UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Kätryn Varkel

Interpolation of Satellite Orbital States in the Context of ShareSat

Bachelor's Thesis (9 ECTS)

Supervisors:
Karl Hannes Veskus, MSc
Sandhra-Mirella Valdma, MSc
Hendrik Eerikson, MSc

Tartu 2026

Interpolation of Satellite Orbital States in the Context of ShareSat

Abstract:

Accurate conjunction analysis requires estimating satellite positions and velocities between discrete trajectory samples. Since high-precision trajectory data can be sensitive, this thesis adds Lagrange interpolation to the ShareSat workflow using secure multi-party computation, allowing trajectory estimation without revealing raw input data. The implementation was written in SecreC and evaluated using European Space Agency satellite trajectory data. Of the tested configurations, only six-point Lagrange interpolation satisfied the required precision limits for both position and velocity. The thesis concludes that Lagrange interpolation can support accurate and privacy-preserving trajectory estimation in collision analysis.

Keywords: Secure multi-party computation, satellite collision analysis, satellite trajectory interpolation, Lagrange interpolation, orbital data privacy

CERCS: P170 Computer science, numerical analysis, systems, control

Satelliidi orbiidiandmete interpoleerimine ShareSati kontekstis

Lühikokkuvõte:

Täpne kokkupõrkeanalüüs nõuab satelliitide asukohtade ja kiiruste hindamist diskreetsete trajektooripunktide vahel. Kuna kõrge täpsusega trajektooriandmed võivad olla tundlikud, käsitleb see lõputöö Lagrange'i interpolatsiooni rakendamist ShareSati töövoos turvalise ühisarvutuse abil, võimaldades trajektoore hinnata ilma toorandmeid avaldamata. Töö käigus loodud lahendus kirjutati SecreC programmeerimiskeeles seda hinnati Euroopa Kosmoseagentuuri satelliitide trajektooriandmeid kasutades. Testitud konfiguratsioonidest vastas ainult kuuepunktiline Lagrange'i interpolatsioon nõutud asukoha- ja kiirustäpsuse kriteeriumitele. Töö tulemused näitavad, et Lagrange'i interpolatsiooni saab kasutada orbiidiandmete hindamiseks viisil, mis säilitab privaatsust kokkupõrke tõenäosuse arvutamisel.

Võtmesõnad: Turvaline ühisarvutamine, satelliidi kokkupõrke analüüs, satelliidi trajektoori interpolatsioon, Lagrange'i interpolatsioon, orbiidiandmete privaatsus

CERCS: P170 Arvutiteadus, arvanalüüs, süsteemid, kontroll

Contents

1. Introduction

Satellites are essential for modern communication, internet connectivity, navigation, weather forecasting, security, economy, and science. At the same time, the Earth's orbital environment is becoming increasingly crowded. According to the European Space Agency's (ESA) Space Environment Statistics [1], last updated on 21 April 2026, about 25,920 satellites have been placed in Earth orbit, of which about 17,610 are still in space. ESA also estimates that there are about 1.2 million debris objects between 1 cm and 10 cm currently in orbit, in addition to even smaller fragments. Together with the rapid growth of commercial and government satellite constellations, this makes reliable collision-risk assessment increasingly important [2].

A single collision can create thousands of debris fragments that may remain in orbit for years or decades, increasing the risk of further collisions through the cascading effect known as Kessler Syndrome [2]. In 2009, two satellites from the United States and Russia collided, producing over 2,000 trackable fragments that continue to orbit the Earth today [3]. This event showed the need for effective space traffic management and reliable conjunction analysis.

Accurate collision prediction depends on precise orbital data, but satellite operators may be unwilling or unable to share detailed trajectory information because of commercial, competitive, or national security concerns. ShareSat addresses this problem by exploring secure computation for privacy-preserving satellite conjunction analysis, allowing operators to jointly analyse collision risk without revealing sensitive orbital data [4, 5].

An important part of collision probability assessment is trajectory estimation. Satellite position data is usually available only at discrete time points; therefore interpolation is needed to estimate satellite positions between available measurements [6]. In earlier development phase of ShareSat, linear interpolation was implemented as a baseline to validate the core privacy-preserving concept [5]. While this approach is useful, improving the accuracy of interpolation is a natural next step.

The goal of this thesis is to improve the trajectory estimation part of ShareSat by replacing the initial interpolation approach with Lagrange interpolation. The thesis investigates whether this method can provide more accurate trajectory estimates while still fitting into the privacy-preserving computation model used by ShareSat.

This thesis is structured as follows: Chapter 2 introduces satellite collision avoidance including conjunction analysis and time of closest approach. Chapter 3 gives the necessary background on interpolation and its use in satellite trajectory prediction. Chapter 4 presents secure computation

and the ShareSat framework. Chapter 5 describes the implementation, verification, and benchmarking setup. Chapter 6 presents the accuracy and performance results. Chapter 7 concludes the thesis and outlines possible future work. The appendices include the relevant code and additional benchmark implementation details.

The assistance of ChatGPT[1] by OpenAI, using the language models GPT-5.5 and GPT-5.4, was used for grammar checking, formatting and text structuring. No trajectory data, source code, or confidential project data was processed using AI tools.

[1] https://chat.openai.com

2. Satellite Collision Avoidance

Satellite collision avoidance is a key part of maintaining safe operations in an increasingly crowded orbital environment. In this context, even small errors in trajectory estimation can affect how potential close approaches are assessed. This chapter provides the necessary background for understanding the problem setting.

2.1 Orbits and Trajectories

The theoretical background in this section follows Curtis' explanation of the two-body problem in *Orbital Mechanics for Engineering Students (Second Edition)* [7].

Satellites move in space according to the laws of orbital mechanics, which describe how objects travel under the influence of gravity. A trajectory refers to the path that a spacecraft follows through space over time, determined by the initial position and velocity [8]. In the simplest case, where only the gravitational attraction between the satellite and the central body is considered, the resulting trajectory can take one of several standard orbital shapes. As shown in Figure 1, the central body sits at the shared focus of all trajectories and depending on the eccentricity, the orbit can take the form of a circular, elliptical, parabolic, or hyperbolic orbit.

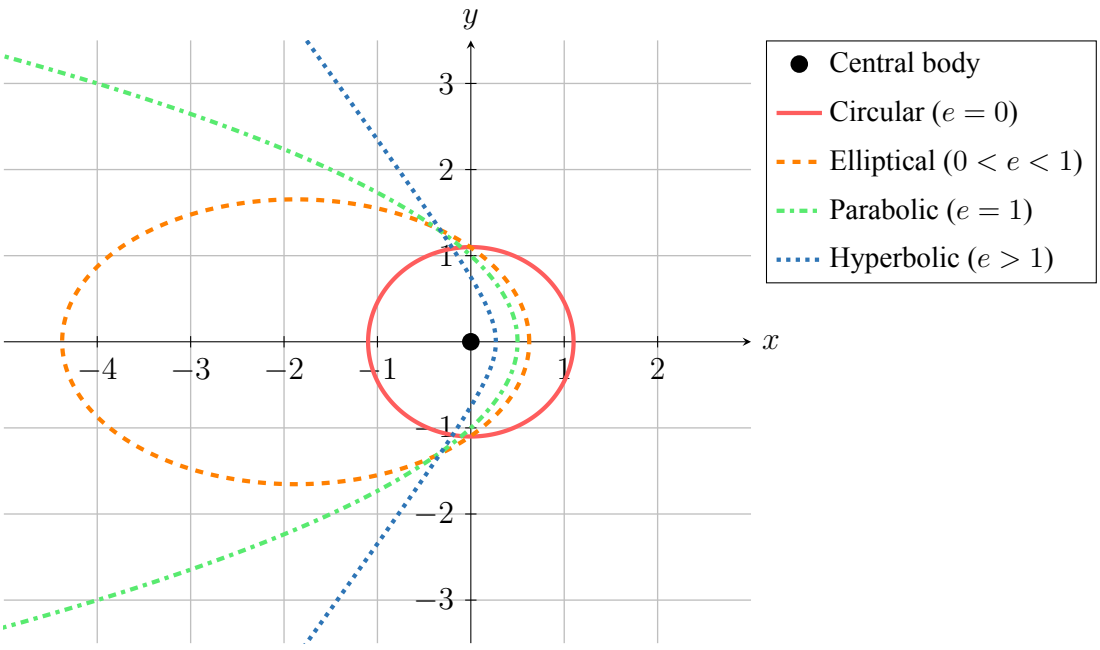

Figure 1. Eccentricity defines the shape and type of an orbit

The following discussion of orbital elements and state vector representations follows Curtis' presentation of three-dimensional orbits [9].

A common approach to representing a satellite's orbit is to use a set of orbital elements, which define the size, shape, and orientation of the trajectory, as well as the satellite's location along it at a given time. Another way is to represent the path using a state vector consisting of position and velocity. Unlike orbital elements, a state vector describes the satellite's position and velocity in discrete points rather than the orbit as a whole, although it can be used to reconstruct the path.

Together, these descriptions make it possible not only to represent where the satellite is at a given moment but also to estimate where it will be later. The prediction of future position and velocity is referred to as orbit propagation. In simple cases, it can be derived from ideal two-body motion, while in practical applications it often has to account for additional influences such as atmospheric drag or gravitational perturbations.

2.2 Conjunction Analysis

A satellite conjunction refers to an event in which two orbiting objects pass near each other in space. Conjunction analysis is the process of identifying such close approaches, determining their timing and geometry, and assessing whether they may pose a collision risk [10]. Since such events must be identified and evaluated in advance, accurate trajectory information is essential for collision probability assessment.

Orbital data are collected in standardised formats for a wide range of purposes, including conjunction analysis and collision probability assessment. One such format is the Orbit Ephemeris Message (OEM), which describes a satellite's orbit by providing its position, velocity, and other relevant information at a specific epoch [11]. Because it provides detailed trajectory data with high precision, it is useful for conjunction analysis and collision probability calculations. An important step in this process is to determine the Time of Closest Approach [5].

2.3 Time of Closest Approach

In this chapter, the Time of Closest Approach (TCA) is defined according to the internal project document [12]. The TCA represents the moment when the distance between two orbiting objects is at its smallest. Accurately determining this moment is critical because the satellite states at that time form the basis for the subsequent collision probability calculation. Since there is no single standardised process for TCA determination, this section presents the approach used in this thesis, as illustrated in Figure 2 [12].

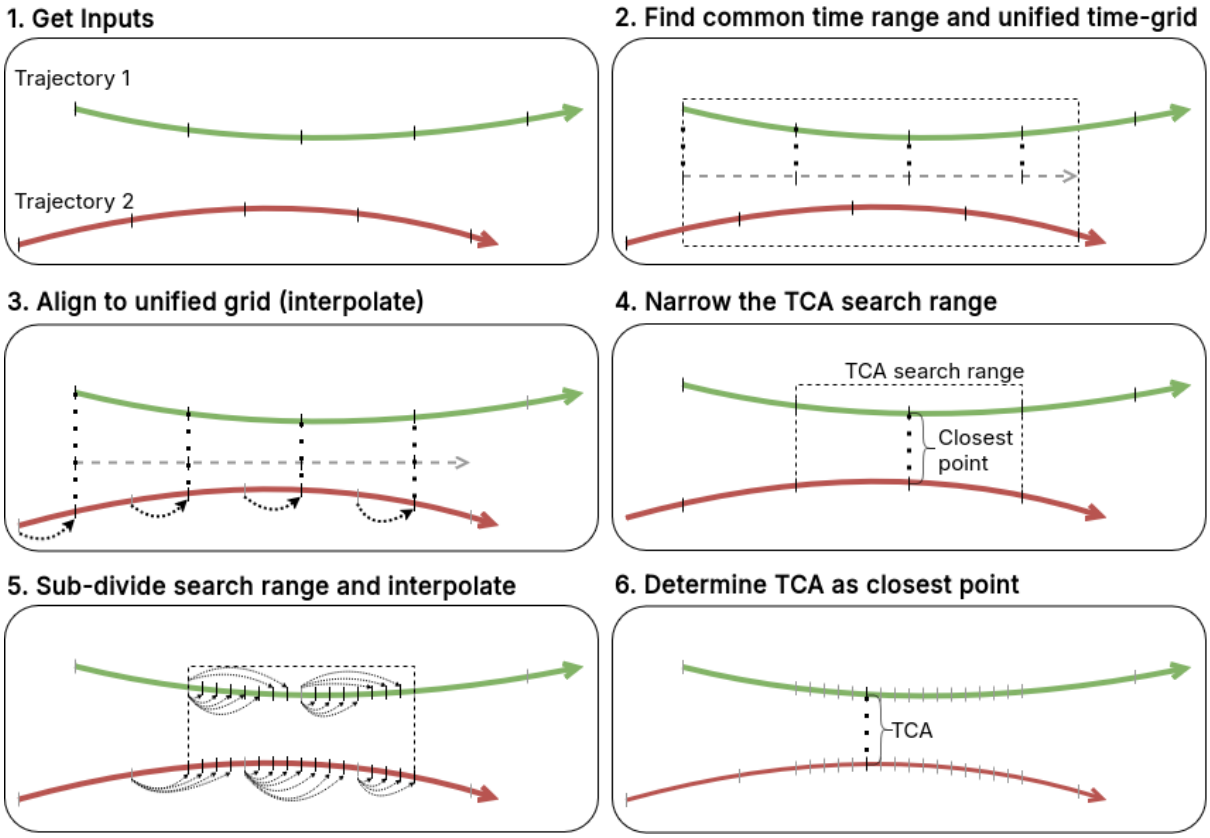

Figure 2. Workflow for determining TCA between two trajectories using a unified time grid and interpolation, adapted from [12]

Satellite positions and velocities are measured at specific time intervals, and interpolation is used to estimate the satellite state between these points. Since the two input trajectories may contain different time spans and with varying intervals between measuring points, a common time-grid must be established. As shown in Figure 2, the TCA determination process consists of the following steps [12]:

Step 1. The process starts with the initial orbital data, which consists of OEM files (see Section 2.2).

Step 2. A common time range is identified for the analysis, and a unified time-grid is created.

Step 3. The positions and velocities of both spacecraft are interpolated at each time point on this unified time-grid (e.g. establishing a 1-minute baseline if input OEM data is sampled at 1-minute intervals).

Step 4. The point on the unified time-grid where the two spacecraft are closest is found, and the TCA search range is defined as the interval between the adjacent time points.

Step 5. The search range is then divided into evenly spaced candidate points, where position and velocity are interpolated again. This involves approximately 100 points per interval ensuring time precision smaller than one second.

Step 6. The TCA is selected as the candidate time point with the smallest distance between the two objects. The corresponding states of both spacecraft are then stored for further use.

This workflow shows where interpolation is needed in conjunction analysis. It is used to both align the trajectories on a common time grid and to refine the TCA search within the selected interval. Since the accuracy of these interpolated states affects the final closest-approach estimate, the next chapters focus on the implementation and evaluation of Lagrange interpolation.

3. Interpolation in Satellite Trajectory Prediction

In satellite trajectory prediction, accurate estimation of a satellite's state between known data points is essential for collision probability assessment [6]. This is typically achieved through interpolation, a mathematical technique that constructs a continuous function by predicting unknown values within the range of discrete data points [13]. There are many interpolation methods, each with varying levels of accuracy, computational complexity, and suitability for orbital dynamics.

Figure 3 compares interpolation methods on an elliptical orbit arc [7]. Linear (two-point Lagrange) interpolation approximates the trajectory with a straight line and therefore has the largest error. This illustrates why a higher-order interpolation method is useful for satellite trajectory prediction, where the motion is curved rather than linear.

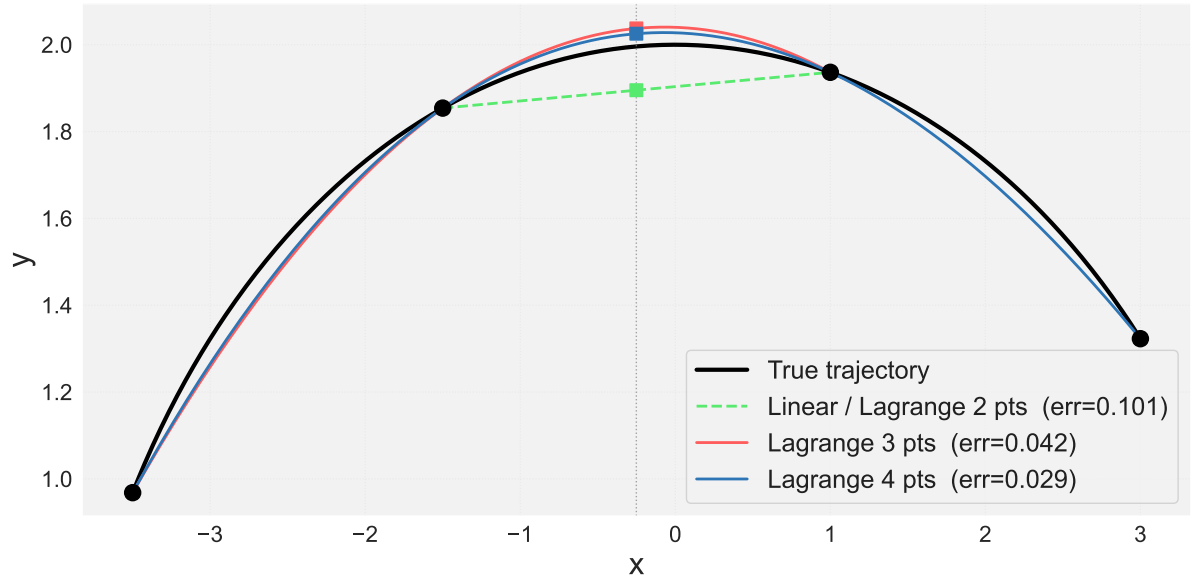

Figure 3. Higher-order interpolation follows the curved orbit more accurately than linear interpolation

Although linear interpolation is easy to implement and computationally efficient, this method is mainly suitable when the relationship between components is also linear [13]. In the context of satellite motion, orbital trajectories are affected by gravitational forces, atmospheric drag, and other perturbations [6]. As a result, linear interpolation often fails to capture the true shape of the trajectory, leading to significant prediction errors.

To address the limitations of linear interpolation, polynomial interpolation methods are used. Unlike linear interpolation, polynomial interpolation can capture the curved and non-linear

nature of satellite motion more accurately [6]. One widely used polynomial interpolation method is Lagrange interpolation [6, 14, 15].

3.1 Lagrange Interpolation

The mathematical formulation in this subsection follows the presentation of Lagrange interpolation given in the Numerical Methods lecture notes from the University of Tartu [13]. The Lagrange interpolation constructs a polynomial of degree n through $n+1$ distinct data points. Given the set of points $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$, the Lagrange interpolation polynomial $P(x)$ is defined as

$$P(x) = \sum_{i=0}^{n} y_i L_i(x) \tag{1}$$

where $L_i(x)$ are the Lagrange basis polynomials, defined as:

$$L_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j} \tag{2}$$

Each basis polynomial is constructed so that it is equal to 1 at its own interpolation point and 0 at all others, that is,

$$L_i(x_j) = \begin{cases} 1, & i = j, \\ 0, & i \neq j. \end{cases}$$

This ensures that $P(x)$ satisfies $P(x_i) = y_i$ for all i, so the polynomial passes through all given points exactly.

As shown by Equations (1) and (2), Lagrange interpolation reconstructs intermediate values as a weighted combination of all known samples, which makes it more suitable than linear interpolation for representing the curved behaviour of satellite trajectories [6].

3.2 Limitations of Lagrange Interpolation

While Lagrange offers advantages in accuracy over linear methods, it has several limitations that must be considered in practical applications. The computational cost of the method increases with the number of interpolation points, as more basis polynomials need to be evaluated [16].

For n interpolation points, the full interpolation polynomial $P(x)$ contains n Lagrange basis polynomials. Each basis polynomial contains approximately $n-1$ factors. When the formula is evaluated directly, this gives $2n(n-1)$ subtractions, $n(n-1)$ divisions, and $n(n-1)+n$ multiplications, which are all $O(n^2)$. The final addition of the terms requires $O(n)$ additions, but the overall cost is determined by the $O(n^2)$ operations.

This is important in secure computation, where division is significantly slower than multiplication, and multiplication is significantly slower than addition or subtraction [17]. As a result, using more interpolation points can improve accuracy, but it also increases the runtime cost.

Furthermore, when one or more new known points are added, the basic Lagrange polynomial has to be constructed again from the beginning, since the previously constructed basis polynomials cannot be directly reused [16].

Additionally, Runge's phenomenon, discovered by Carl David Tolmé Runge [18], shows that high-degree polynomials exhibit large oscillations at the edges of the interpolation interval, even for smooth functions. This shows that higher degrees don't always improve accuracy. In practical use, this limitation can be reduced by keeping the target interpolation time close to the centre of the selected interpolation interval and by avoiding unnecessarily high-degree polynomials. In this thesis, these limitations are acceptable because interpolation is applied only over short intervals with a small number of points, making Lagrange interpolation sufficiently accurate and simple.

4. Secure Computation

Collision probability assessment requires accurate trajectory estimation, which was addressed in Chapter 3 using interpolation techniques. In practice, the data used for conjunction analysis can be sensitive for operational, security, or commercial reasons [4]. Secure computation offers a solution for collaborative collision analysis without revealing private inputs. Rivas describes it as a group of cryptographic methods that allow computations to be carried out on private data while limiting what is revealed during the process [19]. Depending on the method, the goal may be to protect the input data, intermediate values, or both, while still producing the required output. In the context of this thesis, secure computation techniques can be used to compute collision-related results without revealing the full trajectories of the involved satellite operators.

Secure computation can be implemented using different cryptographic approaches, such as homomorphic encryption, garbled circuits, and secret sharing [20]. In this thesis, the focus is on secure multi-party computation based on linear secret sharing.

4.1 Threat model

This section follows Pullonen-Raudvere's description of the threat model [21].

Secure computation protocols are specified under a threat model. This model sets the rules for what kind of attacker the system is expected to protect against and what the protocol must guarantee. In cryptography, this attacker is usually called an adversary and in more familiar terms, the adversary can be understood as a hacker or someone who has compromised part of the system. The hacker may want to observe network traffic, learn the internal state of some participants, or even take control of them; such participants are considered corrupted. If a participant is corrupted, the adversary can see everything that participant sees during the protocol and can potentially influence its actions.

Defining the threat model is important because secure computation does not protect against every possible attack. Instead, it provides security only under the assumptions defined by the threat model. For example, if one participant is compromised, the attacker may learn that organisation's local data, but the private inputs should still remain protected as long as the protocol's corruption assumptions are not violated. However, if the attacker can compromise enough participants, or if the attacker can break the secure communication channels between them, then the security guarantees may no longer hold. Therefore, the threat model makes clear which attacks are

handled by the protocol and which attacks are outside its scope. The main goal is to ensure that private inputs remain protected and that the computation reveals only what is intended.

4.1.1 Passive and Active Adversaries

Adversary behaviour is commonly categorised as passive or active. A passive (also called semi-honest or honest-but-curious) adversary follows the protocol specification but tries to learn extra information from the messages they see [21]. A protocol is said to provide passive security if, despite such corruption, the adversary learns nothing beyond what is implied by the prescribed output [19]. An active (malicious) adversary may deviate arbitrarily from the protocol in order to violate privacy or correctness, for example by sending malformed or inconsistent messages. An actively secure protocol aims to protect both privacy and correctness despite such behaviour, usually by adding verification steps that detect or prevent cheating [21].

4.1.2 Collusion and Corruption Thresholds

In secure multi-party computation, collusion and corruption thresholds are used to describe how much adversarial control a protocol can tolerate [21]. Many secure computation designs therefore state security in terms of a corruption threshold: security holds as long as the number of colluding (corrupted) parties stays below a specified limit, while exceeding that limit can enable reconstruction of secrets or manipulation of results. In practice, some systems assume non-collusion between specific servers operated by independent entities. Under this assumption, compromising one server is not enough, but if the servers do collude, the effective adversary gains more power and the security guarantees may no longer apply [19].

4.2 Secure Multi-Party Computation

Secure multi-party computation (MPC) is a subfield of cryptography and one of the main techniques used for secure computation [21]. It enables multiple parties to jointly compute a function over their private inputs without revealing those inputs to one another. Each participant learns only the final output of the computation, ensuring that sensitive data remains confidential throughout the process. This capability makes MPC particularly valuable in domains requiring collaboration between mutually distrusting but cooperating organisations, such as satellite operators assessing collision risks in space [5, 19].

The concept of MPC was introduced by Andrew Yao through the “Millionaires’ Problem”, which demonstrated how two parties could compare their wealth without disclosing their actual values [22]. Although the theoretical framework was established early, practical implementation

remained elusive for decades due to high computational overhead and complexity. The Danish sugar beet auction in 2008 is the first example of applying MPC to real-world systems [23].

The MPC approach used in this thesis is based on linear secret sharing schemes, because it offers a balanced trade-off between computational efficiency and security guarantees suitable for practical MPC applications [20].

4.3 Secret Sharing

Secret sharing [20] is a scheme which allows a party to share a confidential value x into n shares $x_1, \ldots, x_n$ and distribute them between $P_1, ..., P_n$. Each individual share reveals no information about x and only an authorised group of parties can reconstruct the original secret by combining their shares, any smaller or unauthorised subset learn nothing about x [24].

In a classical (t, n) threshold scheme, the secret is split into n shares such that any t shares are sufficient to reconstruct it, while any set of fewer than t shares provides no information about the secret [25]. This defines a security model in which the threshold t specifies the minimum number of parties needed to reconstruct the secret, ensuring resilience against collusion among up to $t - 1$ parties.

Secret sharing schemes come in various forms ranging in classical threshold constructions, such as Shamir's polynomial-based scheme [26] to simpler linear schemes. The next section focuses on additive secret sharing, a common building block in practical MPC protocols [20, 27].

4.3.1 Additive Secret Sharing

The additive secret sharing construction in this subsection follows the presentations of secret sharing-based secure computation in [21, 24].

Additive secret sharing is a secret sharing scheme in which a secret value x in a finite ring R is divided into n shares $x_1, \ldots, x_n \in R$ such that

$$x_1 + x_2 + ... + x_n = x.$$

We denote the set of shares of x as $[\![x]\!] = \{x_1, \ldots, x_n\}$. The first $n - 1$ shares are chosen uniformly at random from R, and the final share is computed as $x_n = x - \sum_{i=1}^{n-1} x_i$. This construction defines an (n, n) threshold scheme: all n shares are required to reconstruct the secret, while any set of $n - 1$ shares reveals no information about x.

Additive sharing allows addition and subtraction to be performed efficiently, as these operations can be computed locally on the shares. However, more complex operations generally require

communication between the parties. Given $[\![x]\!] = \{x_1, \ldots, x_n\}$ and $[\![y]\!] = \{y_1, \ldots, y_n\}$, the sharing of $x + y$ can be computed locally as

$$[\![x + y]\!] = \{x_1 + y_1, \ldots, x_n + y_n\}.$$

Similarly, multiplication by a public constant c can be computed locally as

$$c \cdot [\![x]\!] = \{c \cdot x_1, \ldots, c \cdot x_n\}.$$

However, multiplication of two private shared values requires a secure protocol. As a result, MPC performance is often dominated by the number of interaction rounds and network latency rather than local arithmetic cost [17].

4.3.2 Example of Secure Addition

The secure multi-party computation process can be illustrated using a scenario with two data owners, denoted as input parties A and B, three computing parties acting as servers, and an analyst interested in learning the sum of the data owners' values. The following steps show how the two inputs are split into additive shares, processed by the computing parties, and reconstructed by the analyst over the finite ring $\mathbb{Z}_{100}$ [27]:

Step 1: Input party A has input $x = 25$. They sample two random numbers, say $x_1 = 57$ and $x_2 = 13$, and compute the third share as $x_3 = 25 - 57 - 13 = 55 \bmod 100$.

Step 2: Input party B has an input of $y = 33$. They sample two random numbers, say $y_1 = 44$ and $y_2 = 30$, and compute the third share as $y_3 = 33 - 44 - 30 = 59 \bmod 100$.

Step 3: Shares are distributed to computing servers, so that server i gets x_i and y_i.

Step 4: Each server locally sums their shares from A and B. For example, Server 1 calculates $z_1 := x_1 + y_1 = 57 + 44 = 1 \bmod 100$.

Step 5: The servers send their partial results z_i to an analyst.

Step 6: The analyst sums the partial results $(1 + 43 + 14 = 58 \bmod 100)$ to reveal the true sum $(25 + 33 = 58)$. No party learns the other party's input or the intermediate shares.

The same steps are visualised in Figure 4 to make the flow of shares between the input parties, the computing parties, and the analyst easier to follow.

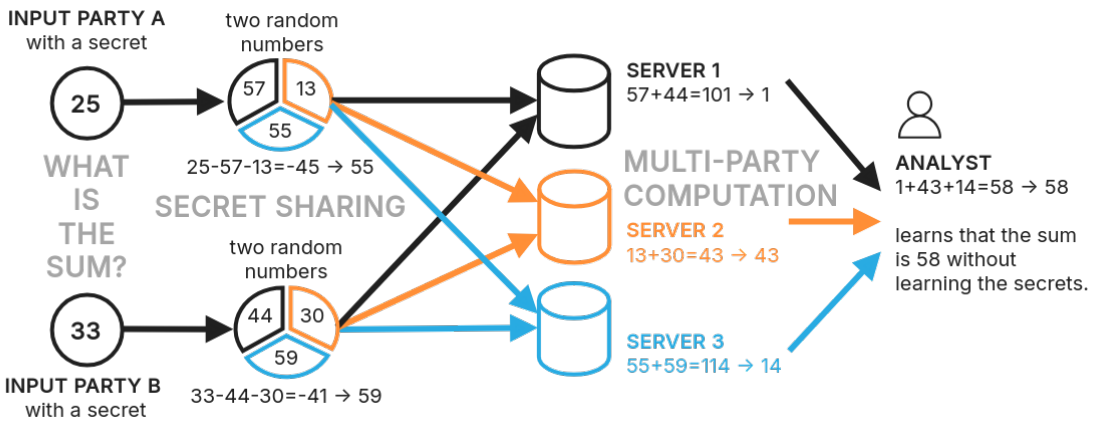

Figure 4. Additive secret sharing allows a sum to be computed without revealing private inputs, adapted from [28]

This example demonstrates that operations can be performed on data without revealing the original values. Sharemind MPC builds on this principle to provide fast and secure computation over protected data [27].

4.4 Sharemind MPC

The description of the Sharemind MPC platform in this section follows the official Sharemind documentation [27, 29]. Sharemind MPC is a framework for building privacy-preserving applications using a three-party additive secret sharing scheme. The architecture defines three roles: input parties that provide the data, output parties that receive the result, and computing parties that perform the calculations. The data is stored as shares across three non-colluding servers, each hosted by one of the computing parties, which jointly compute on the data without ever reconstructing it during computation [20].

4.4.1 Practical Applications

Sharemind MPC has been deployed in real-world privacy-preserving analysis systems. For example, Estonian government agencies have used Sharemind MPC to run a registered study on linked education and tax data [30]. Similarly, MPC has been applied to support collaborative data analysis between multiple institutions, allowing them to jointly compute statistical summaries and models on confidential datasets, while preserving privacy and ensuring compliance with data protection [31, 32]. These examples demonstrate the versatility and reliability of the framework in handling sensitive data in critical sectors.

4.4.2 SecreC Programming Language

Privacy-preserving applications using Sharemind MPC are programmed with the SecreC domain-specific language that partitions data into private and public domains [29]. SecreC's C-like syntax makes it familiar to programmers, and it supports common control flow constructs (conditional statements, loops, functions). Each data type has a security domain assigned to it in addition to its value, for example, `private int` or `public bool` [17]. This requires the programmer to clearly distinguish which values can be public and which must remain encrypted for processing. The type system ensures that only secure protocols are used to compute on private values and private information cannot unintentionally leak into public variables. In addition to elementary arithmetic and logical operations, there are libraries for processing vectors, arrays, and matrices in the private domain [29].

Sharemind MPC supports several MPC schemes, called protection domains. The most advanced of these is `shared3p`, which corresponds to the sharing of secret thresholds $(3, 3)$ under the passive corruption model [27]. Since secure computation protocols require network communication, performance is often critical and, as a result, runtime is strongly affected by network latency. Even simple arithmetic operations can become costly when they trigger many communication rounds. To mitigate this performance bottleneck, optimisations are essential. One example is SIMD (Single Instruction Multiple Data) parallelism, which allows multiple data elements to be processed simultaneously within a single computation step, thus SIMD reduces the number of rounds [33].

4.5 ShareSat

This subsection describes ShareSat and its privacy-preserving conjunction analysis workflow, based primarily on internal project documentation [4, 5] and the ShareSat conference paper [28].

ShareSat is a project developed by Cybernetica in collaboration with the European Space Agency (ESA) as part of the General Support Technology Programme [34]. The project's goal is to enhance space safety using secure computation technique MPC (see Section 4.2) to calculate the collision risk between spacecraft. This is needed because reliable conjunction analysis requires precise orbital data, but such data can be commercially or security sensitive. ShareSat uses the Sharemind MPC framework to enable confidential collaboration between different satellite owners and operators (O/O) without disclosing any private orbital data to one another.

4.5.1 Privacy-Preserving Conjunction Analysis

The main idea behind ShareSat is to change the trust model of conjunction analysis. In conventional workflows, sensitive data is often shared with a single party that must be trusted to perform analysis. ShareSat instead allows the analysis to be computed jointly, so that only the agreed outputs are revealed and the raw inputs remain private. In this way, satellite O/O can collaborate on collision assessment without giving up control over confidential mission data.

ShareSat is designed to enable joint collision probability assessment based on private orbital data provided by multiple input parties. The system is structured around three roles: input parties, computing parties, and output parties. Input parties provide the confidential orbital data, computing parties perform the secure computation without accessing the raw inputs, and output parties receive the final assessment result.

In ShareSat, only the minimal information necessary to avoid a collision is revealed by the analysis output. In practice, the private inputs include detailed orbital states, while the outputs are limited to the identifiers of the involved objects together with the computed collision probability and the estimated TCA. This allows O/O to assess potentially dangerous conjunctions and coordinate further action without disclosing the underlying confidential data.

5. Implementation

This chapter documents the implementation work carried out in this thesis in the ShareSat codebase to replace the existing linear interpolation component with a Lagrange interpolation approach in SecreC (see Section 4.4.2).

After reviewing the interpolation methods in Chapter 3, the goal was to enable polynomial interpolation with configurable polynomial degree in the secure computation pipeline, without breaking the existing conjunction assessment flow.

5.1 Iterative development of the Lagrange Interpolator

The implementation work began by studying the mathematical and technical aspects of the problem. This included studying interpolation theory, learning the SecreC programming language, and exploring the ShareSat repository to understand how the secure computation pipeline is organised. Particular attention was paid to the data flow through the conjunction-assessment process: how trajectory samples are prepared, how the TCA is determined, and where interpolation is applied to estimate satellite states between sampled trajectory points.

5.1.1 Two-Point Prototype

To keep the first change small and easy to validate, the work started with a minimal two-point version of the Lagrange implementation method. This version used one trajectory sample before and one trajectory sample after the target point. Implementing the simplest case first made it easier to verify correctness, debug the SecreC code, and confirm that the interpolation function could be integrated into the MPC pipeline successfully.

In the following pseudocode, $target_x$ denotes the target time at which the interpolated satellite state is evaluated. The values $x_0, x_1, ...$ represent known time points from the trajectory data, and the corresponding vectors $\vec{y_0}, \vec{y_1}, ... \in \mathbb{R}^6$ symbolise the associated satellite states, consisting of position and velocity components in three spacial dimensions. The interpolation result is denoted by $\vec{L}_x$, which represents the interpolated 6-dimensional state vector at $target_x$.

The pseudocode for the two-point case is given in Algorithm 1.

Algorithm 1 Lagrange Interpolation Two-Point Case

Require: $target\_x$, x_0, x_1, $\vec{y_0}$, $\vec{y_1}$, where $target\_x \in \mathbb{R}$ is the target time; x_0, $x_1 \in \mathbb{R}$ are time points; $\vec{y_0}$, $\vec{y_1} \in \mathbb{R}^6$ are 6-dimensional state vectors (position and velocity)

Ensure: interpolated state vector $\vec{L_x} \in \mathbb{R}^6$ at $target_x$

1: $l_0 \leftarrow (target\_x - x_1)/(x_0 - x_1)$
2: $l_1 \leftarrow (target\_x - x_0)/(x_1 - x_0)$
3: $\vec{L_x} \leftarrow l_0 \cdot \vec{y_0} + l_1 \cdot \vec{y_1}$
4: **return** $\vec{L_x}$

This initial two-point implementation served mainly as a correctness check and as a way to validate the SecreC implementation within the existing pipeline. Since the two-point Lagrange form is mathematically equivalent to linear interpolation, it was not expected to provide a qualitative improvement over the previously used linear method.

5.1.2 Extension to Three- and Four-Point Cases

The TCA-based point-selection logic (see Appendix A in Listing A.1) was modified so that the interpolation step could use more trajectory samples instead of the original two points. Once this wider selection was available, the Lagrange implementation was extended first to a three-point version and then to a four-point version.

The pseudocode for the three-point case is given in Algorithm 2.

Algorithm 2 Lagrange Interpolation Three-Point Case

Require: $target\_x$, x_0, x_1, x_2, $\vec{y_0}$, $\vec{y_1}$, $\vec{y_2}$, where $target\_x \in \mathbb{R}$ is the target time; x_0, x_1, $x_2 \in \mathbb{R}$ are time points; $\vec{y_0}$, $\vec{y_1}$, $\vec{y_2} \in \mathbb{R}^6$ are 6-dimensional state vectors (position and velocity)

Ensure: interpolated state vector $\vec{L_x} \in \mathbb{R}^6$ at $target_x$

1: $l_0 \leftarrow (target\_x - x_1) \cdot (target\_x - x_2)/((x_0 - x_1) \cdot (x_0 - x_2))$
2: $l_1 \leftarrow (target\_x - x_0) \cdot (target\_x - x_2)/((x_1 - x_0) \cdot (x_1 - x_2))$
3: $l_2 \leftarrow (target\_x - x_0) \cdot (target\_x - x_1)/((x_2 - x_0) \cdot (x_2 - x_1))$
4: $\vec{L_x} \leftarrow l_0 \cdot \vec{y_0} + l_1 \cdot \vec{y_1} + l_2 \cdot \vec{y_2}$
5: **return** $\vec{L_x}$

The pseudocode for the four-point case is given in Algorithm 3.

Algorithm 3 Lagrange Interpolation Four-Point Case

Require: $target\_x$, x_0, x_1, x_2, x_3, $\vec{y_0}$, $\vec{y_1}$, $\vec{y_2}$, $\vec{y_3}$, where $target\_x \in \mathbb{R}$ is the target time; x_0, x_1, x_2, $x_3 \in \mathbb{R}$ are time points; $\vec{y_0}$, $\vec{y_1}$, $\vec{y_2}$, $\vec{y_3} \in \mathbb{R}^6$ are 6-dimensional state vectors (position and velocity)

Ensure: interpolated state vector $\vec{L_x} \in \mathbb{R}^6$ at $target_x$

1: $l_0 \leftarrow (target\_x - x_1) \cdot (target\_x - x_2) \cdot (target\_x - x_3) / ((x_0 - x_1) \cdot (x_0 - x_2) \cdot (x_0 - x_3))$
2: $l_1 \leftarrow (target\_x - x_0) \cdot (target\_x - x_2) \cdot (target\_x - x_3) / ((x_1 - x_0) \cdot (x_1 - x_2) \cdot (x_1 - x_3))$
3: $l_2 \leftarrow (target\_x - x_0) \cdot (target\_x - x_1) \cdot (target\_x - x_3) / ((x_2 - x_0) \cdot (x_2 - x_1) \cdot (x_2 - x_3))$
4: $l_3 \leftarrow (target\_x - x_0) \cdot (target\_x - x_1) \cdot (target\_x - x_2) / ((x_3 - x_0) \cdot (x_3 - x_1) \cdot (x_3 - x_2))$
5: $\vec{L_x} \leftarrow l_0 \cdot \vec{y_0} + l_1 \cdot \vec{y_1} + l_2 \cdot \vec{y_2} + l_3 \cdot \vec{y_3}$
6: **return** $\vec{L_x}$

These fixed-size implementations showed that Lagrange interpolation could be extended beyond the original two-point case. However, writing a separate version for each point count is not practical, which motivated the generalised n-point implementation.

5.1.3 Generalised n-Point Implementation

After the fixed-size versions had been implemented and verified as working, the Lagrange interpolation implementation was generalised with the pseudocode given in Algorithm 4 and the corresponding SecreC implementation is included in the Appendix A in Listing A.2. The fixed polynomial expressions were replaced with a loop-based formulation, where the interpolation terms are generated according to the selected number of input points.

Algorithm 4 Lagrange Interpolation Generalised Case

Require: $target\_x$, $\vec{x}$, $\mathbf{Y}$, where $target\_x \in \mathbb{R}$ is the target time; $\vec{x} \in \mathbb{R}^{n+1}$ is the 1D array of time points; $\mathbf{Y} \in \mathbb{R}^{n+1} \times \mathbb{R}^6$ is the 2D matrix of state vectors (position and velocity), where row i is $\vec{y}_i \in \mathbb{R}^6$, n is the number of known points

Ensure: interpolated state vector $\vec{L}_x \in \mathbb{R}^6$ at $target_x$

$n \leftarrow \text{size}(x)$
for $i \leftarrow 0$ **to** $n - 1$ **do**
    $dt[i] \leftarrow target\_x - x[i]$
end for
$\vec{L_x} \leftarrow \mathbf{0}$ ▷ 6-dimensional zero vector
for $i \leftarrow 0$ **to** $n - 1$ **do**
    $l_i \leftarrow 1$
    for $j \leftarrow 0$ **to** $n - 1$ **do**
        if $j \neq i$ **then**
            $l_i \leftarrow l_i \cdot \dfrac{dt[j]}{x[i] - x[j]}$
        end if
    end for
    $\vec{L_x} \leftarrow \vec{L_x} + l_i \cdot \vec{y}_i$
end for
return $\vec{L_x}$

This makes the approach configurable and allows the same implementation to be evaluated with different numbers of points, depending on the desired trade-off between accuracy and performance. In Chapter 6, these trade-offs are measured and analysed.

5.2 Verification and Benchmarking Setup

The purpose of benchmarking and testing is to evaluate the correctness and computational performance of the implemented Lagrange interpolation method. In this thesis, verification is used to confirm that the implementation produces correct interpolated state vectors, while benchmarking is used to assess the method's efficiency across different input conditions.

The testing strategy is divided into two distinct phases:

1. **Correctness Testing:** Validating the interpolation logic against expected output data to ensure accuracy.
2. **Performance Benchmarking:** Measuring the execution time of the Lagrange interpolation section under specific MPC configurations.

5.2.1 Correctness Testing

To verify correctness, an existing trajectory point is removed, interpolated from its surrounding points, and then compared with its original value. The correctness evaluation was carried out using the following setup:

- **Data Source:** Real satellite orbit data in OEM format (provided by ESA), which contained four trajectories sampled at one-minute intervals. Two trajectories contained 60 points each, while the other two contained 663 points each.
- **Method:** Local windows of 3, 5, and 7 trajectory points were selected. In each window, the middle point was removed and treated as a reference value. The remaining 2, 4 and 6 points were then used as input for the three interpolation configurations.
- **Execution:** The surrounding points were used as interpolation inputs and processed by the Sharemind MPC application. The Lagrange polynomial was evaluated on secret-shared data to reconstruct the missing state vector at the masked timestamp.
- **Metric:** The reconstructed state vector was compared with the original reference value. Position and velocity errors were evaluated separately and summarised using the mean, 95th percentile, and 99th percentile. Position error was evaluated in metres and compared against the target accuracy of ± 1 m. Velocity error was evaluated in millimetres per second and compared against the target accuracy of ± 1 mm/s. This requirement was based on the operationally used error margins in ESA [35].

This approach ensures that the SecreC implementation preserves numerical accuracy and does not introduce significant errors.

5.2.2 Chunk-Based Benchmarking Procedure

A chunk-based benchmarking routine was used to turn a small number of input trajectories into a large number of tests. The benchmark uses four trajectories and processes each of them in overlapping chunks. For an interpolation configuration using n points, a trajectory of length $total\_len$ is divided into $total\_len - n - 1$ valid chunks. Each chunk forms one interpolation test case.

The corresponding SecreC implementation is provided in Appendix A in Listing A.3.

5.2.3 Performance Benchmarking

Benchmarking evaluated the computational overhead of performing interpolation securely. The benchmarking setup was defined as follows:

- **Environment:** 3 Sharemind MPC Servers (2× Intel Xeon E5-2640, 128 GB RAM, 10 Gb/s network).
- **Procedure:** Each benchmark test was repeated 3 times to account for system variance.
- **Metric:** Performance was measured as the execution time per-chunk. The results were reported using the mean, 95th percentile, and 99th percentile.

The setup made it possible to evaluate how its execution time changes with the number of interpolation points. The measured results are used in Chapter 6 to compare how the computational cost changes when the interpolation window is increased.

6. Results

This chapter presents the results of the verification and benchmarking experiments for the implemented Lagrange interpolation method. The experimental setup, validation method, benchmark procedure, and target accuracy were described in Section 5.2. Here, the focus is on the obtained correctness and performance results for the tested 2-point, 4-point, and 6-point interpolation windows.

6.1 Accuracy Results

Accuracy is reported in terms of mean error and 95th and 99th percentile errors. The mean error indicates the typical interpolation accuracy across all evaluated samples. The 95th and 99th percentile errors show how large the errors become in the worst-performing portion of the data, without basing the analysis on a single extreme maximum value. This is important because the interpolation method must remain reliable for almost all samples, not only perform well on average. The ideal target accuracy was ± 1 m for position and ± 1 mm/s for velocity [35]. The results are summarised in Table 1 and Table 2. In both tables, the configuration that satisfies the required accuracy is highlighted in bold.

Table 1. Position error by interpolation point count

Configuration	Mean (m)	Top 95% (m)	Top 99% (m)
2	15208.06	15609.99	15727.88
4	22.45	23.69	24.43
6	**0.030**	**0.041**	**0.049**

Table 2. Velocity error by interpolation point count

Configuration	Mean (mm/s)	Top 95% (mm/s)	Top 99% (mm/s)
2	16874.48	17630.79	17894.27
4	24.91	26.51	28.30
6	**0.192**	**0.474**	**0.774**

The results show a clear and consistent improvement as the number of interpolation points increases. The 2-point case produces errors of the order of tens of kilometres for position and tens of metres per second for velocity, which is consistent with the expectation stated in Chapter 3 that two surrounding points cannot capture the curvature of the satellite orbit sufficiently.

Moving to 4 points reduces the mean position error from 15208.06 m to 22.45 m and the mean velocity error from 16874.48 mm/s to 24.91 mm/s. This is an improvement of roughly three orders of magnitude compared with the 2-point case. However, the 4-point configuration still does not meet the target accuracy. Its 95th- and 99th-percentile errors also remain above the required limits, which shows that the remaining error is not caused only by a few extreme samples. Instead, the configuration is still consistently outside the required accuracy range.

The 6-point configuration reduces the mean position error to 0.030 m and the mean velocity error to 0.192 mm/s. The 95th- and 99th-percentile position errors are 0.041 m and 0.049 m, while the corresponding velocity errors are 0.474 mm/s and 0.774 mm/s. Since even the 99th-percentile errors are below the required limits, at least 99% of the evaluated samples satisfy the target accuracy for both position and velocity.

Figure 5 shows the same improvement trend on a logarithmic scale. This scale is used because the errors decrease by several orders of magnitude from 2-point to 6-point interpolation. While the tables show whether each configuration meets the absolute accuracy limits, the figure makes the overall convergence behaviour easier to compare visually.

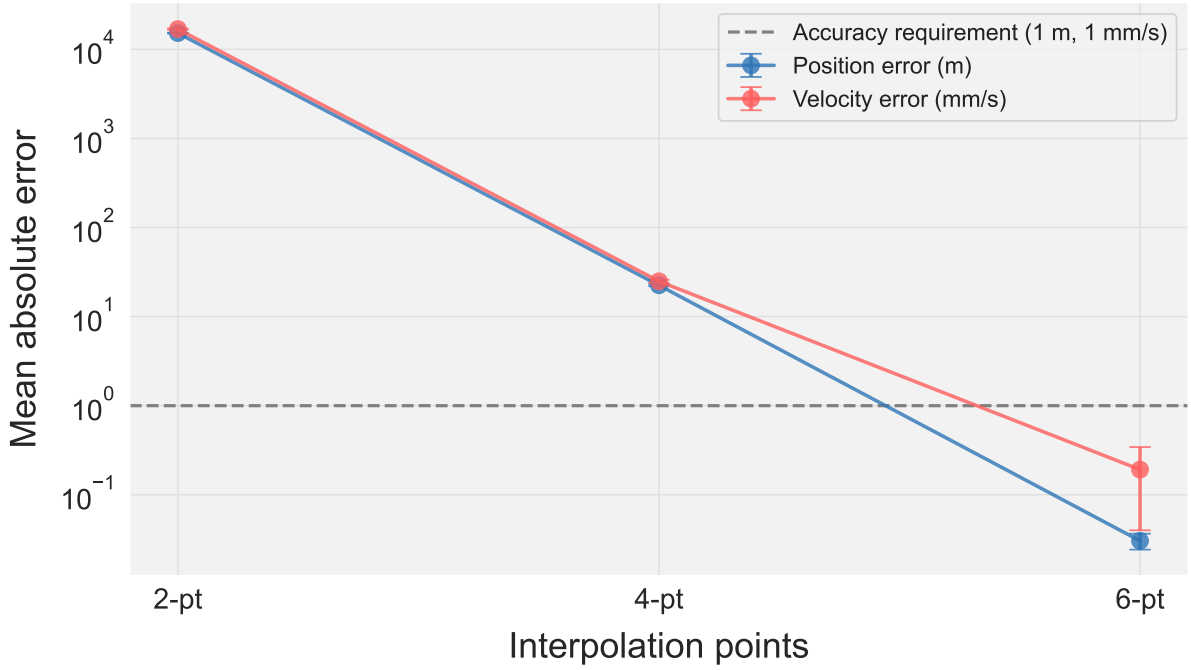

Figure 5. Only 6-point interpolation brings both mean position and velocity errors below the required accuracy limit

Overall, the accuracy results show that increasing the interpolation window substantially improves the reconstructed state vectors. The 4-point configuration is a major improvement over the 2-point configuration, but only the 6-point configuration reaches the required practical accuracy level for both position and velocity.

6.2 Performance Results

Performance was evaluated by measuring the execution time of the Lagrange interpolation routine in the Sharemind MPC cluster (see Section 5.2.3). As described in Section 5.2, performance is reported as execution time per-chunk in Table 3. Each configuration was evaluated across all valid chunks from all four trajectories.

Table 3. Per-chunk Lagrange Duration by interpolation Point Count

Polynomial degree	Mean (ms)	Top 95% (ms)	Top 99% (ms)
2	1.667	1.676	1.679
4	6.792	6.821	6.829
6	15.551	15.625	15.642

The per-chunk execution time increases as the number of interpolation points increases. The mean duration increases from about 1.67 ms for 2-point interpolation to 6.79 ms for 4-point interpolation and 15.55 ms for 6-point interpolation. Compared with the 2-point configuration, the 4-point configuration is approximately 4.1 times slower, while the 6-point configuration is approximately 9.3 times slower.

This increase is expected because higher-point Lagrange interpolation requires evaluating more basis terms and, therefore, performs more arithmetic operations for each chunk. The 95th- and 99th-percentile durations are very close to the mean values for all three configurations, which shows that the runtime is stable across the evaluated chunks and is not strongly affected by occasional outliers.

Figure 6 shows the measured mean durations per-chunk together with a quadratic fit. The measured values closely follow the fitted curve, supporting the expectation that runtime increases approximately quadratically with the number of interpolation points.

Considering both accuracy and performance, the 6-point configuration is the most suitable option. It is the only configuration that reaches the practical target accuracy, although it is also the most computationally expensive, with the highest average per-chunk runtime. The 4-point

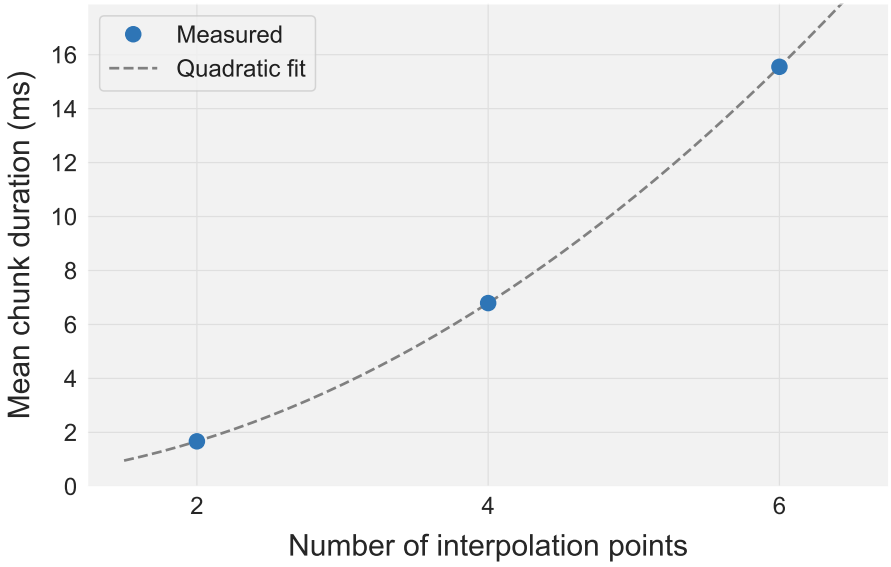

Figure 6. Per-chunk interpolation runtime increases approximately quadratically with the number of points

configuration is faster, but its accuracy remains outside the desired target range. Therefore, the 6-point configuration is the most suitable of the tested options when accuracy is prioritised, while future optimisation is needed to reduce its runtime cost.

6.3 Future Work

The results show that the 6-point configuration provides the best accuracy, but it also has the highest runtime cost. Future work should therefore focus on optimising the implementation while not deteriorating the interpolation accuracy.

6.3.1 SIMD Optimisation

One possible optimisation direction is utilising SIMD algorithms (Single Instruction, Multiple Data) [33]. SIMD allows the same operation to be applied to several data values in parallel. This is relevant because Lagrange interpolation repeatedly performs similar arithmetic operations, such as products and sums, over multiple interpolation points and coordinate values. Instead of executing many small secret-shared operations consecutively, identical operations could be grouped and processed together where the algorithm allows it. This reduces loop overhead and improves performance, especially for the 6-point configuration where the number of operations is highest.

To put the runtime into context, the complete ShareSat conjunction-processing workflow takes about 7 minutes per conjunction [35]. If the overlapping part of two trajectories contains X

trajectory points, then the workflow first performs X interpolations to align one trajectory with the time points of the other. After this, one point is selected as the centre of the TCA search window. The final refinement step then evaluates 100 subdivisions for both trajectories, which adds $2 \times 100 = 200$ interpolation calls. Therefore, the total number of interpolation calls is approximately $X + 200$.

Combined with the operation counts from Section 3.2, this gives a large number of MPC operations for the 6-point configuration. Each interpolation call requires approximately 6 divisions and 36 multiplications. For example, if the overlapping part contains 663 trajectory points, then the workflow requires approximately $(663 + 200) \times 6 = 5178$ divisions and $(663 + 200) \times 36 = 31068$ multiplications. Since multiplication and division are expensive under MPC, this makes SIMD a relevant optimisation direction. Independent interpolation operations could be grouped and evaluated in parallel, reducing the cost of repeated arithmetic operations without changing the interpolation method itself.

6.3.2 Further Accuracy Evaluation

Future work could also investigate whether increasing the interpolation window beyond 6 points provides additional accuracy improvements, especially for velocity. However, this should be carefully evaluated because adding more points also increases computational cost and does not always guarantee better interpolation accuracy.

The accuracy evaluation should also be repeated on a wider range of orbit types and trajectory conditions. For example, future benchmarks could compare low Earth orbit and geostationary Earth orbit objects, elliptic and hyperbolic trajectories, fast and slow objects, and trajectories with stable or changing acceleration. This would indicate whether the 6-point configuration remains reliable across different orbital regimes, rather than only for the trajectories used in the current benchmark.

Another direction is to compare Lagrange interpolation with alternative interpolation methods. Spline-based interpolation or other polynomial interpolation approaches could be evaluated to determine whether they provide a better balance between accuracy, numerical stability, and performance in the MPC setting.

The scope of this evaluation is limited to the accuracy and runtime of the interpolation step. The effect of the interpolation error on the final collision probability estimate is left for future work.

7. Conclusion

This thesis implemented and analysed Lagrange interpolation in the ShareSat system for satellite state estimation under secure multi-party computation. Since satellite data is available only at discrete time steps, interpolation is needed to estimate positions and velocities between known points, particularly when determining the Time of Closest Approach in conjunction analysis.

The main contribution of the thesis was the implementation and evaluation of Lagrange interpolation in SecreC. The work started with fixed-size interpolation versions and later generalised to a n-point implementation. This allowed the same method to be tested with different interpolation window sizes, such as two, four, and six points. A benchmarking setup was also developed to evaluate both correctness and performance by applying interpolation to trajectory chunks.

The results show that Lagrange interpolation can significantly improve trajectory estimation accuracy when more interpolation points are used. The 2-point configuration did not provide sufficient accuracy, and the 4-point configuration improved the results but still did not meet the required limits. The 6-point Lagrange configuration was the only tested option that satisfied the target accuracy for both position and velocity, including the mean, 95th-percentile, and 99th-percentile errors.

Performance results showed that accuracy improvement comes with increased runtime. Although the 6-point configuration was the slowest, its execution time remained consistent across trajectory chunks and followed a quadratic growth trend. Overall, the results indicate that the 6-point Lagrange interpolation is the most suitable tested configuration when accuracy is prioritised.

Overall, the thesis shows that Lagrange interpolation can be integrated into the ShareSat workflow based on multi-party computation to support accurate trajectory estimation without revealing raw orbital data. The main limitation is that the evaluation was limited to the tested trajectories and interpolation configurations. The thesis also focused on interpolation accuracy and runtime, rather than evaluating the full effect on final collision probability estimates. Future work could evaluate more orbital scenarios, compare other interpolation methods, and further optimise the 6-point implementation.

References

[1] ESA Space Debris Office. Space Environment Statistics. https://sdup.esoc.esa.int/discosweb/statistics/ (25.04.2026). 2026.

[2] Kelvey J. Understanding the Misunderstood Kessler Syndrome. https://aerospaceamerica.aiaa.org/features/understanding-the-misunderstood-kessler-syndrome/ (09.05.2026). 2024.

[3] NASA Orbital Debris Program Office. Satellite Collision Leaves Significant Debris Clouds. *Orbital Debris Quarterly News* vol 13.2 (2009). https://orbitaldebris.jsc.nasa.gov/quarterly-news/pdfs/ODQNv13i2.pdf (27.04.2026), pp. 1–2.

[4] Cybernetica AS. ShareSat - D1: Space object trajectory propagation data models and algorithms. Restricted internal material. s.a.

[5] Cybernetica AS. ShareSat - D2: Adapt conjunction analysis techniques for MPC. Restricted internal material. s.a.

[6] Wen T.-H., Teo T.-A., Hsu K. H., Wu M. C., and Lee Y. S. Assessment of Orbital Interpolation Methods for Accurate Baseline Estimation in InSAR Applications. *Asian Journal of Geoinformatics* vol 24.1 (2024). DOI: 10.30217/AJG.202404_24(1).0003.

[7] Curtis H. D. Chapter 2 - The Two-Body Problem. *Orbital Mechanics for Engineering Students (Second Edition)*. Aerospace Engineering. Boston: Butterworth-Heinemann, 2010, pp. 61–153. DOI: 10.1016/B978-0-12-374778-5.00002-7.

[8] University of Helsinki. 3.1. Satellite Orbits. https://courses.mooc.fi/org/uh-physics/courses/sustainable-space/chapter-3/satellite-orbits (26.04.2026). s.a.

[9] Curtis H. D. Chapter 4 - Orbits in Three Dimensions. *Orbital Mechanics for Engineering Students (Second Edition)*. Aerospace Engineering. Boston: Butterworth-Heinemann, 2010, pp. 199–254. DOI: 10.1016/B978-0-12-374778-5.00004-0.

[10] NASA Office of the Chief Engineer. NASA Spacecraft Conjunction Assessment and Collision Avoidance Best Practices Handbook. https://nodis3.gsfc.nasa.gov/OCE_docs/OCE_51.pdf (12.05.2026). 2023.

[11] Consultative Committee for Space Data Systems. Orbit Data Messages. CCSDS 502.0-B-3. 4. https://ccsds.org/Pubs/502x0b3e1.pdf (27.04.2026). 2023.

[12] Cybernetica AS. ShareSat - D6: Architecture with precision and performance targets. Restricted internal material. s.a.

[13] Oja P. Numerical Methods. Lecture notes, University of Tartu. https://courses.ms.ut.ee/MTMM.00.005/2021_spring/uploads/Main/nm2018-eng.pdf (26.04.2026). 2018.

[14] Feng Y. and Zheng Y. Efficient Interpolations to GPS Orbits for Precise Wide Area Applications. *GPS Solutions* vol 9.4 (2005), pp. 273–282. DOI: 10.1007/s10291-005-0133-y.

[15] Pustoshilov A. S. and Tsarev S. P. Universal Coefficients for Precise Interpolation of GNSS Orbits from Final IGS SP3 Data. *2017 International Siberian Conference on Control and Communications (SIBCON)*. 2017, pp. 1–6. DOI: 10.1109/SIBCON.2017.7998463.

[16] Nguyen Thi Xuan Mai and Nguyen Trung Thanh. The Analysis of the Advantages and Disadvantages of Construction Method of Lagrange Interpolated Polynomial and Newton Interpolated Polynomial. *International Journal of Advanced Multidisciplinary Research and Studies* vol 2.4 (2022). https://www.multiresearchjournal.com/admin/uploads/archives/archive-1659090406.pdf (29.01.2026), pp. 431–434.

[17] Cybernetica AS. SecreC Language Reference. https://docs.sharemind.cyber.ee/sharemind-mpc/2023.09/development/secrec-reference.html (09.05.2026). s.a.

[18] Runge C. Über empirische Funktionen und die Interpolation zwischen äquidistanten Ordinaten. *Zeitschrift für Mathematik und Physik* vol 46 (1901).

[19] Rivas V. Multi-Party Computation (MPC): Secure, Private Collaboration. https://www.cyfrin.io/blog/multi-party-computation-secure-private-collaboration (27.04.2026). 2025.

[20] Bogdanov D., Laur S., and Willemson J. Sharemind: A Framework for Fast Privacy-Preserving Computations. *Computer Security - ESORICS 2008*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 192–206. DOI: 10.1007/978-3-540-88313-5_13.

[21] Pullonen-Raudvere P. Foundations of Efficient and Secure Algorithm Development for Secure Multiparty Computation. https://dspace.ut.ee/items/f52fc12d-ae92-4906-9205-ac87ed250d56 (10.05.2026). PhD thesis. University of Tartu Institute of Computer Science, 2024.

[22] Yao A. C. Protocols for secure computations. *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*. 1982, pp. 160–164. DOI: 10.1109/SFCS.1982.38.

[23] Bogetoft P., Christensen D. L., Damgård I., Geisler M., Jakobsen T., Krøigaard M., Nielsen J. D., Nielsen J. B., Nielsen K., Pagter J., Schwartzbach M., and Toft T. Secure Multiparty Computation Goes Live. *Financial Cryptography and Data Security*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 325–343. DOI: 10.1007/978-3-642-03549-4_20.

[24] Malkin T. Summary of Lecture on Secret Sharing. Lecture notes, COMS W4261: Introduction to Cryptography, Columbia University. https://www.cs.columbia.edu/~tal/4261/F19/secretsharingf19.pdf (27.04.2026). 2019.

[25] ScienceDirect Topics. Threshold Scheme. https://www.sciencedirect.com/topics/computer-science/threshold-scheme (29.01.2026). s.a.

[26] Shamir A. How to share a secret. *Commun. ACM* vol 22.11 (1979), pp. 612–613. DOI: 10.1145/359168.359176.

[27] Cybernetica AS. Sharemind MPC: Overview. https://docs.sharemind.cyber.ee/sharemind-mpc/2023.09/prologue/overview.html (27.04.2026). s.a.

[28] Veskus K. H. and Valdma S.-M. Decade of Advancement: Evaluating Multi-Party Computation Progress in Satellite Collision Avoidance. *Proceedings of the 9th European Conference on Space Debris*. https://conference.sdo.esoc.esa.int/proceedings/sdc9/paper/286/SDC9-paper286.pdf (29.04.2026). Bonn, Germany: ESA Space Debris Office, 2025.

[29] Cybernetica AS. Sharemind MPC. https://docs.sharemind.cyber.ee/sharemind-mpc/2023.09/prologue/sharemind-mpc.html (26.04.2026). s.a.

[30] Cybernetica AS. PRIST: Privacy-Preserving Statistical Studies. https://cyber.ee/research/projects/prist/ (27.04.2026). s.a.

[31] Cybernetica AS. PAI-MACHINE: Synthesis of Machine-Optimized Cryptographic Protocols with Applications in Secure Machine Learning Systems. https://cyber.ee/research/projects/paimachine/ (27.04.2026). s.a.

[32] Cybernetica AS. JOCONDE: Joint On-Demand Computation with No Data Exchange. https://cyber.ee/research/projects/joconde/ (27.04.2026). s.a.

[33] Cybernetica AS. The SecreC Programming Language: Parallel Execution and SIMD. https://docs.sharemind.cyber.ee/sharemind-mpc/2023.09/development/secrec-tutorial.html#_parallel_execution_and_simd (26.04.2026). s.a.

[34] European Space Agency. About the General Support Technology Programme (GSTP). https://www.esa.int/Enabling_Support/Space_Engineering_Technology/Shaping_the_Future/About_the_General_Support_Technology_Programme_GSTP (03.02.2026). s.a.

[35] Cybernetica AS. ShareSat - D5: Benchmarking and precision results. Restricted internal material. s.a.

A. Appendix

A.1 TCA point selection in SecreC

Listing A.1 shows the modified TCA-based point selection logic used to choose the surrounding trajectory points for interpolation. This listing is included because the point-selection step determines which samples are passed to the Lagrange interpolation function.

```
void test_lagrangeInterp() {
// Read OEM data
trajectory<pd_shared3p, float64> traj = getTraj();

// Read CDM data for interpolation point and expected value
// at interpolation point
pd_shared3p float64[[1]] true_p(3), true_v(3);
    pd_shared3p uint64 true_tca = argument("tca");
    true_p[0] = argument("tx1");
    true_p[1] = argument("ty1");
    true_p[2] = argument("tz1");
    true_v[0] = argument("tdx1");
    true_v[1] = argument("tdy1");
    true_v[2] = argument("tdz1");

// Finds previous and next measurement points in the trajectory
pd_shared3p uint[[1]] idxs =
    find_nearest_indices(traj.t, true_tca);
pd_shared3p uint64 t_prev = vectorLookup(traj.t, idxs[0]);
pd_shared3p uint64 t_next = vectorLookup(traj.t, idxs[1]);

// Determine time points two steps before and after
// the current nearest indices
pd_shared3p uint64 x = 1;
pd_shared3p uint64 t_prev_x2 = t_prev -x;
pd_shared3p uint64 t_next_x2 = t_next +x;

// Finds the point before previous and after next measurement
// points in the trajectory
pd_shared3p uint[[1]] idxs_prev =
    find_nearest_indices(traj.t, t_prev_x2);
pd_shared3p uint[[1]] idxs_next =
    find_nearest_indices(traj.t, t_next_x2);
```

```
    t_prev_x2 = vectorLookup(traj.t, idxs_prev[0]);
    t_next_x2 = vectorLookup(traj.t, idxs_next[1]);

    pd_shared3p uint64 O = t_prev_x2;
    pd_shared3p float64[[1]] X = {
        (float64)(t_prev_x2-O),
        (float64)(t_prev-O),
        (float64)(t_next-O),
        (float64)(t_next_x2-O)};
    pd_shared3p float64 target_x = (float64) (true_tca- O);

    pd_shared3p float64[[2]] Y(4,6);
    Y[0,:3] = matrixLookupRow(traj.p, idxs_prev[0]);
    Y[1,:3] = matrixLookupRow(traj.p, idxs[0]);
    Y[2,:3] = matrixLookupRow(traj.p, idxs[1]);
    Y[3,:3] = matrixLookupRow(traj.p, idxs_next[1]);

    Y[0,3:] = matrixLookupRow(traj.v, idxs_prev[0]);
    Y[1,3:] = matrixLookupRow(traj.v, idxs[0]);
    Y[2,3:] = matrixLookupRow(traj.v, idxs[1]);
    Y[3,3:] = matrixLookupRow(traj.v, idxs_next[1]);

    pd_shared3p float64[[1]] PV_lagrange =
        interpolate_lagrange(target_x, X[:4], Y[:4, :]);
    check(PV_lagrange[:3], PV_lagrange[3:], true_p, true_v);
}
```

A.2 Lagrange Interpolation in SecreC

Listing A.2 shows the generalised SecreC implementation of the Lagrange interpolation routine. This listing is included to document how the two-, three-, and four-point prototypes were replaced with a configurable n-point implementation.

```
template <domain D : shared3p, type T>
D T[[1]] interpolate_lagrange(
    D float64 target_x, D T [[1]] x, D T[[2]] y
) {
    uint32 section = newSectionType("Lagrange");
    uint32 id = startSection(section, n);
    uint64 n = size(x);

    // Compute the differences between `target_t` and each x[i]
    // These differences are used to evaluate
    // the Lagrange basis polynomials.
    D T[[1]] dt(n);
    dt = (T)target_x;
    dt = dt - x;        // pointwise subtraction
    dt[n] = target_x - x[n]

    D T[[1]] Lx(6);
    D T[[1]] l(n) = 1;
    // Compute Lagrange basis polynomials
    for (uint64 i = 0; i < n; i++) {
        for (uint64 j = 0; j < n; j++){
            if (j != i) {
                l[i] *= dt[j] / (x[i] - x[j]);
            }
        }
        // Interpolation of the 6D State Vector
        // The interpolated state vector `Lx` is computed
        as a weighted sum of all the 6D vectors
        Lx += l[i] * y[i,:];
    }
    endSection(id);
    return Lx;
}
```

A.3 Chunk-Based Benchmark in SecreC

Listing A.3 shows the chunk-based benchmarking routine used during correctness and performance testing. This listing is included because it explains how the input trajectories were divided into repeated interpolation test cases.

```
void test_lagrangeInterp_chunks(
    trajectory<pd_shared3p, float64> traj, uint64 n
) {
    uint total_len = size(traj.t);
    uint num_chunks = total_len - n - 1;
    uint n_r = n/2;     // n = 5 -> idx = 2

    // temporary arrays for interpolation
    pd_shared3p float64[[1]] X_chunk(n);            // n time points
    pd_shared3p float64[[2]] Y_chunk(n, 6);         // n point vectors
    pd_shared3p float64[[2]] X_1_out(num_chunks, n-1);  // n-1 time points
    pd_shared3p float64[[3]] Y_1_out(num_chunks, n-1, 6);   // n-1 vectors
    pd_shared3p float64[[2]] remove_P(num_chunks, 3);   // removed position
    pd_shared3p float64[[2]] remove_V(num_chunks, 3);   // removed velocity
    pd_shared3p float64[[1]] target_x(num_chunks);  // time to interpolate
    pd_shared3p float64[[2]] PV_lagrange(num_chunks, 6);    // result
    pd_shared3p float64[[1]] O(num_chunks);     // origin time for chunk

    // Split trajectory into n-point chunks
    for (uint i = 0; i < num_chunks; i++) {
        uint start_idx = i * 1;
        uint end_idx = start_idx + n;

        X_chunk[:] = (float64) traj.t[start_idx : start_idx+n];
        Y_chunk[:, :3] = traj.p[start_idx : end_idx, :];
        Y_chunk[:, 3:] = traj.v[start_idx : end_idx, :];

        // save the middle point for validation
        remove_P[i, :] = Y_chunk[n_r, :3];      // position (x, y, z)
        remove_V[i, :] = Y_chunk[n_r, 3:];      // velocity (dx, dy, dz)
        target_x[i] = X_chunk[n_r];             // time of removed point

        // create n-1 -point subset
        uint subset_idx = 0;
        for (uint k = 0; k < n; k++) {
```

```
            if (k == n_r) continue;              // Skip the middle point

            X_1_out[i, subset_idx] = X_chunk[k];
            Y_1_out[i, subset_idx, :] = Y_chunk[k, :];
            subset_idx++;
        }
    }

    uint32 section = newSectionType("Full-Lagrange");
    uint32 id = startSection(section, num_chunks);
    for (uint i = 0; i < num_chunks; i++) {
        PV_lagrange[i, :] = interpolate_lagrange(
            target_x[i], X_1_out[i, :], Y_1_out[i, :, :]
        );
    }
    endSection(id);

    for (uint i = 0; i < num_chunks; i++) {
        check(
            PV_lagrange[i, :3], PV_lagrange[i, 3:],
            remove_P[i, :], remove_V[i, :]
        );
    }
}
```

License

I, Kätryn Varkel,

1. grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the digital archives of the University of Tartu until the expiry of the term of copyright, my thesis ”Interpolation of Satellite Orbital States in the Context of ShareSat”, supervised by Karl Hannes Veskus, Sandhra-Mirella Valdma, Hendik Eerikson;
2. grant the University of Tartu the permit to make the thesis specified in point 1 available to the public via the web environment of the University of Tartu, including via the digital archives, under the Creative Commons licence CC BY NC ND 4.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work from 14/05/2026 until the expiry of the term of copyright;
3. am aware that the author retains the rights specified in points 1 and 2;
4. confirm that granting the non-exclusive licence does not infringe other persons’ intellectual property rights or rights arising from the personal data protection legislation.

Kätryn Varkel

14/05/2026