. Princeton Series in Applied Mathematics, 2014.

[32] Shamir A. How to Share a Secret. Communications of the ACM, 1979, Vol. 22, No. 11, pp.612-613.

[33] Stinson D. & Wei R. Bibliography on Secret Sharing Schemes. 1998, https://cs.uwaterloo.ca/∼dstinson/ssbib.html.

[34] Sui Y. & Wang K. & Zou X. A Multiple Secret Sharing Scheme based on Matrix Projection. 33rd Annual IEEE International Computer Software and Applications Conference, 2009.

[35] Turner J. Skew-Symmetric Algorithm for Inverting NxN Quaternion Equations Using NxN Real Variable Matrix Arithmetic. The Journal of the Astronautical Sciences, 2009, Vol. 57, Nos. 1 & 2, pp.275-28.

[36] Vince J. Quaternions for Computer Graphics. Springer, 2011.

[37] Waring E. Problems concerning interpolations. Philosophical Transactions of the Royal Society of London, 1779, Vol. 69, pp.59-67.

[38] Zhang F. Quaternions and Matrices of Quaternions. Linear Algebra and Its Applications, Elsevier, 1997, Vol. 251, pp.21-57.

[39] Ziegler M. Quasi-optimal Arithmetic for Quaternion Polynomials. International Symposium on Algorithms and Computation, Springer, 2003, pp.705-715.