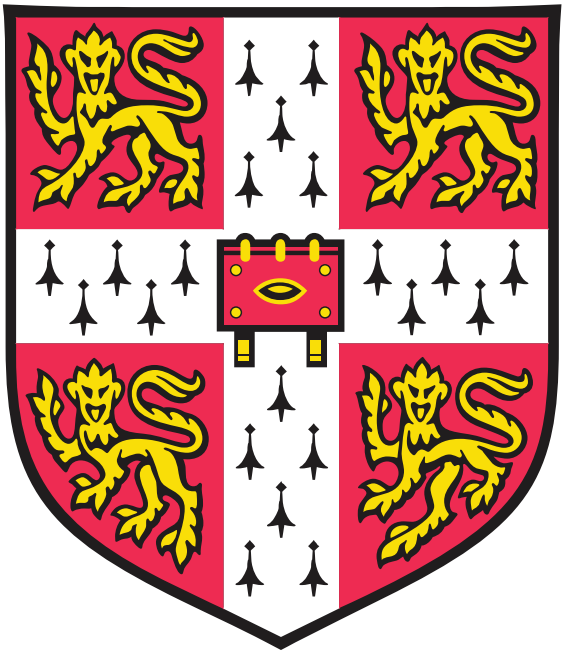
Privacy-Preserving Moving Object Detection

Computer Science Tripos - Part II

Candidate Number



Department of Computer Science and Technology
University of Cambridge

Friday 13 May, 2022

Declaration of Originality

. . . DECLARATION OF ORIGINALITY . . .

Proforma

. . . PROFORMA . . .

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

In the modern world, computers have improved almost every aspect of our lives. Recently, home security has become the latest target of the technology revolution. Companies like Ring [50] and Eufy [17] offer IoT devices like doorbells and cameras to allow their customers to monitor their property 24/7. On top of traditional surveillance, these companies also provide software solutions to monitor the footage recorded by their devices and interpret it. For example, a doorbell may recognise who is at the front door and allow them to enter, or alert the user to the presence of a stranger if it doesn't. However, the computational intensity of these inferences means footage must be transferred from the devices to more powerful servers.

In order to preserve privacy, video is encrypted before it is sent to the server. However, the footage must be decrypted when the inference algorithms are executing. This is an immediate privacy concern. Having the ability to decrypt the footage exposes the opportunity for employees of these companies to access constant surveillance of peoples' homes. The possibilities for exploitation are endless. Malicious actors could use this information to monitor people's location, appraise their belongings, or use the contents of footage for extortion, to name a few. Homomorphic Encryption may provide a solution to this.

Homomorphic Encryption (henceforth HE) is a cryptographic method of encrypting data such that mathematical operations can be performed on encrypted data, or *ciphertext*, itself, rather than on the raw data, or *plaintext*. For example, consider the operation 3×5. In a traditional encryption scheme, the plain values 3 and 5 would be multiplied before encrypting the result. Using a homomorphic scheme, the 3 and 5 can be encrypted, and the ciphertexts multiplied so that when the ciphertext is decrypted, the plaintext is 15. An open question is, can this technique be scaled to more complex algorithms, like those required for surveillance?

More specifically, is it possible to extract the moving objects from a frame of HE video data? Moving object detection, also known as *foreground extraction* or *background subtraction*, is fundamental to modern surveillance systems. Detecting when, for example, somebody enters a property, allows the security systems to alert their owners, possibly pre-empting a break-in. To perform this analysis, the contents of a video must be modelled using a, usually probabilistic, function that allows significant changes in a pixels' value to be discerned. The difficulty of this arises when account-

ing for environmental changes that cause numerical variation, such as light levels when moving from day to night or different weather conditions causing objects to distort.

1.2 Related Work

The lack of privacy caused by constant surveillance is not a new concern. There have been many attempts at solving video inference in the encrypted domain, but none are without flaws. For example, in 2013, Chu et al. [13] proposed an encryption scheme that supports real-time moving object detection, but this was quickly shown to suffer from information leakage, leaving it vulnerable to chosen-plaintext attacks[1]. Similarly, in 2017, Lin et al. [39] proposed a different encryption scheme to achieve the same goal by only encrypting some of the bits in each pixel, but this is unprotected against steganographic[2] attacks. Therefore, while research has been able to solve the weaknesses in privacy, it is yet to offer a solution that also preserves security against adversaries directly attacking the encryption, making them useless to real-world applications.

Likewise, researchers have been investigating inference using HE for many years. In 2012, Graepel et al. [23] introduced machine learning in the HE domain. Dowlin et al. [23] built upon this when they developed the CryptoNets model for deep learning with HE in 2016. However, deep learning neural networks are considered overly complex for moving-object detection. Instead, GMMs are the most widely used technique for background modelling. There is much less research into this area of unsupervised learning within the HE domain. The best example appears to be when, in 2013, Pathak and Raj [48] proposed a HE implementation of a GMM for audio inference. But there does not seem to be any investigations linking HE and GMMs to video analysis.

It appears that the most prevailing explanation for this lack of research is HE's inapplicability to real-time applications, due to its high computational complexity. While this may be true now, it is important to acknowledge that advances in computing capability will reduce the relative difficulty of HE operations. Consequently, more insight into its applicability will become increasingly valuable, as suggested by the trend in the growing popularity of HE research. This dissertation attempts to offer some beginnings to this insight as it attempts to find the constraining limitations of current HE implementations with respect to surveillance.

1.3 Aims and Contributions

This dissertation documents the design and implementation of a potential solution to the questions posed in §1.1, while attempting to follow the constraints restricting the aforementioned real-world systems. In particular, the contribution of the work is:

- The creation of a client-server application simulating the device-server stack utilised by existing products, allowing secure transmission of video data from client to server and back again after performing inference.

[1] A *chosen-plaintext attack* is a scenario in which an adversary can encrypt plaintexts of their choosing, and analyse the corresponding ciphertext in an attempt to break the encryption.

[2] *Steganography* describes the technique of information hiding. Like cryptography, steganography attempts to prevent adversaries from reading messages. Unlike cryptography, where the existence of a message is known but its contents are not, steganography attempts to hide the message's existence.

- The use of Microsoft's Secure Encrypted Arithmetic Library (SEAL) [37] to integrate the CKKS HE scheme [11] for encrypting videos while they are away from the client.
- The implementation of a series of algorithms for enabling private and plain inference of video data to extract moving objects by producing a mask that can be applied to videos in the clear by the client.
- An investigation of Gaussian Mixture Models (GMMs) for HE encrypted background subtraction, beginning with the work by Stauffer and Grimson [56] then moving into more general Expectation-Maximisation GMM algorithms [SOURCE?].
- As an extension, the fabrication of a CKKS implementation from scratch, called MeKKS, based on the Homomorphic Encryption for Arithmetic of Approximate Numbers paper by Cheon et al. [11, 10] to improve understanding of HE.
- A demonstration of the efficacy of the above solutions using timing, accuracy, and *(hopefully)* energy usage data to compare inference of CKKS and MeKKS solutions to plain videos, highlighting the advantages of the MeKKS implementation being targeted to this application over the more generic CKKS.

Chapter 2

Preparation

This chapter discusses the preparatory work done before beginning the project's implementation. §2.1 introduces useful preliminary information and background, §2.2 discusses the methodologies used when approaching the design of the project's implementation, and §2.3 provides an overview of the foundations of the project.

2.1 Preliminaries

This section is dedicated to introducing the fundamental underlying concepts, terminologies, and notations fundamental to this project. The project spans two domains of computer science: cryptography and unsupervised machine learning. These are vast, active research areas. Therefore, for brevity, only concepts essential to understanding this dissertation will be presented - links to further resources may be provided for the sake of brevity.

2.1.1 Threat Model

The design of this project considers the Machine Learning as a Service (MLaaS) framework, in which users first send their data to a server where machine learning inference is performed, and the results are returned. In particular, the transfer of surveillance video data to be analysed by security companies. Suppose that a subscriber to one of these services, Alice, sets up a camera to record activity at her front door. In this scenario, there are two critical threats: (*i*) an adversary, Eve, may eavesdrop on the data while it is being transmitted between the client and server, and (*ii*) an employee, Mallory, of the MLaaS provider may access the data while it is stored by the server, allowing them to monitor the activities of members of a household or observe the contents of a house, exposing risks of theft Figure 2.1 illustrates this succinctly. The first threat can be mitigated easily enough using cryptographic protocols such as TLS [57]. However, the second threat is much more difficult to defend against, particularly because data must usually be decrypted for inference to be performed [4].

Fortunately, the use of HE can mitigate both of these risks. Firstly, HE is a secure cryptographic encryption scheme, so using it to encrypt data during transmission is sufficient to thwart eavesdropping adversaries. Secondly, HE allows computation to

be performed on the data without decryption, so it can prevent the exploitation of plain data.

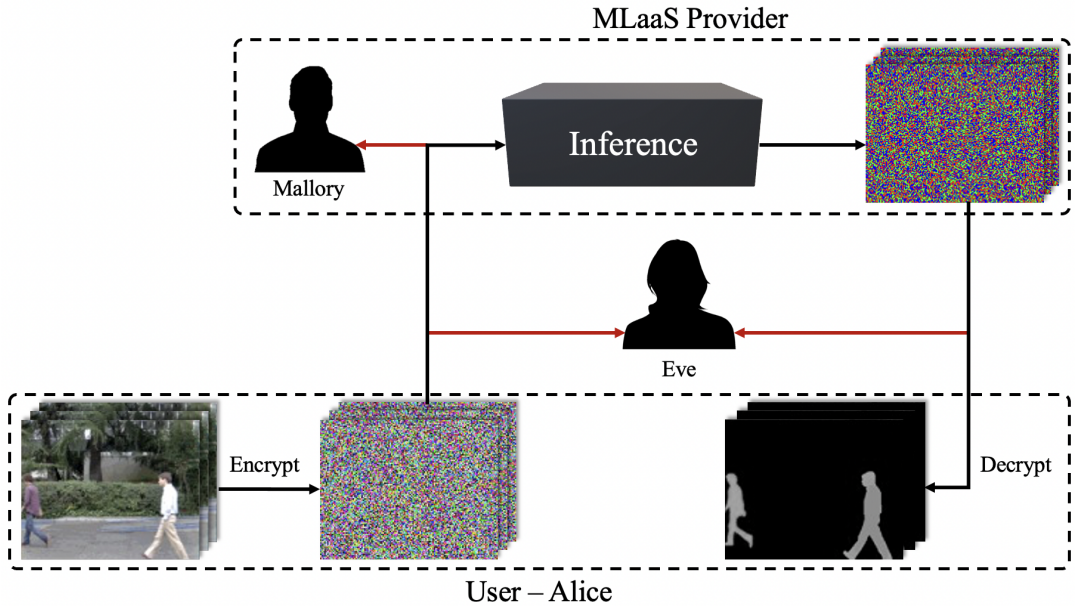


Figure 2.1: A graphical representation of the threat model. Eve is an eavesdropper able to listen to communications between user an service provider. Mallory is a malicious individual within the service provider able to access users' video data. HE is able to obstruct both Eve and Mallory, preventing them from discovering the contents of the user's video.

2.1.2 Homomorphic Encryption

Introduction

There are two broad categories of cryptographic encryption schemes: private-key (symmetric) and public-key (asymmetric). While both can be applied to HE, this dissertation will address public-key encryption because it is the technique adopted by all HE schemes used in the project.

A public-key encryption scheme is defined by a triple of functions $\Pi = (\texttt{KeyGen}, \texttt{Enc}, \texttt{Dec})$. `KeyGen` is a function used to generate a *public key* (`PK`) and *private key*[1] (`SK`) such that $(\texttt{PK}, \texttt{SK}) \leftarrow \texttt{KeyGen}(1^l)$, where the security parameter, l, measures how hard it is for an adversary to break the scheme[2]. Denoting the space of all possible plaintext messages as $\mathcal{M}$ and ciphertext messages as $\mathcal{C}$, a message $m \in \mathcal{M}$ is encrypted into its corresponding ciphertext $c \in \mathcal{C}$ by $c \leftarrow \texttt{Enc}_{\texttt{PK}}(m)$. Similarly c is decrypted back into m by $m \leftarrow \texttt{Dec}_{\texttt{SK}}(c)$.

In order to extend Π into a HE scheme, a fourth function $\texttt{Eval}(f, c_1, \ldots, c_n)$ must be introduced. The evaluation function, `Eval` applies a Boolean circuit, f, to the ciphertext arguments, $c1, \ldots, c_n$ such that, for all arguments, it holds that

$$\texttt{Dec}_{\texttt{SK}}(\texttt{Eval}(f, c_1, \ldots, c_n)) = f(m_1, \ldots, m_n) \tag{2.1}$$

[1] The *private key* is referred to as a *secret key* by some literature. While these terms are equivalent, general convention is to use *secret key* in relation to symmetric encryption, and *private key* when discussing asymmetric - a practice that this dissertation will follow.

[2] An l-bit security parameter would require an expected 2^l attempts to guess the keys.

where $m_1, \ldots, m_n$ are the plaintext equivalents of $c_1, \ldots, c_n$. This is, perhaps, better illustrated by Figure 2.2 below.

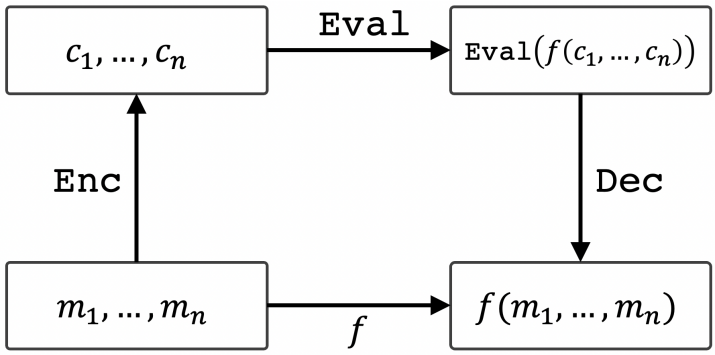


Figure 2.2: Homomorphic Encryption

Theoretically, a *fully* homomorphic scheme[3] allows the evaluation of Boolean circuits indefinitely. However, in practice, the time complexity of operations means many schemes are *levelled* homomorphic schemes. This means they only support operations up to a *bounded depth* - that is, only a predefined number of circuits can be applied to a ciphertext before the plaintext becomes irrecoverable. The maximum depth is a critical factor when applying HE to practical problems because it significantly limits the scope of supported algorithms.

Ring Learning with Errors

For an encryption scheme to be *perfectly secure*, a ciphertext must provide no additional information about its plaintext (see Equation 2.2). In other words, the probability of generating a given ciphertext from a particular plaintext is independent of the plaintext (see Equation 2.3). The two equations below can be shown to be equivalent using Bayes' rule.

$$\mathbb{P}(M = m \mid C = c) = \mathbb{P}(C = c) \tag{2.2}$$

$$\mathbb{P}(C = c \mid M = m) = \mathbb{P}(M = m) \tag{2.3}$$

for all $m \in \mathcal{M}$, $c \in \mathcal{C}$.

While it is possible to create perfectly secure encryption schemes, they are impractical in real applications[4]. Therefore, *computational security* is considered sufficient. Relying on the hardness of certain mathematical problems, computational security means that an encryption scheme is *practically unbreakable*. That is, the most efficient known algorithm for breaking a cipher would require far more computational steps than an attacker would be able to perform, regardless of the hardware available to them. An example of this is the RSA encryption scheme which relies on the fact that there exists no known, efficient algorithm for computing the prime factors of a

[3]The prefix *fully* derives from the existence of *partially* homomorphic schemes. A partially homomorphic scheme will only allow certain operations on the ciphertext - usually multiplication and division - and have existed for many years. Some examples of partially homomorphic schemes include RSA, ElGamal, and Paillier encryption [46].

[4]For example, the One-Time Pad [58]

large number on a classical computer - the integer factorisation problem. In complexity theory, this problem falls into the set of $\mathcal{NP}$.

Similarly, the HE schemes used in this project rely on computational security rather than perfect security. More specifically, they utilise the hardness of the *ring learning with errors* (henceforth RLWE) problem introduced by Lyubashevsky et al. [40]. A polynomial-time reduction from the *shortest vector problem* to RLWE can be derived. Therefore, since the shortest vector problem is $\mathcal{NP}$-hard under the correct choice of parameters, it is safe to rely on RLWE for computational security.

RLWE considers the mathematical objects, *rings*. To understand *rings*, *groups* must first be understood. A group $(\mathbb{G}, \bullet)$ is a set, $\mathbb{G}$, and an operator, $\bullet : \mathbb{G} \times \mathbb{G} \to \mathbb{G}$, such that the following properties hold:

- **Closure**: $a \bullet b \in \mathbb{G}$ for all $a, b \in \mathbb{G}$.
- **Associativity**: $a \bullet (b \bullet c) = (a \bullet b) \bullet c$ for all $a, b, c \in \mathbb{G}$.
- **Neutral Element**: there exists an $e \in \mathbb{G}$ such that for all $a \in \mathbb{G}$, $a \bullet e = e \bullet a = a$
- **Inverse Element**: for each $a \in \mathbb{G}$ there exists some $b \in \mathbb{G}$ such that $a \bullet b = b \bullet a = e$

If $a \bullet b = b \bullet a$ for all $a, b \in \mathbb{G}$, the group is called **commutative** (or **abelian**). If there is no inverse element for each element, $(\mathbb{G}, \bullet)$ is a **monoid** instead.

From this, a *ring* is defined as $(\mathbf{R}, \boxplus, \boxtimes)$, where $\mathbf{R}$ is a set, $\boxplus : \mathbf{R} \times \mathbf{R} \to \mathbf{R}$, and $\boxtimes : \mathbf{R} \times \mathbf{R} \to \mathbf{R}$, such that

- $(\mathbf{R}, \boxplus)$ is an abelian group.
- $(\mathbf{R}, \boxtimes)$ is a monoid.
- $\boxplus$ and $\boxtimes$ are distributive - for all $a, b, c \in \mathbf{R}$, $a \boxtimes (b \boxplus c) = (a \boxtimes b) \boxplus (a \boxtimes c)$ and $(a \boxplus b) \boxtimes c = (a \boxtimes c) \boxplus (b \boxtimes c)$.

If $a \boxtimes b = b \boxtimes a$ then it is a **commutative** ring, but this is not necessary for rings generally. One example of a ring is $(\mathbb{Z}[x], +, \times)$.

Specifically, the RLWE problem is concerned with the ring formed by the set of polynomials modulo $\Phi(X)$ that also have coefficients in $\mathbb{Z}_q$[5]. Known as a *quotient ring*, this can be denoted by $\mathcal{R}_q = \mathbb{Z}[X]/(\Phi(X))$, where $\Phi(X)$ is an *irreducible polynomial* - a polynomial which cannot be factored into two non-constant polynomials.

Informally, RLWE describes the problem of finding an unknown $s \in \mathcal{R}_q$ given a vector of polynomials computed using s and some sampled errors. Consequently, an encryption scheme can be created such that, after encoding a plaintext vector, $\mathbf{v}$, as a list of polynomials and then using a secret polynomial to convert this list to a polynomial, it is infeasible to recover $\mathbf{v}$ in polynomial time.

The HE schemes discussed in this dissertation rely on the RLWE problem to assert *indistinguishable encryptions under a chosen-plaintext attack* (IND-CPA) security. Fundamentally, any encryption scheme is IND-CPA secure if, when attacked by a probabilistic, polynomial-time adversary, the chances of correctly deciding which of two plaintexts a particular ciphertext corresponds to is even. Therefore, in practice, even if an adversary has access to an encryption *oracle* that will encrypt any data an

[5] $\mathbb{Z}_q$ is the set of integers modulo q. For example, $\mathbb{Z}_7 = \{0, 1, 2, 3, 4, 5, 6\}$.

adversary provides, the adversary's chances of calculating the correct plaintext when given a ciphertext should be no better than if they were randomly guessing.

Practically, HE schemes choose a *cyclotonic polynomial* for $\Phi(X)$, where the n-th cyclotonic polynomial, $\Phi_n(X)$, is defined as

$$\Phi_n(X) = \prod_{k \in [1,n];\ gcd(k,n=1)} X - e^{\frac{2i\pi k}{n}} \tag{2.4}$$

In order to speed up computation, n is selected to be an even power of two. Consequently, $\Phi_n(X) = X^{\frac{n}{2}} + 1$. This allows the *number theoretic transform* (NTT)[6]. The advantage of this is that it can be easily accelerated using hardware [34].

The sampled errors used when deriving ciphertexts implies almost exponential growth in error in the number of multiplications applied. The limited depth property of levelled HE schemes is a direct result of this. However, the relative growth size can be reduced by increasing the modulus q. Although this is not without risks. If a polynomial degree of size $n/2$ is used, efficient attacks exist against the RLWE problem for a small value of q [1]. Therefore, a fundamental trade-off is introduced between the supported depth of multiplication and the security level.

2.1.3 Moving Object Detection

Introduction

A discussion of *image segmentation* has been well established in the fields of digital image processing and computer vision research. The problem describes the process of partitioning a digital image into multiple regions, represented as sets of pixels. The goal of segmentation is to simplify the representation of an image so that it is easier to analyse. For example, to locate objects, or boundaries, in an image. More precisely, image segmentation is the process of labelling each pixel of an image such that all pixels sharing a particular characteristic are assigned the same label [54].

There are two broad categories of segmentation techniques: *semantic segmentation* and *instance segmentation*. Semantic segmentation is an approach for grouping objects based on predefined categories. For example, all people in an image may be labelled as "people", all vehicles labelled "vehicles", and all animals labelled as "animals" [24]. In contrast, instance segmentation provides a more refined categorisation of objects. This approach splits each category into separate occurrences. For example, each person in an image may be highlighted distinctly [62].

One application of image segmentation is *foreground extraction*, also known as *background subtraction*, which describes the techniques of segmenting an image into two groups: the foreground and the background, so that further processing can be applied. In order to perform background subtraction, the background of an image must be modelled so that changes in the scene can be detected. However, this can be difficult to achieve. Image data can be very diverse, with factors such as variable lighting, repetitive movements (like leaves, waves, and shadows) making robust models hard to develop.

Once a background subtraction model has been developed, it can be used to detect

[6] A specialisation of the discrete Fourier transform, the NTT is a generalisation of the Fast Fourier transform in the case of finite fields. [6]

moving objects in videos. This is accomplished by comparing the foreground of the current frame to the foreground of a reference frame and extracting the observed differences. There are several methods for achieving this. Below are the five algorithms investigated for this dissertation.

Frame Differencing

The most straightforward moving object detection algorithm, *frame differencing* works by iterating through the frames of a video in a single pass. First, a reference frame must be established as the background. There are several options for this. One approach is to store the first frame of the video. Another, more common option is to compare each frame to the frame directly before it. The advantage of this is that it will evolve, so if a new object is permanently added to the scene, it won't be included in the foreground forever[7]. However, there are disadvantages to this approach if an object is moving slowly enough to overlap with itself across frames. A balance can be found by periodically updating the reference frame or comparing each frame to one several before it, for example.

Each frame can be considered once the reference frame, B, has been selected. Denoting the frame at time t as f_t, the value of each pixel in B, $P(B)$, can be subtracted from the corresponding pixel in f_t, $P(f_t)$. This can be represented mathematically by Equation 2.5.

$$P(F_t) = P(f_t) - P(B) \tag{2.5}$$

where F represents the frames in the resultant video highlighting moving objects.

Mean Filter

A *mean filter* approach to moving object detection attempts to overcome the weaknesses of selecting a reference frame when performing frame differencing. Instead of taking a frame directly from the video, the value of B at time t is calculated using Equation 2.6.

$$B = \frac{1}{N} \sum_{i=1}^{N} f_{t-i} \tag{2.6}$$

where N is the number of preceding images included in the average, and f_t is the frame in the video at time t. N would depend on the video speed and the amount of motion expected in the video.

After B has been calculated at time t, the value of the resultant video F_t can be calculated using the same method as frame differencing, given by Equation 2.5.

Median Filter

Performing moving object detection using a median filter is almost identical to the mean filter method. The difference arises in how the reference frame is calculated. In this approach, the median of the preceding N frames is calculated instead of the mean.

[7] For example, if a fence was added around someone's property or a car was parked in the scene, comparing to a static frame would mean these objects would always be highlighted by frame differencing, despite not moving

Then, like the preceding methods, the moving objects are extracted by subtracting the reference frame from each frame in the video, according to Equation 2.5.

Gaussian Average

Wren et al. originally proposed fitting a Gaussian probabilistic density function to the most recent N frames [60]. Rather than storing a simple reference image that is subtracted from each frame in the video, this method stores a mean and variance value for each pixel in a video frame. The likelihood of a value of a particular pixel occurring can be calculated using this. It is assumed that the most likely value for a pixel will be equivalent to the background of a scene, so if an observed value is sufficiently unlikely according to its Gaussian distribution, it must be because a change has occurred in that portion of the frame. Therefore, that pixel is added to the foreground segment of the video.

A naïve approach to the Gaussian Average method would be to iterate through each frame in the video, calculating the mean and variance for each pixel and evaluating the likelihood from scratch every time. This would have quadratic complexity since, for each frame, N calculations per pixel would have to be performed. A more efficient algorithm would utilise a cumulative function for the mean and standard deviation that can be updated in constant time. Consequently, this would have linear complexity because only a single calculation would need to be performed for each pixel in each frame. This approach would update the mean using Equation 2.7, and the variance using Equation 2.8.

$$\mu_t = \begin{cases} f_0 & \text{if } t = 0 \\ \alpha f_t + (1-\alpha)\mu_{t-1} & \text{otherwise} \end{cases} \tag{2.7}$$

$$\sigma_t^2 = \begin{cases} c & \text{if } t = 0 \\ d^2\alpha + (1-\alpha)\sigma_{t-1}^2 & \text{otherwise} \end{cases} \tag{2.8}$$

where α determines the size of the *temporal window* used to fit the Gaussian model[8], $d = |f_t - \mu_t|$ gives the Euclidean distance from the pixel to the mean, and c is some constant defined by the model creator.

From these models, the foreground can be extracted according to Equation 2.9,

$$\frac{|f_t - \mu_t|}{\sigma_t} > k \tag{2.9}$$

where k is a constant that the model creator can tune to achieve optimal results.

A variant of this method exists where the mean and variance are only updated if a pixel is believed to be in the background. This prevents the model from becoming skewed if there is lots of movement in the frame. However, it has severe limitations. For example, it only works if the image is initially entirely background, and it cannot cope with gradually changing backgrounds.

[8] α acts similarly to a decay factor, weighting the most recent frames as more impactful on the model. Eventually, an old frame will be weighted so insignificantly that its impact is negligible. Therefore, it determines how far into the past the model uses to predict future pixels, hence the 'temporal window'.

Gaussian Mixture Models

Stauffer and Grimson proposed *Gaussian mixture models* (henceforth GMMs) for moving object detection in 1999 [56]. GMMs are probabilistic models that represent the presence of normally distributed subpopulations within an overall population. They are particularly useful because they don't require the subpopulation of a data point to be identified. Instead, subpopulations are learned automatically, constituting a form of unsupervised machine learning.

A simple application of GMMs is in modelling human heights. Typically, two Gaussian distributions will be used: one for males and one for females. Consequently, given just height data, with no gender assignments, it can be assumed that the distribution of all heights should follow the sum of two scaled and shifted Gaussian distributions. A model that can make this assumption on its own (or unsupervised) is an example of a GMM. In general, GMMs can be applied to more than two components.

There are two types of parameters necessary for GMMs, the *component weights*, and the component *means* and *variances*. For a GMM with N components, the i^{th} component has μ_i and variance σ_i in the *univariate case*, and mean μ_i and a covariance matrix Σ_i in the *multivariate case*. The component weights, ϕ_k for component k, are constrained by the equation $\sum_{i=1}^{K} \phi_i = 1$. If the component weights aren't learned, they are known as an *a-priori*[9] distribution over components such that $\mathbb{P}(x \text{ generated by component k}) = \phi_k$. If the component weights are learned, they are known as *a-posteriori*[10] estimates of the component probabilities given the data.

There are several methods for fitting the GMM's parameters. A common method is to use *expectation maximisation* (EM), if the number of components of the GMM is known. When fitting a GMM, the Gaussian distributions are being tuned to match the distributions observed in the data. If all of the data is known, it can all be incorporated into this stage to achieve the most accurate fitting. However, in practice, it is likely that only a subset of the data will be available, so the GMM will then have to extrapolate to the real values. This technique can also be taken advantage of if there is limited time in which to perform fitting.

The EM algorithm is split into two many stages. Firstly, the E-step calculates the expectation of assigning each data point to each component of the GMM, given the GMM's parameters. Secondly, the M-step updates the values of each parameter to maximise the expectations. These two steps repeat until the algorithm converges. Informally, this works because knowing the component assignment for each data point makes solving for the parameters easy, and knowing the parameters makes inferring the probability of a component given the data point easy[11]. Therefore, by alternating which values are assumed known, maximum-likelihood estimates of the unknown values can be efficiently calculated.

Once a GMM has been fitted, it can be used for inference. There are two common uses of inference: *density estimation* and *clustering*. This dissertation will focus on clustering because it is most useful for moving object detection. The posterior component assignment probabilities can be estimated using Bayes' theorem combined with the GMM parameters. Knowing the component that a data point most likely belongs

[9] A probability derived purely through deductive reasoning.

[10] From Bayesian statistics, referring to conditional probability $\mathbb{P}(A|B)$.

[11] The E-step corresponds to the latter case, and the M-step corresponds to the former.

to provides a way to group the points into clusters. In the scenario of moving object detection, this would be two clusters: the foreground and the background.

2.2 Project Strategy

2.2.1 Requirements Analysis

The requirements for this project are listed below. The nature of the project required the development of theoretical knowledge before implementation began, and, as such, the requirements subtly evolved as understanding matured. The original requirements are given in Appendix A for comparison. The final requirements have been grouped into two categories. The first, labelled A, are the core requirements deemed essential to the project's success. The second, labelled B, are considered extensions, aiming to improve understanding of the components used in the core implementation or further the investigation into the applications of HE in surveillance.

Core

`A1:` *Implement a client-server application allowing videos to be homomorphically encrypted and transmitted in both directions.*

This component is vital because it provides the foundation for implementing and integrating all other components. It is also essential to emulate the MLaaS software stack. While conceptually simple, the nature of HE data adds many challenges to transferring videos, forming a significant portion of the investigation into HE applicability.

`A2:` *Implement background subtraction models that can extract moving objects from homomorphically encrypted videos.*

This requires designing and implementing the five moving object detection algorithms detailed in §2.1.3 so that they can act on HE data.

`A3:` *Evaluate the accuracy of HE inference to investigate its applicability to real systems.*

This involves analysing different metrics to understand the efficacy of moving object detection on HE data. Comparisons can be made between inference methods and between plain and encrypted data.

Extensions

`B1:` *Implement a bespoke HE scheme and integrate it into the core application, providing the same functionality as the established scheme already used.*

While this implementation will likely have worse performance than existing implementations, it will offer helpful insight into the inner workings of HE. Also, it may provide opportunities for optimisations for this specific application.

`B2:` *Analyse the security of the encryption schemes used in the project.*

This will be useful in ensuring HE can overcome both security and privacy concerns of existing surveillance solutions and doesn't accidentally introduce insecurities that could allow adversaries to extract information.

`B3`: *Implement an object recognition algorithm acting on HE data using neural networks.*

Implementations like Cryptonets [15] have demonstrated the application of neural networks to HE. This would allow further services offered by surveillance companies to be emulated and reveal a greater insight into the limitations of HE in video analysis.

2.2.2 Methodology

The different stages of the project were best suited to different development methodologies. For the core components, a waterfall development methodology was adopted [51]. The requirements were detailed and unambiguous, so the project lent itself to a structured methodology, not requiring the flexibility of an iterative approach. The stages of the model are detailed below.

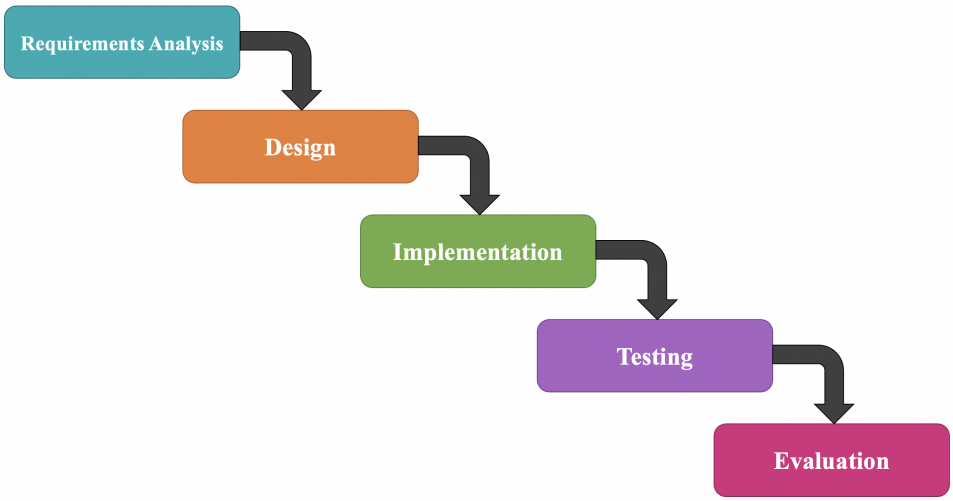


Figure 2.3: The Waterfall Development Model

The results of the *requirements analysis* stage have been detailed in §2.2.1. This stage is where most of the research was performed so that the project's design would be better informed.

The *design* phase incorporates expanding on the requirements into a physical project; this includes, for example, creating the class diagrams shown in Figure 3.3, Figure 3.4, and Figure 3.5.

The *implementation* and *testing* stages were intertwined where possible in order to promote a test-driven approach to development. This was made easier by the object-oriented methodology and unit testing practices (see §2.2.3) adopted during the design stage.

The *evaluation* stage replaces the *maintenance* stage of the traditional Waterfall model because it is unsuitable for this project. This stage involves running experiments to evaluate the project.

When working on the extensions, an iterative model was more appropriate. For example, extension `B1` could be easily split into distinct modules, so they could be developed then tested in a repeating process. Therefore, development transitioned to an Agile model, depicted in Figure 2.4.

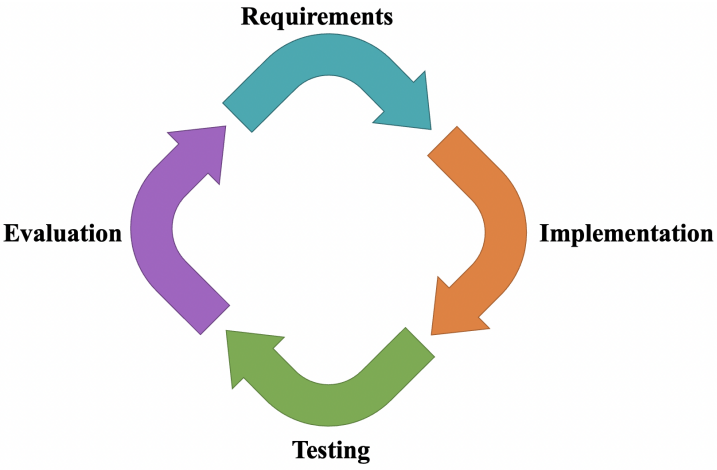


Figure 2.4: The Agile Development Model

A Gantt chart of the project's timeline is shown in Figure 2.5.

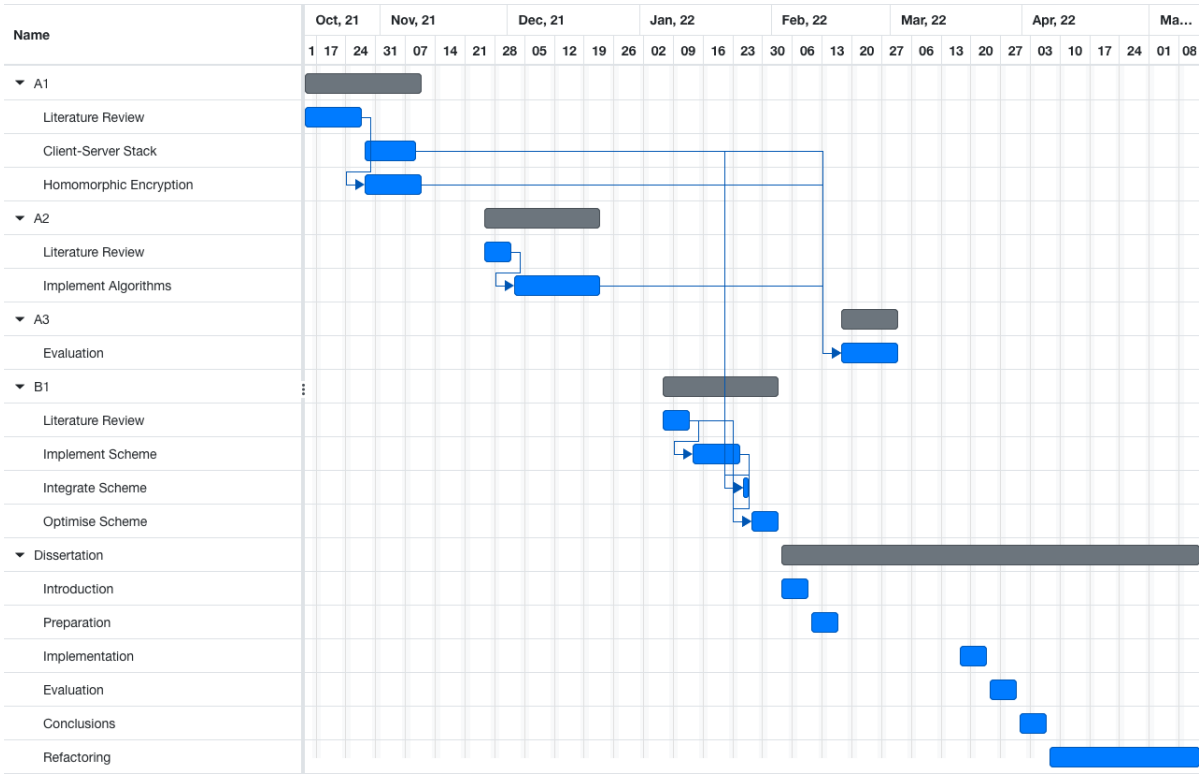


Figure 2.5: Project Timeline

2.2.3 Testing

Challenges

Unlike traditional software engineering, machine learning does not provide precise criteria against which the correctness of an implementation can be verified. The models used for background subtraction are probabilistic, so the outputs cannot be precisely predicted. Consequently, a variety of testing methodologies were required.

Unit Tests

Designed for testing atomic units of source code, unit testing takes advantage of the independent nature of components written following object-oriented principles to test functions in isolation. It does this by providing known expected, boundary, and erroneous data and ensuring the results of a function match the expected. Unit tests can be automated, making them easy to run repeatedly as changes to the source code are made, ensuring errors aren't introduced.

Unit tests were particularly useful when developing a HE scheme from scratch. They verified the correctness of the encoding, encryption, and decryption functions and the HE Boolean circuits automatically and repeatedly.

Integration Tests

While similar to unit tests, integration tests increase the scope of functionality covered by each test. The goal of integration testing is to ensure separate modules interact correctly. Once unit testing has been completed, these tests take the verified modules, group them into larger aggregates, and provide expected, boundary, erroneous data to ensure the output is correct.

Integration testing was useful in verifying that the software stack functioned correctly. For example, ensuring the client and server communicated correctly. While some integration testing can be automated, more complex engineering work was prioritised over creating a more comprehensive testing suite, so manual integration testing was the primary technique used.

Manual Verification

Manual verification was used to overcome the challenges of testing the background subtraction models. Since the project involves video data, human inspection provides a good intuition of whether or not a background has been correctly removed. If a more detailed analysis is required, pixel values can be compared to check for expected results, or verify consistency across multiple tests[12].

2.3 Starting Point

2.3.1 Knowledge and Experience

Prior to beginning the project, the following Tripos courses that considered similar themes had been completed: *Scientific Computing*, *Machine Learning and Real-world Data*, *Software and Security Engineering*, *Concurrent and Distributed Systems*, *Data Science*, *Computer Networking*, and *Security*. The Part II course, *Cryptography* was also useful in understanding the theoretical underpinnings of encryption.

However, it should be noted that HE is not included in the scope of the Cryptography course, so the theory was learned independently of Tripos studies. The study of applied HE is sparsely documented, particularly with modern schemes. Therefore,

[12]While humans are able to perform this verification, simple Python scripts are usually written to make testing more efficient.

most understanding came from academic papers; notably [11] and [37]. Although, articles such as [8] were more useful for foundational knowledge.

Similarly, there was little mention of computer vision artificial intelligence in Tripos, so most understanding came from independent research. For example, academic papers such as [56] and [36] were helpful, particularly when considering privacy-preserving computer vision. Some understanding also came during a summer internship completed in the field of object recognition deep learning.

2.3.2 Tools Used

Programming Languages

All of the code written for this dissertation was written in *Python* [20]. The main reasons for this were the large machine learning ecosystem, ease of use, and ease of debugging. Consequently, it was best suited to the project's tight schedule since it should allow for quick implementation.

However, it must be acknowledged that Python is not a language traditionally used for cryptographic applications. Usually, lower-level, faster languages like C++ are favoured. Since the original focus of the project was investigating the efficacy of moving object detection in the HE domain, the speed of execution was not prioritised over the speed of implementation.

Software Development

Visual Studio Code [44] development environment was used for writing code because of support for Python as well as a wide variety of plugins that allow integration of other valuable tools such as ESLint. In addition, *Git* [21] and *GitHub* [27] were used for version control and source code management, as well as storing a backup of source code. *OneDrive* [43] was also used to hold another backup, for safety.

Encryption Schemes

The project focuses on HE schemes based on the RLWE problem. The main reason for this was the abundance of academic literature discussing them. In particular, the CKK scheme [11] was selected because it supports representing real numbers[13]. However, the project is designed in such a way that any HE encryption scheme can be substituted in CKKS's place, as long as it follows the same API.

Libraries

The project uses Microsoft's SEAL library [37], which provides a C++ implementation of the CKKS scheme. This was chosen because of the extensive optimisations that have been applied. In particular, SEAL uses a residue-number-system variant of CKKS to support large plaintext moduli. The SEAL API was integrated using a Python wrapper library [25].

[13] As opposed to, for example, the BFV scheme, which only supports integers [7, 18].

Datasets

There were two publicly available datasets used in this project:

- The Moving-MNIST dataset contains ten thousand sequences each of length twenty frames showing two handwritten digits from the standard MNIST dataset moving in a 64×64 pixel frame. This is a relatively simple dataset to perform moving object detection on because it only contains white objects on a black background. Therefore, it was useful in providing a baseline for the performance of the inference algorithms. [59, 38].

- The LASIESTA dataset contains a variety of sequences showing objects moving across static backgrounds in a range of conditions. Specifically designed to evaluate segmentation algorithms, this dataset provides a more realistic example of surveillance video. Therefore, this dataset can provide a truer evaluation of moving object detection in the HE domain. [26].

Licensing

All software dependencies in this project use permissive libraries that allow their code to be used without restrictions. The same is true for the datasets. Table 2.1 gives the specific licenses.

Dependency	Licence

`Multiprocessing`
`NumPy`
`SymPy`
	3-Clause BSD [28]

`Microsoft SEAL` `SEAL-Python`	MIT [29]

`Matplotlib`
`Python`
`Tkinter`
	PSFL [19]

Table 2.1: Licenses

2.3.3 Computer Resources

The original project proposal mentioned that external computational resources might have been required during the implementation phase, such as AWS or Microsoft Azure. However, the project was entirely developed, tested, and evaluated on a MacBook Pro laptop. The specifications are listed below, in Table 2.2.

Processor	
CPU	8 Cores
GPU	14 Cores
Neural Engine	16 Cores
Memory Bandwidth	200 GB/s
Memory	
RAM	32 GB (unified memory)

Table 2.2: Computer Specifications

Chapter 3

Implementation

3.1 Architecture

This section is dedicated to detailing the high-level architecture and design of the project. It will discuss the purpose of each component and the software engineering principles applied to make design choices.

3.1.1 Framework

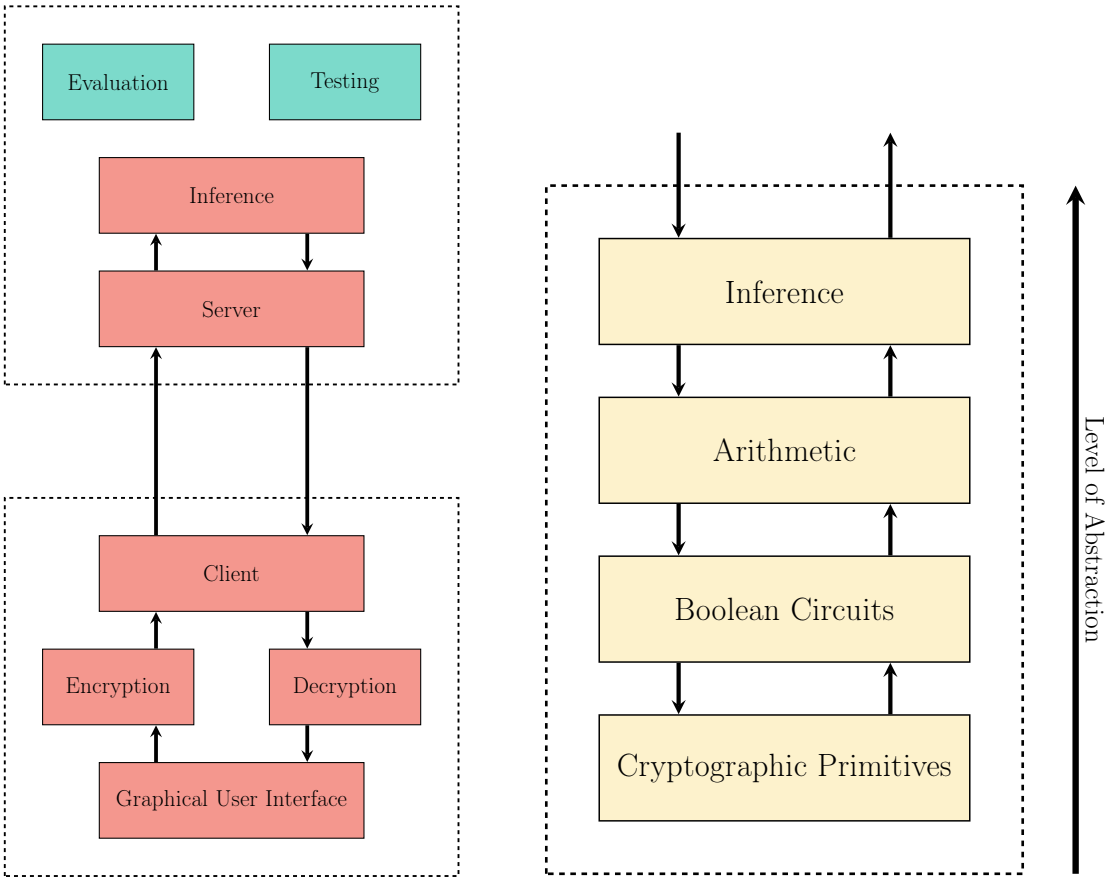
The project's design began with an abstract framework around which the individual components were designed. Based on the threat model outlined in §2.1.1, Figure 3.1a depicts a high-level layout of the core components. The server-side application has been split into two categories: *online* in red and *offline* in blue. Online describes inference being performed directly in response to a request from the client. Offline describes generating inference results on batches of data, independent of the front-end.

- The online components are where most of this dissertation's discussion occurs. This portion of the application is responsible for emulating the MLaaS model. The *GUI* allows users to select the encryption scheme and inference method used. The user's video is then passed to the *encryption* component, responsible for encrypting data using the selected scheme - either CKKS or MeKKS in the current implementation. The resulting encrypted data is then passed through the *client* to the *server*. In the *inference* component, the received data is privately analysed, and a video containing only the moving objects is returned to the *client* via the *server*. The *decryption* component must then decrypt the inference results and the video played for the user.

- The offline components are intended for use before the application is deployed. The framework's *testing* component refers to developing and refining the inference algorithms used to extract moving objects. The *evaluation* component encompasses the process of evaluating the application, including both inference performance and client-server activity.

Figure 3.1b provides a deeper insight into the composition of the inference component. The scope of this project only considers the layers above encryption primitives. However, it is important to note that lower layers of abstraction exist. In particular, a layer that may be particularly relevant to the investigation begun by this dissertation

is the hardware implementation. Hardware modifications could potentially impact the application's performance considerably, both positively and negatively. For example, accelerators, such as GPUs, could be used to perform cryptographic operations [3]. Equally, the hardware used in current surveillance implementations may produce weaker results.



(a) The project components. *(b) Abstract layers of the inference stack.*

Figure 3.1: High-level implementation overview.

3.1.2 Software Interface

An overview of the project's repository is given in Figure 3.2. The project was written to clearly distinguish the layers depicted in Figure 3.1. The object-oriented approach to design allowed separate components to be implemented independently. As well as aiding comprehension, this architecture was chosen to minimise interaction across abstraction layers, and make the project straightforward to expand with, for example, more HE schemes or inference methods.

The application can be split into four layers of abstraction, from the high-level interface to the low-level implementation.

- The highest level is the graphical user interface that the user directly interacts with. It allows the user to configure the encryption scheme and inference method used by the server, and upload and receive videos.

- The next layer contains the networking functionality of the application. Managed by the `connection` files in both the `client` and `server` packages, this layer is responsible for passing any data between the client and the server.
- The third layer establishes the API for the cryptographic principles. The HE functionality required by the application is contained within this layer so that the particular scheme in use can be substituted without changes to the above layers.
- The lowest level contains the cryptographic primitives. Contained within the `lib` folder, the libraries `Seal-Python` and `MeKKS` contain the implementations of these primitives so that the preceding levels may use them.

3.1.3 Class Structure

This section provides a more verbose insight into the project's structure. Figure 3.3 details the arrangement of the client-side, Figure 3.4 contains the classes composing the server-side, and Figure 3.5 depicts the structure of the MeKKS library. While there is overlap in these class diagrams, they have been separated into three figures for the sake of clarity.

3.2 Encryption

This section briefly discusses the fundamentals of the CKKS HE scheme, as described in the original paper by Cheon et al. [11], and how it is implemented in this project. Also, to provide further insights and an investigation in specialising HE schemes, a bespoke CKKS implementation is detailed, called MeKKS.

3.2.1 CKKS Primitives

Fundamentally, the CKKS scheme encrypts plaintext polynomials into ciphertext polynomials. Addition and multiplication operations can then be performed on the data, introducing uncertainty to the values. Finally, ciphertext polynomials can be decrypted to retrieve approximations of plaintext polynomials, with the accuracy depending on the number of operations performed.

Therefore, to encrypt vectors of real values, they must be first be encoded as polynomials in the ring $\mathcal{R} = \mathbb{Z}[X]/(X^N+1)$, where N is a power of 2. During encoding, the real values must be rounded. To preserve precision, the vector is multiplied by a *scaling factor*, Δ. Once the vector has been encoded into a polynomial in $\mathcal{R}$, it can be encrypted into a *pair* of polynomials: the ciphertext.

Consider two vectors $\mathbf{v}$ and $\mathbf{w}$. To encode these vectors, they are scaled to $\Delta\mathbf{v}$ and $\Delta\mathbf{w}$. They can then be encrypted and multiplied to give ciphertext equivalents of $\Delta^2(\mathbf{v} \bigodot \mathbf{w})$. It is clear from this that a sequence of multiplication operations will continue to increase the scale factor indefinitely. To overcome this, CKKS introduces a *rescaling* procedure, which can be understood as dividing the ciphertext by Δ to reduce the scale.

However, rescaling is not a free operation, so it cannot be continuously applied to

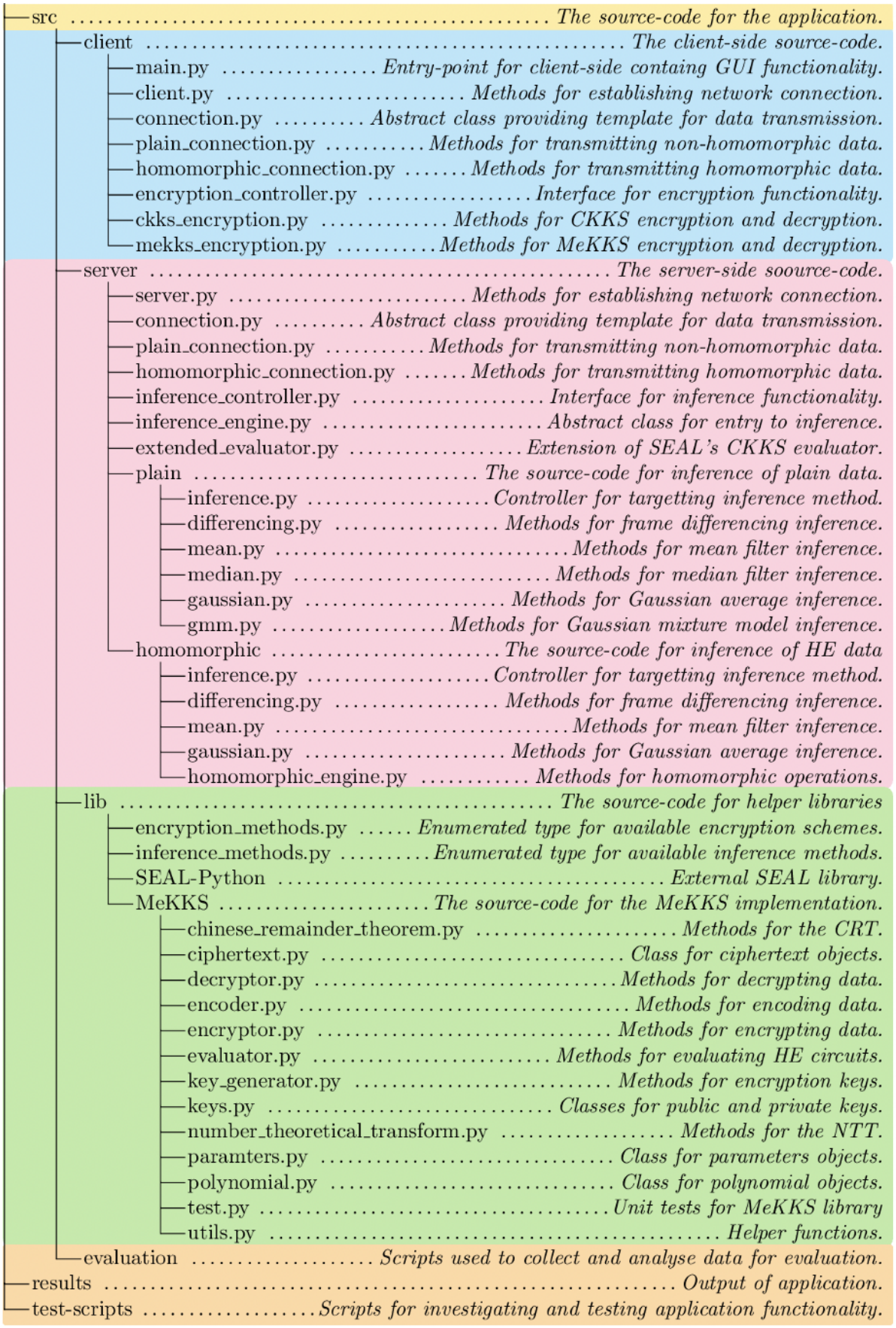


Figure 3.2: Repository Overview.

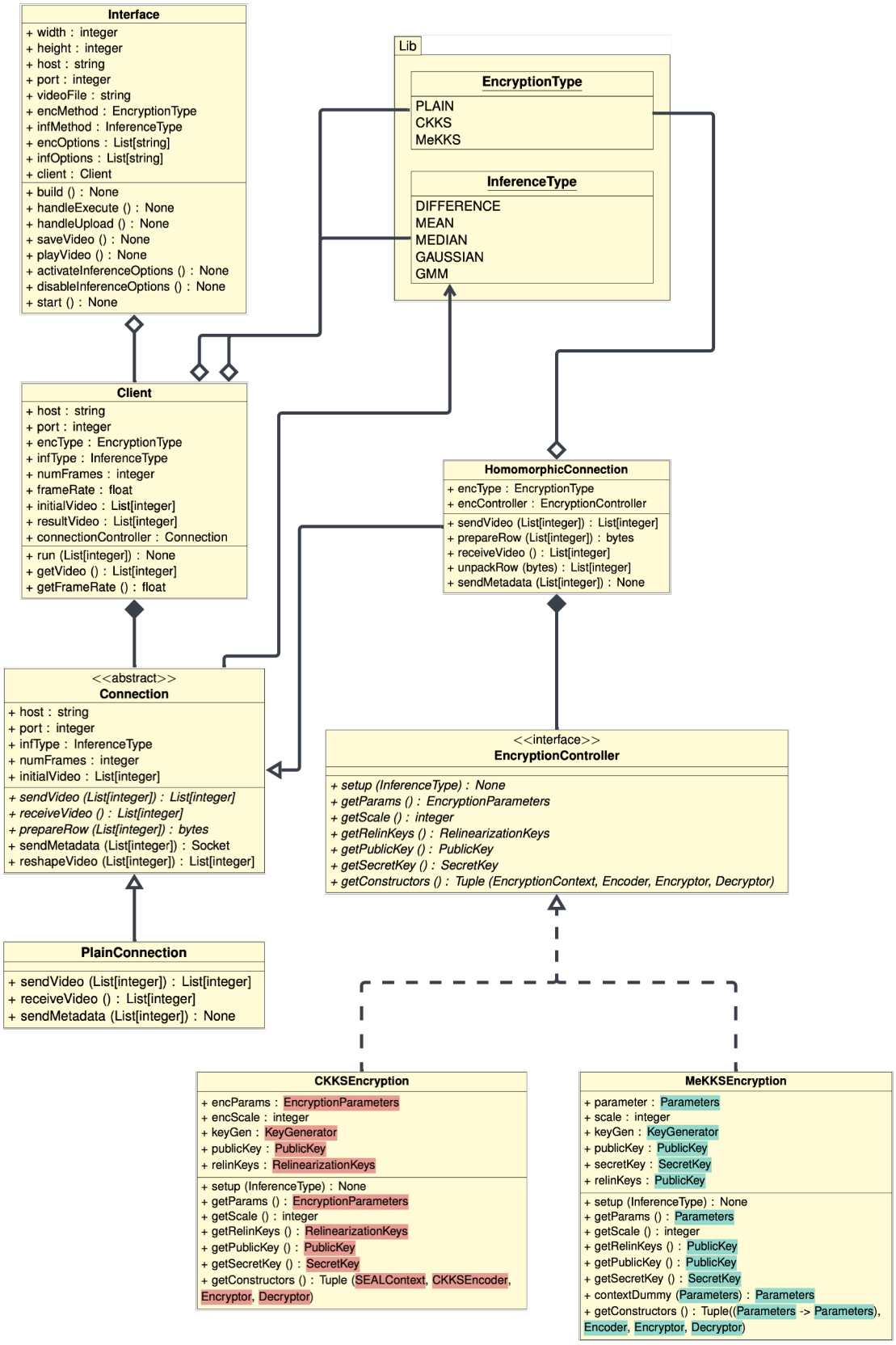


Figure 3.3: UML Class Diagram showing the components of the client side.

External CKKS classes:

External MeKKS classes:

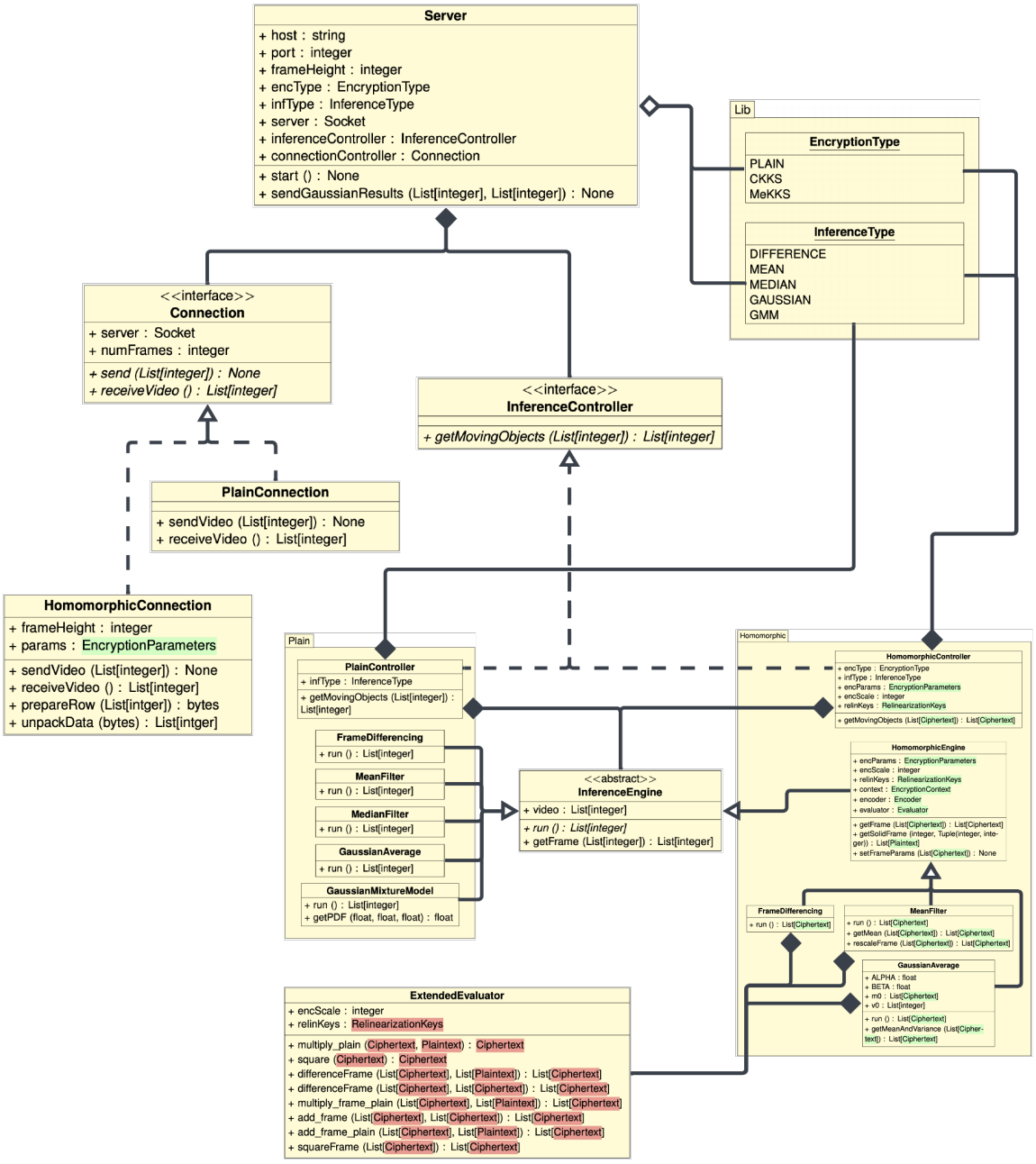


Figure 3.4: UML Class Diagram showing the components of the server side.

External CKKS classes:

Abstract HE objects:

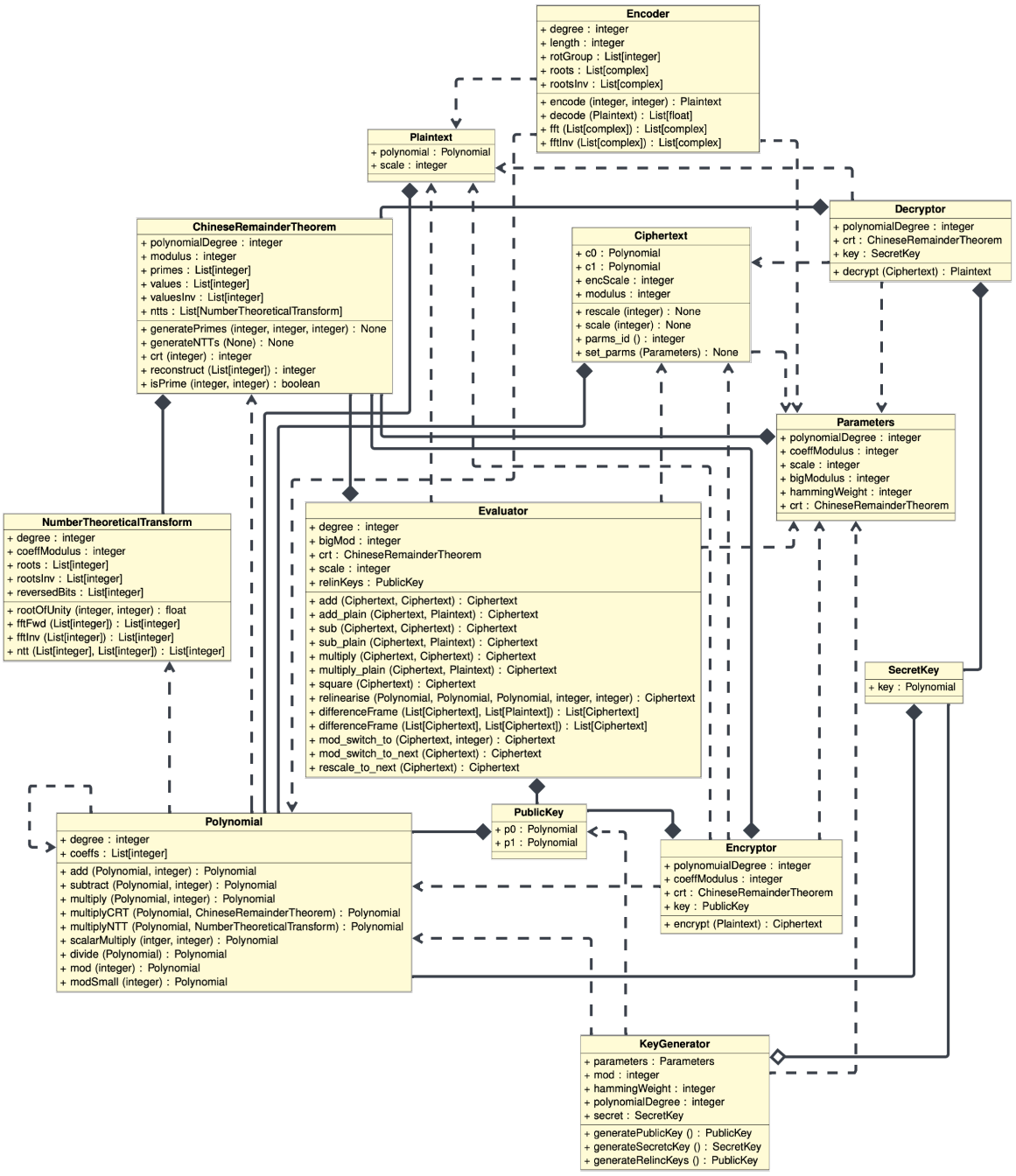


Figure 3.5: UML Class Diagram showing the components of the MeKKS encryption scheme.

allow unlimited multiplications. CKKS is a levelled HE scheme, so each ciphertext resides at a discrete level. Each level has a coefficient modulus q that dictates a ciphertext's coefficients must be in $\mathbb{Z}_q$. When a polynomial is first encrypted, it exists in the maximum level, L, with coefficient modulus $Q = q_0\dot{\Delta}^L$, for a *base modulus* q_0. When a ciphertext is rescaled, the coefficient modulus is divided by Δ, reducing it to $q_0 \cdot \Delta^{L-1}$. Hence, the ciphertext is lowered to the next level. If a ciphertext reaches level 0, no further multiplications can be applied, to preserve the encrypted values. For correct decryption, the coefficients of a polynomial cannot exceed q_0. Consequently, in practice, q_0 is much larger than Δ.

Encoding and Decoding

In CKKS, the plaintext message space is the *cyclotomic polynomial ring* $\mathcal{R} = \mathbb{Z}[X]/\Phi_M(X)$, where $\Phi_M(X)$ is the M-th cyclotomic polynomial and M is a power of two. Encoding is the process of mapping a complex vector to an element in $\mathcal{R}$, and decoding is the process of reversing this mapping.

Another way of defining decoding is as a mapping of an element $r \in \mathcal{R}$ to a vector in $\mathbb{C}^N$ using the embedding $\sigma : \frac{\mathbb{R}[X]}{X^N+1} \to \mathbb{C}^N$. This is applied to each root of $\Phi_M(X)$ through the evaluation of r as

$$\sigma(r) = (r(\xi), r(\xi^3), \ldots, r(\xi^{M-1})) = (r(\xi^k) \mid k \in \mathbb{Z}_N^*) \in \mathbb{C}^N \tag{3.1}$$

where $\xi = e^{\frac{2\pi i}{M}}$ is a primitive M-th root of unity.

However, to establish a one-to-one mapping, the input must be *scaled* and *rounded* during encoding, and half of the complex vector must be *discarded* during decoding.

As a result of the above, the CKKS encoding function has the form given in Equation 3.2.

$$\texttt{Encode}(\mathbf{v}, \Delta) : \mathbb{C}^{\frac{N}{2}} \times \mathbb{Z}^+ \to \mathcal{R} \tag{3.2}$$

Operations

Figure 3.6 lists the operations supported by CKKS and their definitions. An important observation is the multiplication of polynomials required during ciphertext multiplication. This makes multiplication a much more computationally expensive operation than addition.

Rescaling

The rescaling function introduced above is defined by equation 3.3.

$$\texttt{Rescale}(\mathbf{c}, \Delta^{l_{\text{new}}}) : \mathcal{R}_{ql}^2 \times \mathbb{Z}^+ \to \mathcal{R}_{ql}^2 = \lfloor \Delta^{l_{\text{new}}-l} \cdot \mathbf{c} \rceil \mod q_{l_{\text{new}}} \tag{3.3}$$

Rescaling truncates some of the least-significant bits of a ciphertext by dividing by a power of the scaling factor. Practically, this is performed after every multiplication to revert the resulting Δ^2 back down to just Δ.

Rotations

An advantage of RLWE based HE schemes over others is the ability to *rotate* ciphertext slots. Given a vector $\mathbf{V} = [v_0, v_1, v_2, \ldots, v_n]$ and an offset d. A rotation

- `KeyGen` : sample $s \leftarrow \chi_{\text{key}}$, $\mathbf{r}' \rightarrow \mathcal{R}_{qL}$, $\mathbf{r}' \rightarrow \mathcal{R}_{P \cdot qL}$, $\mathbf{e} \leftarrow \chi_{\text{err}}$, and $\mathbf{e}' \leftarrow \chi_{\text{err}}$.
  1. Calculate $\mathbf{a} := -\mathbf{rs} + \mathbf{e} \mod q_L$ and $\mathbf{a}' := -\mathbf{r}'\mathbf{s} + \mathbf{e}' + P\mathbf{s}^2 \mod P \cdot qL$.
  2. Return $\texttt{SecretKey} := (1, \mathbf{s})$, $\texttt{PublicKey} := (\mathbf{a}, \mathbf{r})$, $\texttt{EvaluationKey} := (\mathbf{a}', \mathbf{r}')$.
- $\texttt{Enc}_{\texttt{PublicKey}}(\mathbf{m})$: sample $\mathbf{v} \leftarrow \chi_{\text{enc}}$ and $\mathbf{e}_0, \mathbf{e}_1 \leftarrow \chi_{\text{err}}$.
  1. Calculate $\mathbf{c}_0 := \mathbf{v} \cdot \mathbf{a} + \mathbf{m} + \mathbf{e}_0$ and $\mathbf{c}_1 := \mathbf{v} \cdot \mathbf{r} + \mathbf{e}_1$.
  2. Return $(\mathbf{c}_0, \mathbf{c}_1) \mod q_L$.
- $\texttt{Dec}_{\texttt{SecretKey}}(\mathbf{c}_0 + \mathbf{c}_1 \cdot \mathbf{s}) = (\mathbf{c}_0 + \mathbf{c}_1) \mod q_l$.
- $\texttt{Add}((\mathbf{c}_0, \mathbf{c}_1), (\mathbf{d}_0, \mathbf{d}_1)) = (\mathbf{c}_0 + \mathbf{d}_0, \mathbf{c}_1 + \mathbf{d}_1) \mod q_l$.
- $\texttt{AddPlain}((\mathbf{c}_0, \mathbf{c}_1), x) = (c_0 + \texttt{Enc}_{\texttt{PublicKey}}(x), \mathbf{c}_1) \mod q_l$.
- $\texttt{Multiply}((\mathbf{c}_0, \mathbf{c}_1), (\mathbf{d}_0, \mathbf{d}_1)) = (\mathbf{r}_0, \mathbf{r}_1) + (\lfloor P^{-1} \cdot \mathbf{r}_2 \cdot \texttt{EvaluationKey}_0 \rceil, \lfloor P^{-1} \cdot \mathbf{r}_2 \cdot \texttt{EvaluationKey}_1 \rceil) \mod q_l$.
- $\texttt{MultiplyPlain}((\mathbf{c}_0, \mathbf{c}_1), x) = (\mathbf{c}_0 \cdot \texttt{Enc}_{\texttt{PublicKey}}(x), \mathbf{c}_1 \cdot \texttt{Enc}_{\texttt{PublicKey}}(x)) \mod q_l$.

Figure 3.6: A list of the basic operations supported by the CKKS scheme. P is a large scaling factor and $\lfloor \cdot \rceil$ denotes rounding to the nearest integer. Recreated from [11].

would cyclically shift each element by d indices[1]. This is achieved by homomorphically performing a *Galois automorphism* on the ciphertext using standard results from *Galois theory*. However, these operations are significantly the most expensive operations offered by RLWE-based schemes [32].

RNS Optimisations

SEAL utilises a *residue number system* [9] to overcome the slow computations caused by very large polynomial coefficients requiring arbitrary precision arithmetic [RNS]. This exploits the Chinese Remainder Theorem (CRT) to decompose the coefficients in $\mathbb{Z}_q$ into n smaller ones: $\mathbb{Z}_{p_1}, \ldots, \mathbb{Z}_{p_n}$. To do this, the CRT uses a *moduli switching chain* $p_1, \ldots, p_n$ such that $\prod_i p_i = q$ and each p_i is a prime number requiring fewer than 64-bits to store. Thanks to hardware limitations, it is faster to compute the n separate multiplications in $p_1, \ldots, p_n$ than it is to compute a single multiplication in q. The results of the separate multiplications can be easily combined thanks to the isomorphism between $\mathbb{Z}_q$ and $\mathbb{Z}_{p_1}, \ldots, \mathbb{Z}_{p_n}$.

Unfortunately, as a result of the RNS optimisation, $q_l = q_0 \Delta^l$ cannot always be maintained due to the difficulty of finding every q_l as the product of n primes. To overcome this, SEAL instead defines $q_l = q_0 \prod_{i=1}^{l} p_i$ where q_0 must be prime. Where the original CKKS scheme divides by Δ to rescale, SEAL divides by p_l. Consequently, the resulting scaling factor is only approximately equal to Δ. Therefore, in practice, the

[1] For example, given $\mathbf{v} = [1, 2, 3, 4, 5]$ and $d = 2$ would produce $\mathbf{v}' = [4, 5, 1, 2, 3]$.

scaling factor must be manually reset to Δ after each multiplication and rescaling[2], and primes are chosen to be approximately the same size as Δ.

3.2.2 MeKKS

Initially, the goal of implementing the CKKS scheme from first principles was to understand further the fundamental mathematical principles that allow it to work. There were little to no expectations of providing a performance improvement for the reasons detailed in §3.2.2. However, during the implementation phase, it became apparent that there may be some advantages to a specialised HE scheme that can overcome weaknesses found in SEAL during the earlier stages of development.

Limitations

The primary limitation of MeKKS was the language used for implementation. Traditionally, encryption schemes are written in C or C++ because they are lower-level than most languages, so they run more efficiently. Consequently, the more computationally expensive operations (such as the multiplications between large prime numbers required by CKKS, detailed above) would run much quicker than in Python.

Another limitation was the number of optimisations that could be applied. Since the focus of the implementation was on understanding, the implementation began with the foundational CKKS scheme. However, since this scheme was released, many optimisations have been produced. For example, using residue number systems, bootstrapping extensions, or reduced approximation error [9, 10, 33]. The opportunities for developing MeKKS are limitless. However, it was essential to define a clear endpoint for the implementation for the sake of scheduling the project and the inflexible final deadline. Therefore, the performance expectations compared to SEAL - which has had years of development from a team of experts - were further bounded.

Implementation

The first iteration of MeKKS implementation began using the scheme outlined by Cheon et al. in 2016. This allowed all of the functionality described in §3.2.1 to be implemented, and the API constructed following SEAL to allow for easy integration into the core application.

However, the performance of this implementation meant it was infeasible to integrate into the rest of the application. Therefore, the next iteration of the HEAAN paper was used to add a bootstrapping procedure, dramatically speeding up computation [10]. Taking advantage of the *approximate computation* characteristic of this encryption scheme, the bootstrapping approach aims to evaluate the decryption formula approximately so that an encryption of the original message can be obtained in a large ciphertext modulus. Hence, an approximation of the *modular reduction* formula is implemented such that it can be efficiently evaluated using standard HE arithmetic operations - where the error induced by the approximation is small enough to maintain precision.

[2]Setting the scaling factor to Δ can be achieved by multiplying by a constant close to 1, or using a ciphertexts `scale` function if Δ is known.

Using the observation that the modular reduction function, $F(t) = [t]_q$, is the identity nearby zero and periodic in q, bootstrapping uses a trigonometric function to approximate the function when $t = \langle \texttt{Ciphertext}, \texttt{SecretKey} \rangle$ is close to a multiple of q (the ciphertext modulus). Specifically, using the *sine* function in the formula given by Equation 3.4.

$$[\langle \texttt{Ciphertext}, \texttt{SecretKey} \rangle]_q = \frac{q}{2\pi} \cdot \sin\left(\frac{2\pi}{q} \cdot \langle \texttt{Ciphertext}, \texttt{SecretKey} \rangle\right) + O(\epsilon^3 \cdot q) \quad (3.4)$$

when $F(\langle \texttt{Ciphertext}, \texttt{SecretKey} \rangle) < \epsilon \cdot q$.

The Taylor polynomial can be used to approximate the trigonometric function so that it can be calculated in the HE domain. The input, t, is bounded by $K \cdot q$ for some constant $K = O(\lambda)$, where λ is the security parameter. The degree of the Taylor polynomial should be at least $O(K \cdot q)$ in order to make the error term small enough on the interval $(-K \cdot q, K \cdot q)$. Cheon et al. proposed using the *Paterson-Stockmeyer* method to reduce the complexity of these calculations [10, 47]. However, the complexity of recryption grows exponentially with the depth of the decryption circuit, which still has a substantial impact.

Figure 3.7 provides a graphical representation of the bootstrapping approximation.

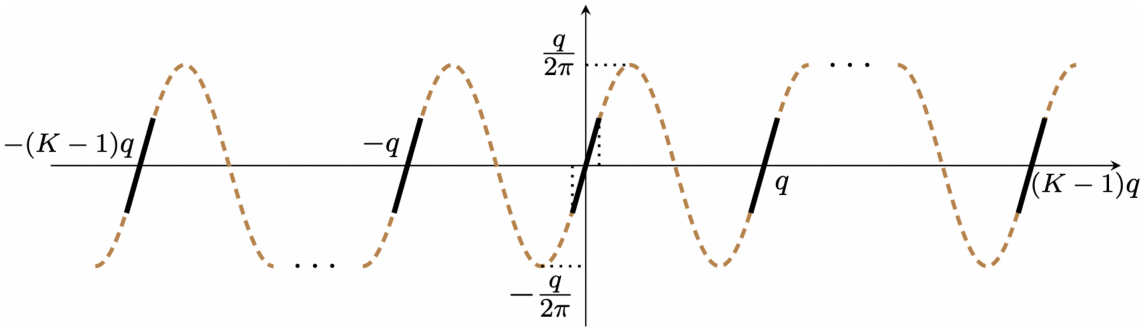


Figure 3.7: Modular Reduction and Scaled Sine Functions. Reproduced from [10].

Optimisations

The principal optimisations of MeKKS came through the methods of data representation. When integrating SEAL into the project, a Python wrapper for a C++ implementation was used. Consequently, the application was forced to handle large C++ objects, hidden by a further layer of abstraction via the wrapper. Therefore, Python struggled to manipulate these objects efficiently, specifically during the networking stages of execution. As explained in §**??**, the bottleneck for SEAL's performance was in data serialisation. However, since MeKKS was written in Python, the underlying functionality provided much more efficient manipulation as it could directly interact with the objects and their attributes.

Another optimisation came through the specialisation of the library. The project planning ensured MeKKS was only implemented once the application's core had been complete. As a result, the functionality required had been investigated and almost wholly finalised. Therefore, only the components needed for each class in MeKKS

were implemented. This further reduced the size of objects and removed any unnecessary computations, making execution more efficient. For example, the most expensive operations offered by SEAL are rotations. However, since the application did not use them, they were ignored during the implementation of MeKKS.

3.3 Networking

This section is dedicated to detailing the techniques investigated for tackling the networking portion of the project. An established limitation of HE is its impact on memory usage [42]. Consequently, the *transmission time* of data is significantly impaired. Given a video file, the *transmission time* can be defined by Equation 3.5.

$$\text{transmission time} = \frac{\text{video size } \textit{(bytes)}}{\text{transmission rate } \textit{(bytes/second)}} \tag{3.5}$$

The core design of this project focused on emulating the MLaaS model employed by surveillance technology companies. Therefore, transferring large volumes of data to the cloud is a critical component, so a substantial portion of the investigation was considering how to reduce the effect of HE on uploading videos. The problem was considered from two angles: attempting to reduce the *video size* (see §3.3.1 and §3.3.2) and attempting to increase the *transmission rate* (see §3.3.3).

3.3.1 Seam Carving

Developed by Avidan and Shamir in 2007, *seam carving* describes a method of resizing images using *geometric constraints* while also considering *image content* [2]. The advantage of accounting for both aspects is that an image can be resized to some target dimensions while preserving important features, such as people or buildings. There are two categories of methods for distinguishing these features. Firstly, *top-down* methods use tools such as *face detectors* to highlight were the features appear in the image [31]. Whereas a *bottom-up* approach uses *saliency maps*[3] to locate the most important [35].

However, instead of focussing on the most critical pixels of an image, seam carving targets the least important - those that "won't be noticed" if removed. To do this, it defines an *energy function* for each pixel in an image. The original paper proposes multiple energy functions, beginning with the function shown in Equation 3.6 before developing Equation 3.7.

$$e(\mathbf{I}) = \left|\frac{\partial}{\partial x}\mathbf{I}\right| + \left|\frac{\partial}{\partial y}\mathbf{I}\right| \tag{3.6}$$

$$e_{HoG}(\mathbf{I}) = \frac{\left|\frac{\partial}{\partial x}\mathbf{I}\right| + \left|\frac{\partial}{\partial y}\mathbf{I}\right|}{\max\left(HoG(\mathbf{I}(x,y))\right)} \tag{3.7}$$

where $HoG(\mathbf{I}(x,y))$ is a histogram of oriented gradients at every pixel. The paper recommended an eight-bin histogram over an eleven-pixel square window around each pixel. Figure 3.9b depicts the application of an energy function to an example image.

[3] a representation highlighting the regions of an image where a persons' eyes are first drawn, see [55].

Once the energy of each pixel has been calculated, the image can be split into *seams*. A *vertical seam* is a path of pixels connecting the top of an image to the bottom, such that there is only a single pixel from each column in the path. A *horizontal seam* is a path of pixels connecting the left of an image to the right, such that there is only a single pixel from each column in the path. Formally, this is defined by Equation 3.8a and Equation 3.8b for vertical and horizontal seams respectively.

$$\mathbf{s^x} = \{s_i^x\}_{i=1}^n = \{(x(i), i)\ s.t.\ \forall i\ |x(i) - x(i-1)| \leq 1\}_{i=1}^n \tag{3.8a}$$

$$\mathbf{s^y} = \{s_j^y\}_{j=1}^m = \{(j, y(j))\ s.t.\ \forall j\ |y(j) - y(j-1)| \leq 1\}_{j=1}^m \tag{3.8b}$$

For an $n \times m$ image, $\mathbf{I}$, where $x : [1, \ldots, n] \to [1, \ldots, m]$ and $y : [1, \ldots, m] \to [1, \ldots, n]$. Figure 3.9c depicts generating seams from pixel energies.

From this set, the *optimal seam* is found. That is, the seam that minimises the *seam cost* in Equation 3.9. Some implementations will calculate this seam using a variant of Dijkstra's algorithm, but a more common method is to use dynamic programming to implement Equation 3.10 (for vertical seams - the definition of M for horizontal seams is similar).

$$E(\mathbf{s}) = E(\mathbf{I_s}) = \sum_{i=1}^{n} = e(\mathbf{I}(s_i)) \tag{3.9}$$

$$M(i,j) = e(i,j) + \min\left(M(i-1, j-1), M(i-1, j), M(i-1, j+1)\right) \tag{3.10}$$

Building on the original algorithm, this project implements the *forward energy* function developed by Rubinstein et al. [52]. Another dynamic programming algorithm, forward energy, improves the selection of the optimal seam in iteration i by accounting for the impact on energy in the iteration $i+1$ of the algorithm and iteration i. To do this, the *energy difference* function is defined by Equation 3.11, where C is the cost of removing the seam.

$$\Delta E_{i+1} = E(\mathbf{I}_{i+1}) - (E(\mathbf{I}_i) - E(C_i)) \tag{3.11}$$

Consequently, this criterion looks forward to the resulting image to search for the seam, which would inject minimal energy. The cost of removing a seam is measured as the forward differences between the pixels that become neighbours once the seam is removed. There are three cases for this, the pixels are diagonally adjacent in either direction, or the pixels are orthogonally adjacent, as demonstrated by Figure 3.8 and formalised by Equation 3.12. From this, the Equation 3.10 can be updated to become Equation 3.15.

$$C_L(i,j) = |\mathbf{I}(i, j+1) - \mathbf{I}(i, j-1)| + |\mathbf{I}(i-1, j) - \mathbf{I}(i, j-1)| \tag{3.12}$$

$$C_U(i,j) = |\mathbf{I}(i, j+1) - \mathbf{I}(i, j-1)| \tag{3.13}$$

$$C_R(i,j) = |\mathbf{I}(i, j+1) - \mathbf{I}(i, j-1)| + |\mathbf{I}(i-1, j) - \mathbf{I}(i, j+1)| \tag{3.14}$$

$$M(i,j) = e(i,j) + \min \begin{cases} M(i-1, j-1) + C_L(i,j) \\ M(i-1, j) + C_U(i,j) \\ M(i-1, j+1) + C_R(i,j) \end{cases} \tag{3.15}$$

It is important to note that there have been several extensions to seam carving that may apply to this project. In particular, optimisations for videos by introducing two-dimensional seams to allow time to be accounted for, and implementations using GPUs to reduce execution time [52, 16].

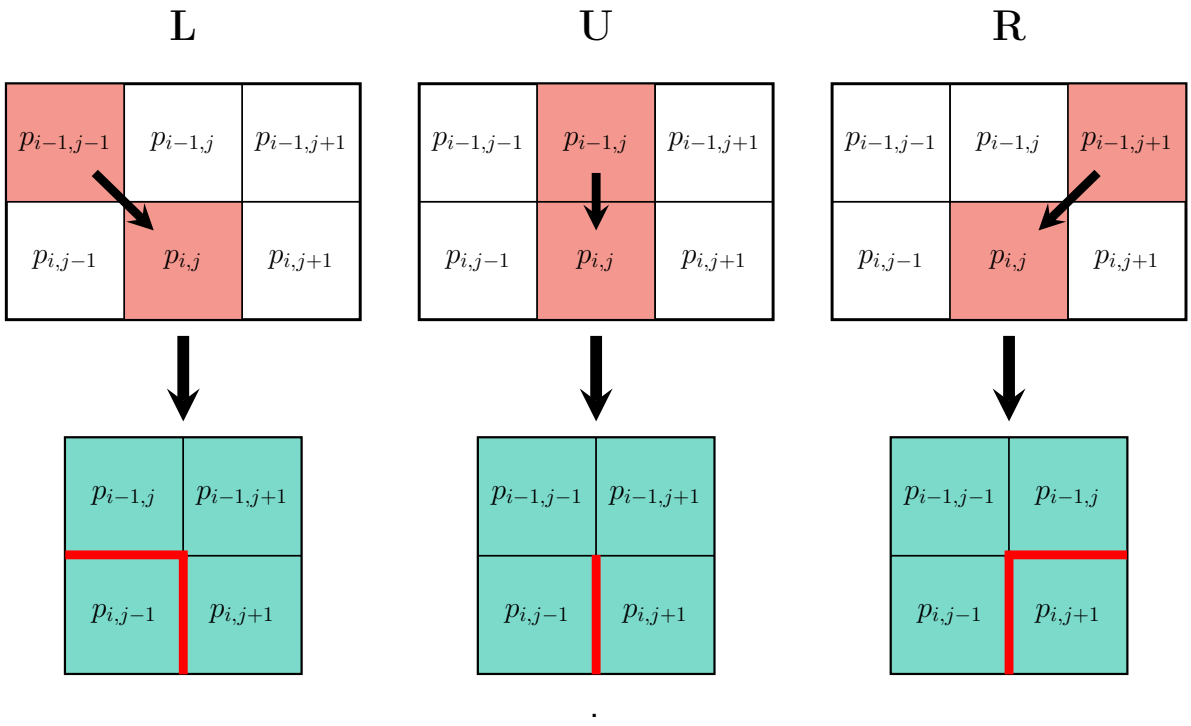


Figure 3.8: There are three possible vertical seam costs for pixel (i, j). After removing the seam, new neighbours (blue) and new edges (red) are created.

3.3.2 Graph Representation

There are several advantages to representing an image using a graph. Firstly, graphs are discrete, mathematically simple objects that are well-suited to developing efficient, provably correct algorithms. Also, graph theory is a well-established research area with a wide variety of existing algorithms and theorems that can be utilised. More pertinent to this dissertation, graphs provide minimalistic representations of images that are flexible enough to account for different image types.

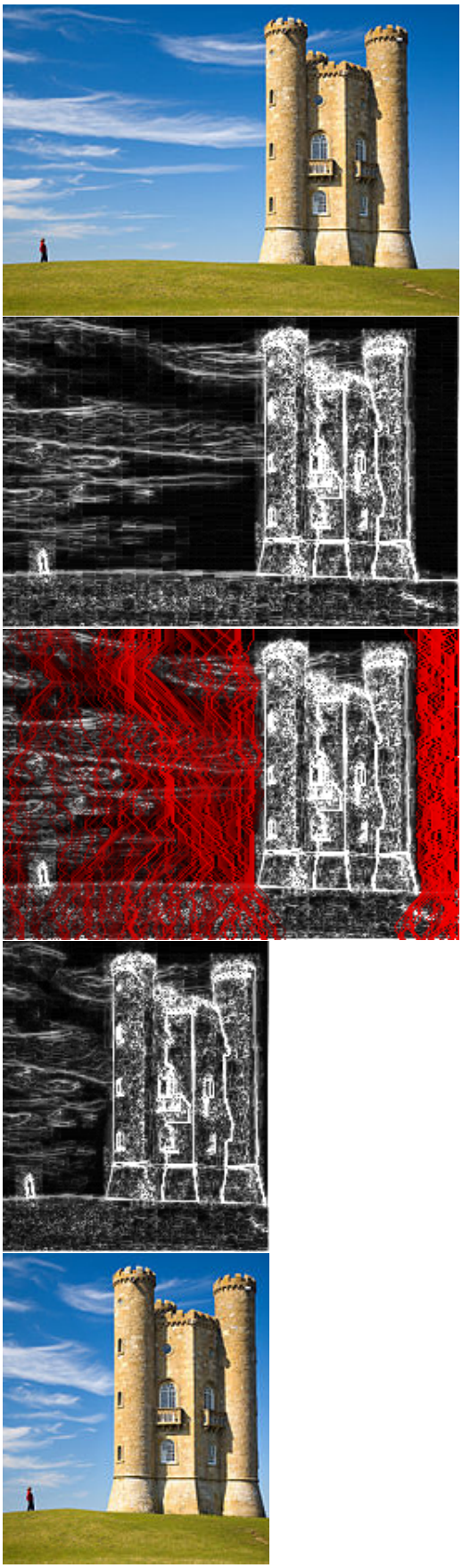
Graph-based image processing methods operate on *pixel adjacency graphs*. More specifically, graphs whose vertex set is the set of image elements and edge set is given by an adjacency relation between image elements. A common approach to this uses the *Euclidean adjacency relation* to define the edge set, formalised by Equation 3.12.

$$d(v, w) \leq \rho \tag{3.16}$$

for all vertices v, w in the vertex set. An example of some pixel adjacency graphs is given by Figure 3.10. When handling video files, three-dimensional pixel adjacency graphs are used to account for relationships between video frames. An example of these is depicted by Figure 3.11.

The pixel adjacency graphs can naïvely be applied to any image by creating a node for each pixel and an edge for every adjacency. While this does provide some opportunity for optimisations in regards to inference algorithms, it is unlikely to reduce the *video size*, so it will have little positive impact on *transmission time*.

However, all hope is not lost. Improvements can be made to this representation to reduce the overall size of each frame in the video. Primarily, the pixel adjacency graph can be extended to become *region adjacency graphs*. In this case, rather than representing each individual pixel with a node, similar regions of an image are amalgamated



(a) Original image.

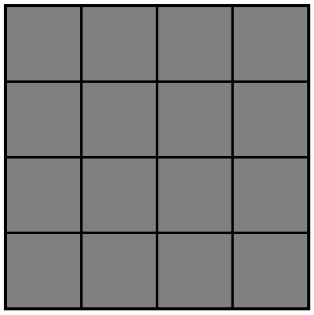
(b) The energy of each pixel.

(c) Potential seams to delete.

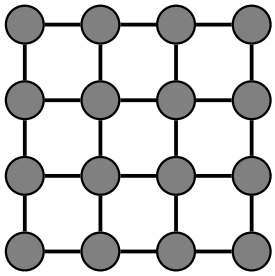
(d) Low energy seams removed.

(e) Resulting image.

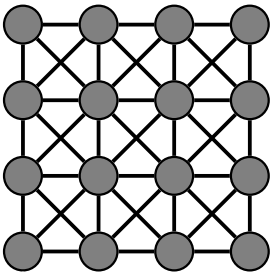
Figure 3.9: Each stage of the seam carving algorithm. Images taken from [45].



(a) A 2D image with 4×4 pixels.

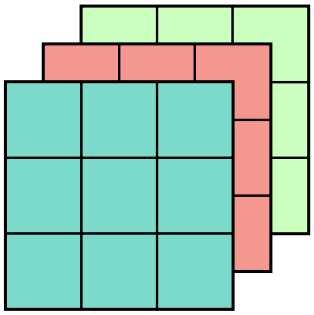


(b) A 4-connected pixel adjacency graph.

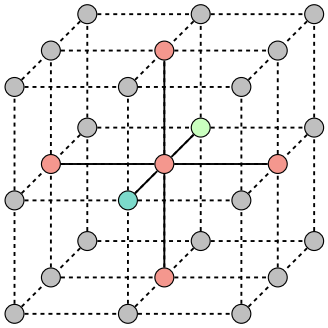


(c) An 8-connected pixel adjacency graph.

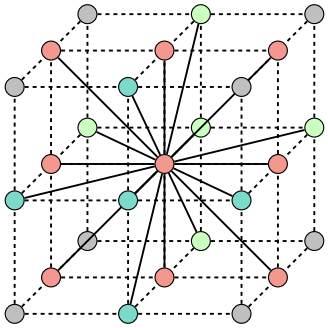
Figure 3.10: Pixel adjacency graphs.



(a) A video with three frames of 3×3 pixels.



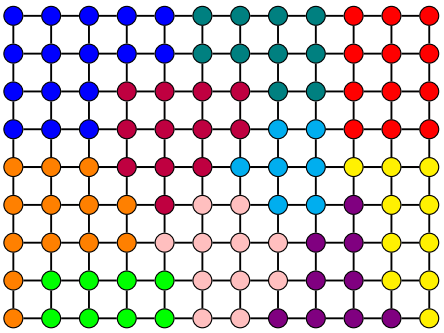
(b) A 6-connected 3D pixel adjacency graph.



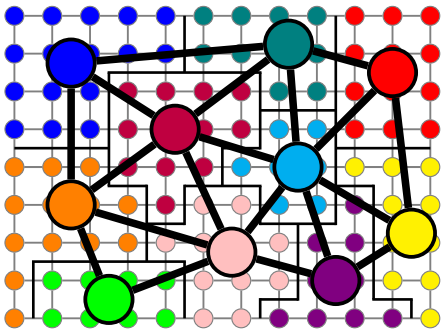
(c) An 18-connected 3D pixel adjacency graph.

Figure 3.11: 3D pixel adjacency graphs.

into a single node, reducing the number of elements that need to be transmitted. Figure 3.12 provides a pictorial example of this.



(a) A pixel adjacency graph.



(b) The corresponding region adjacency graph.

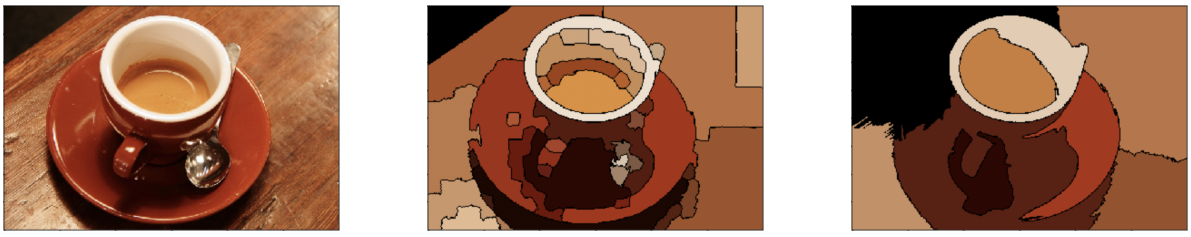
Figure 3.12: Converting a pixel adjacency graph to a region adjacency graph.

To achieve this, the notion of similarity of pixels must be quantified. An unsophisticated method might convert the pixel adjacency graph into a weighted graph, creating a formula converting the difference in the intensity of the pixels to a weight. An example of this is normalising the difference between values - if the pixels have precisely the same intensity, give their edge a weight of 1; if they are exact opposites (i.e., one black and one white), give their edge a weight of 0. Once this has been completed, nodes with sufficiently high weight can be combined into a single node.

Immediate issues begin to arise when considering this method. Firstly, what function should be used to assign weights? Secondly, how should nodes be combined if the pixels have different values? Should the mean value be taken? The median? One chosen at random? Moreover, how many nodes should be combined? What will happen if too many or too few are collected together?

The concerns surrounding weighting edges indicate that this problem may be more complicated than it first appears. In fact, grouping the nodes is a form of low-level image segmentation. While this makes the problem more computationally complex, it has the advantage that there exist well-established algorithms providing reasonable solutions to it. Unsupervised clustering algorithms such as *the watershed transform* [5] or *k-means clustering* [41] are two such methods that have been applied to this problem previously.

More importantly to this investigation, the number of regions in the image will directly impact the transmission time. Reducing the number of nodes in the graph is advantageous because it reduces the amount of data transmitted. However, in doing so, it also reduces the image's resolution. Consequently, removing too many nodes from the graph will remove any clarity of the contents of the image, making the process worthless. Figure 3.13 depicts this. Therefore, this balance must be struck heuristically to find the optimal point between maximising performance benefits and minimising loss of image data. Moreover, this optimal point is likely to be different for every image, adding a further layer of complexity.



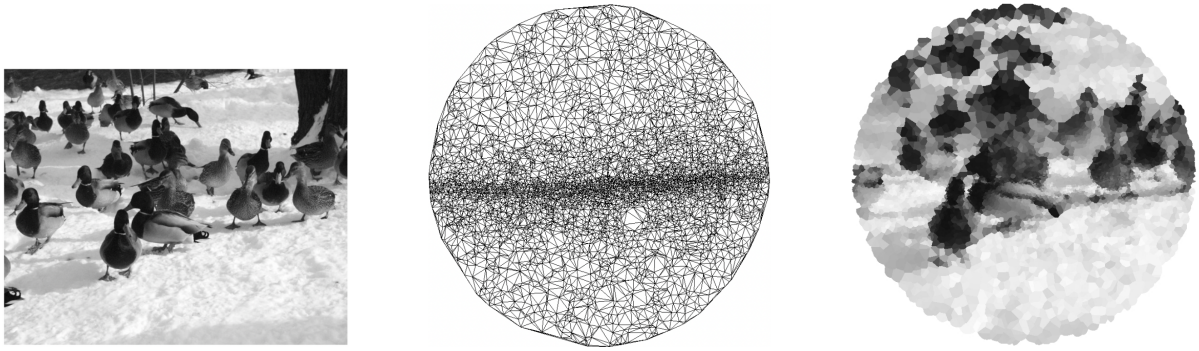
(a) A photograph of a cup of coffee.

(b) The picture of coffee split into approximately 400 regions.

(c) The picture of coffee split into approximately 10 regions.

Figure 3.13: A demonstration of the impact of reducing the number of regions on image quality. Images taken from [22].

Using similar techniques to seam carving, it is possible to make this trade-off less severe. For example, *Foveal sampling* is a method of recreating the visual activity of the eye in the mapping of an image [22]. The *Fovea centralis* is a region of the retina responsible for the sharp central vision used by mammals to focus on particular objects. Foveal sampling uses the shape of the Fovea centralis to produce a graph that can be overlayed onto an image, extracting the areas that an observer will focus on. Consequently, more regions are created in areas critical to perception, and fewer in areas out of focus. This allows the region budget to be more efficiently used, so the overall number required can be smaller without impacting image quality as significantly. Similar techniques have been applied using saliency maps or other methods for determining the importance of regions in an image.



(a) A photograph of some ducks.

(b) The retinal topography of a kangaroo.

(c) The photograph of ducks sampled using the retinal topography of a kangaroo.

Figure 3.14: An example of foveal sampling. Images taken from [22].

3.3.3 Parallelisation

Where the previous sections aimed to improve video transmission time by reducing the size of video files, this section targets the bottlenecks limiting the transmission rate of the system. To do this, the project investigates the application of parallel computing.

Parallel computing is often conflated with *concurrent computing*. However, the terms are distinct, and can exist both separately and together. In parallel computing, a task is broken down into numerous, very similar sub-tasks that can be completed independently and recombined later [49]. In concurrent computing, the various sub-tasks will address unrelated processes varying in nature, often requiring inter-process communication during execution [49]. This area of the project began considering parallelisation alone to attempt to maximise performance gains, but expanded into concurrency as the breadth of functionality that could benefit from these modifications became apparent. In Figure 3.15, the abstract layers of the networking processes have been coloured to indicate whether concurrent or parallel computing is used.

Traditionally, computer design has focussed on *serial computation*. To solve a computational problem, a sequence of instructions was written, they were executed in order, and a result was returned. As predicted by Moore's law, technology scaling was able to double the performance of processors so that they could compute more complex expressions more efficiently for decades [53]. However, factors like Dennard scaling mean this cannot last forever [61]. Therefore, to continue improving performance, computer architects turned to multiprocessing. Rather than making processor components more efficient so that a single instruction executes faster, similar performance gains can be made by executing multiple instructions simultaneously. Consequently, since evidence suggests processors will continue to be optimised to parallel computation, this seemed like a viable opportunity for investigating where performance might be gained in future iterations of surveillance technology.

Transmission

Parallelisation already exists in some communication protocols. The *transmission control protocol* (TCP) uses a *sliding window protocol* to send a group of data packets concurrently, ensuring they are ordered correctly at the receiving end. Figure 3.16

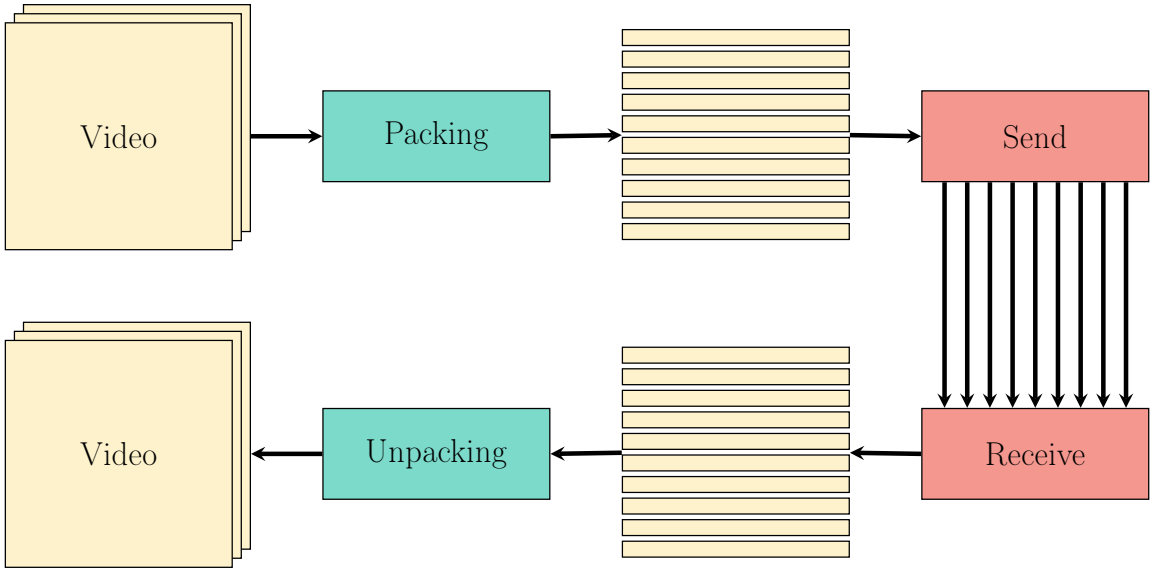
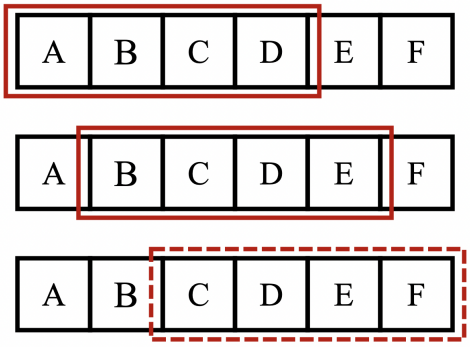


Figure 3.15: The stages required for a video to be sent across the network.

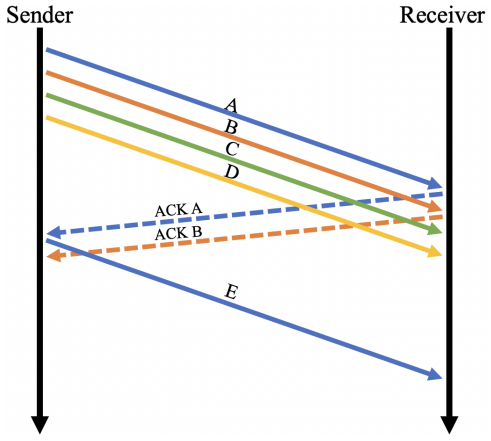
Parallel processes:

Concurrent processes:

depicts this. This and similar protocols exist in the *data-link* layer of the *OSI network model*. The goal of this section of the investigation was to attempt to move the parallelisation higher up the abstract stack.



(a) The sliding window (red) moves across the packets as each one is sent. Initially the first four packets are sent (top). When packet A is acknowledged, the window slides along one, and E is sent (middle). After the acknowledgement for B is received the window will slide along and F will eventually be sent (bottom).



(b) The packets currently in the window are sent at the same time, without waiting for any acknowledges. Once an acknowledgment has been received, the next packet in the queue can be sent.

Figure 3.16: A high-level view of TCP's sliding window protocol.

Taking inspiration from sliding windows, instead of sending all video data in a single stream, videos are split into frames, and each frame is divided into packets. Meanwhile, a pool of threads can be created to represent the size of the window. When a packet is ready to be sent, it is assigned a thread from the pool, and the thread establishes a connection with the server, transmitting the data. Consequently, multiple connections will be open in parallel, so, in a given moment, more data will

be sent.

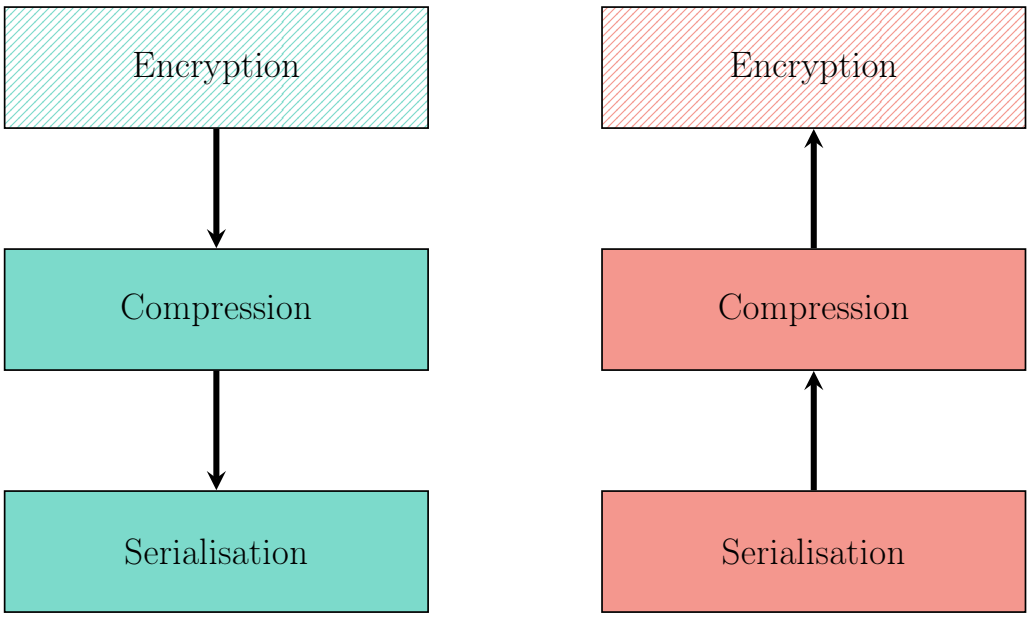
However, there are limitations to this technique. Firstly, more data will have to be transmitted than in sequential communication. The algorithm is non-deterministic, so there can be no guarantees about the order in which the packets will arrive after transmission. Consequently, further information must be provided to ensure videos are reassembled correctly. More specifically, a frame number and packet identifier will have to be attached to each packet. While this is worth noting, the size of this additional data is negligible compared to HE data, so it is not a critical issue.

A more pressing concern is the overhead of creating threads and establishing connections. The cost is such that creating too many threads will remove parallelisation benefits or even go so far as to make data transmission slower. Consequently, an optimal balance between the cost of parallelisation and the amount of data to send must be found to maximise gains from this approach.

Data Manipulation

Splitting videos into small packets has further advantages. Before data can be sent from the client to the server, and vice versa, it must be prepared. This can be referred to as *packing* the data. Similarly, when it arrives at its destination, data must be reorganised, or *unpacked*.

Succinctly depicted by Figure 3.17, there are three distinct stages of the packing in the client *encryption*, *compression*, and *serialisation*. The unpacking process will reverse these stages in order. In the server-side of the project, the encryption and decryption operations are missing from these pipelines.



(a) The packing stack. *(b) The unpacking stack.*

Figure 3.17: The packing and unpacking stacks. Block colours indicate processes occur in both client and server, patterned colours indicate the process is only occurs in the client.

In a naïve implementation, each pixel of a video would be encoded and encrypted independently. However, the CKKS scheme operates on vectors of real values. Therefore, breaking a frame down into rows provides the opportunity of *vectorising* the

application by encrypting each row of a frame as a single ciphertext object. This has the advantage of reducing the number of ciphertext objects needed – for an $n \times m$ pixel frame, the number of objects is reduced from nm to n - so reduces the memory consumption and significantly improving transmission time[4].

Similarly, compressing and serialising subsections of a video rather than the whole video does not change the outcome but has the advantage of being parallelisable. Therefore, once encryption has been complete, these operations can be performed in the same thread, without having any disadvantages of requiring more threads to be created. The functions are only limited in their need for the previous stage to entirely terminate before they can begin. Consequently, the smaller the quantum of data they operate on, the quicker they will finish. However, the same concerns that occur during data transmission also occur here. That is, the overhead of creating threads must be balanced against reducing the decomposition of data.

3.4 Inference

This section discusses the implementation of the inference models required for moving object detection. It will examine the necessary modifications to support HE video data and detail the more complex algorithms needed for unsupervised learning. Adaptations were required to incorporate the HE Boolean circuits and overcome operation depth limitations. The more straightforward adjustments are summarised in §3.4.1, and the investigations into GMMs are given a more in-depth presentation in §3.4.2 and §3.4.3.

3.4.1 Homomorphic Encryption Adaptations

There were two main challenges to overcome when converting inference algorithms to the HE domain. Firstly, the number of operations that can be applied is limited by the depth of the ciphertext. Secondly, the set of operations supported by CKKS is more limited than is available when working with plain data. Consequently, algorithms needed to be modified, and compromises made to produce accurate results without introducing detrimental side effects - for example, to accommodate more operations, the depth of a ciphertext could be increased, but this significantly reduces the efficacy of transmitting data.

The adaptations required for each of the less complex algorithms are detailed below. A discussion of the techniques investigated for implementing GMMs is left to §**??**, where more time can be given to an in-depth analysis.

Frame Differencing

Frame differencing is a relatively simple algorithm to adapt for the HE domain. It only requires a single operation, *subtraction*, that is provided by the standard CKKS implementation. Moreover, subtraction does not require a ciphertext to be rescaled. As such, its level is never decreased, so the size of the ciphertext can be minimised. As well as making networking more efficient, this also makes the inference algorithm

[4]This would fall under concurrency rather than parallelism because objects such as encoders, encryptors, and keys must be shared between processes.

faster because the data being operated over is more minor.

Therefore, the only modification required to operate over HE data is to replace the subtraction function in the traditional algorithm with a call to the subtraction circuit provided by the HE library.

Mean Filter

The traditional mean filter algorithm has two standard implementation versions. One option is to calculate the mean from scratch every time. To do this, a list of all frames that have been observed so far must be stored; then, when a new frame is received, it can be added. From this, the pixels can be summed and divided by the size of the list to provide an array of mean values in the same shape as the video frames. In contrast, the second version does not require storing all previous frames. Instead, the mean frame is updated using an iterative formula every time a new frame is received.

It is obvious that the second method will perform better in both time and space complexity than the first when considering plain video data. However, when investigating HE data, the distinction is not as straightforward. The second method becomes problematic in that the mean must have both *multiplication* and *addition* operations applied to it during the updating phase. Each time a multiplication circuit is applied, the ciphertext will be reduced. Therefore, the ciphertext must have the same number of levels as frames in the video. This quickly becomes infeasible. Increasing the size of ciphertext as far as would be required would significantly detriment the time complexity of the operations. Likewise, it would also make transmitting data between client and server much slower. Consequently, this method can be immediately ruled out.

The first method encounters different difficulties. One particular issue is the space complexity of storing all frames in the video. Since HE data can get very large, storing multiple copies of each frame significantly impacts the application's memory usage. Similarly, HE operations are noticeably slower than plaintext operations. Therefore, while the number of frames to be handled may not significantly impact the running time of plaintext implementations, as the video progresses, a HE implementation will become considerably slower. However, this method can be used to derive a solution. A compromise can be achieved by setting an upper bound on the number of preceding frames stored and forgetting the oldest frame whenever a new one arrives. Significantly, this may reduce the accuracy of moving object detection, so a balance must be struck between running time and inference quality.

Gaussian Average

Similarly to one of the methods proposed for implementing mean filter inference, the mean and variance values required to fit the Gaussian distributions are calculated iteratively using Equation 3.17 and Equation 3.18 (also given by Equation 2.7 and Equation 2.8).

$$\mu_t = \begin{cases} f_0 & \text{if } t = 0 \\ \alpha f_t + (1 - \alpha)\mu_{t-1} & \text{otherwise} \end{cases} \tag{3.17}$$

$$\sigma_t^2 = \begin{cases} c & \text{if } t = 0 \\ d^2\alpha + (1-\alpha)\sigma_{t-1}^2 & \text{otherwise} \end{cases} \tag{3.18}$$

where α determines the size of the temporal window, $d = |f_t - \mu_t|$ represents the Euclidean distance between a pixel and the mean, and c is some constant.

Consequently, using these definitions, a Gaussian average inference implementation will be equally infeasible as the mean filter implementation. Fortunately, the implementation can be adapted to reduce the number of applications of multiplication required.

The first step of the adaptation requires the observation that expanding the iterative definitions of the mean and variance highlights that the frames of the video are *multiplicatively independent* of each other - in other words, the frames are multiplied by a constant value, and then they are added together. This is demonstrated in Equation 3.19 for the fifth frame of the video.

$$\begin{aligned} \mu_4 &= \alpha f_4 + \alpha(1-\alpha)f_3 + \alpha(1-\alpha)^2 f_2 + \alpha(1-\alpha)^3 f_1 + (1-\alpha)^4 f_0 \\ \sigma_4^2 &= \alpha d_4^2 + \alpha(1-\alpha)d_3^2 + \alpha(1-\alpha)^2 d_2^2 + \alpha(1-\alpha)^3 d_1^2 + (1-\alpha)^4 c \end{aligned} \tag{3.19}$$

Interestingly, the coefficient terms for each frame are identical, so the computation can be shared across both calculations. More importantly, the value of α is predetermined, decided by the server before runtime to weight frames according to recency. Consequently, the coefficients can be pre-calculated in the clear before being encoded for HE multiplication. This means that only a single multiplication needs to be applied to each frame before the results can be summed to give the means and variances. Therefore, the number of levels required for a ciphertext is minimised, so the ciphertext size is minimised.

However, it must be noted that this is assuming each frame is known. Therefore, a list of preceding frames must be stored so that each formulae can be recalculated entirely each time a new frame is received. Like with a mean filter, this introduces the trade-off of running time against inference accuracy, as increasing the number of frames stored will make the results of inference more accurate but will take longer to calculate as more multiplications will have to be performed each time.

3.4.2 Online Mixture Model

In 1999 Stauffer and Grimson proposed *adaptive background mixture models* for real-time moving object detection [56]. To overcome a single Gaussian distribution's inability to cope with the changing lighting conditions in practice, they proposed a mixture of adaptive Gaussians. For each frame of the video, the parameters of the Gaussians are updated, and they are heuristically evaluated to hypothesise which are most likely part of the *background process*. The advantage of this technique over other GMMs is that the model runs *online*. Consequently, no training phase is required. Instead, the model can be fitted, and results returned in a single phase. While this is useful for real-time inference acting on a constant stream of data, it has the disadvantage of producing less accurate results earlier in the execution sequence.

Fitting

For a particular pixel, the values that occur over time are known as the *pixel process*. This is a time series of pixel values such that, at any time t, the process of pixel (x, y) is defined by Equation 3.20.

$$\{X_0, \ldots, X_t\} = \{I(x, y, i) \mid 0 \leq i \leq t\} \tag{3.20}$$

where I is the image sequence.

Many guiding factors influence the definition of the model and updating procedure. For example, lighting changes must be tracked, static objects added to the scene must be incorporated into the background, and no camera sensor is perfect, so random noise in pixel values must be ignored. From these factors, it can be deduced that more recent observations will be more useful when determining Gaussian parameter estimates.

The recent history of a pixel can be modelled as a mixture of K Gaussian distributions. K is usually a value between 3 and 5, depending on available memory and computational power availability. Given a pixel process, the probability of observing the pixel value at time t is given by

$$\mathbb{P}(X_t) = \sum_{i=1}^{K} \omega_{i,t} \times \eta(X_t, \mu_{i,t}, \Sigma_{i,t}) \tag{3.21}$$

where $\omega_{i,t}$ represents an estimate of the proportion of the data accounted for by the i^{th} Gaussian at time t, $\mu_{i,t}$ and $\Sigma_{i,t}$ are the mean and covariance matrix of the i^{th} Gaussian at time t respectively. η is the Gaussian probability density function given by Equation 3.22.

$$\eta(X, \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(X-\mu)^T \Sigma^{-1} (X-\mu)} \tag{3.22}$$

Rather than using the *expectation maximisation algorithm* (see §3.4.3 for more information) to maximise the likelihood of the observed data, Stauffer and Grimson suggested using a *K-means* approximation to engender the online aspect of the system. Each pixel in a new frame is compared against the existing K Gaussian distributions until a *match* is found. A match occurs when a pixel value is within a predefined number of standard deviations of a distribution. The number of standard deviations will vary across distributions as each distribution will account for different factors such as lighter or shadier regions.

In the event that none of the distributions match a pixel's value, the least likely Gaussian is replaced by a new distribution defined with the pixel value as its mean, an initially high variance, and low prior weight. Then, the prior weights are adjusted at time t using Equation 3.23.

$$\omega_{k,t} = (1 - \alpha)\omega_{k,t-1} + \alpha M_{k,t} \tag{3.23}$$

where α is a learning rate, and M is an indicator function of 1 if Gaussian k at time t matched, and 0 otherwise. After this approximation is complete, the weights are normalised.

For unmatched distributions, the μ and σ parameters are unchanged. However,

the parameters of the matching distributions are updated according to Equation 3.24, where rho is defined by Equation 3.25.

$$\mu_t = (1 - \rho)\mu_{t-1} + \rho X_t \tag{3.24a}$$

$$\Sigma_t^2 = (1 - \rho)\Sigma_{t-1}^2 + \rho(X_t - \mu_t)^T(X_t - \mu_t) \tag{3.24b}$$

$$\rho = \alpha\eta(X_t, \mu_k, \Sigma_k) \tag{3.25}$$

Predicting

Once the parameters have been updated, the Gaussian most likely produced by the background process must be determined to segment the foreground and background. This decision is based on the assumption that there will be a relatively little variance in the Gaussian distributions when a static, persistent object is in the frame. In contrast, when a new object occludes the background, it will generate significant variance and not match an existing distribution. Consequently, a method of defining the proportion of the GMM representing the background process is required.

To achieve this, first, the Gaussians are ordered based on the value of $\frac{\omega}{\Sigma}$. The definitions of ω and Σ mean that this value will increase as both the distribution gains more evidence and the variance decreases. This value will only differ from the last iteration for matching distributions, so the sorting process can be made more efficient. The ordered list can then be iterated over, and the first B distributions are taken as the *background model*, where

$$B = argmin_b \left(\sum_{k=1}^{b} \omega_k > T \right) \tag{3.26}$$

The threshold, T, is a measure of how much data should be accounted for. In other words, the best-fitted distributions are taken until a certain portion of recent data has been considered.

Once the background model has been decided, it can be used to label the pixel as either *foreground* or *background*, allowing the moving objects to be extracted as the foreground.

3.4.3 Expectation-Maximisation Algorithm

Proposed by Dempster et al. in 1977, the *expectation-maximisation* (EM) algorithm is a general iterative method for maximising the likelihood of *latent variables* using a statistical model [14]. There are two stages in the algorithm: the expectation stage, or *E-step*, and the maximisation stage, or *M-step*, which are iterated over until the model converges. The E-step generates a function for the expectation of the likelihood of the data points occurring given the current model parameters. The M-step computes new parameters to maximise the function found in the E-step. While this will always increase the *marginal likelihood function*, there is no guarantee that the EM algorithm will converge to a maximum likelihood estimator. For example, the algorithm may converge on a local maximum. To overcome this, techniques such as *random-restart hill climbing* can be employed [30].

Although the EM algorithm can be applied to any statistical model, this dissertation will discuss its application to GMMs. The algorithm can be used to assign observed data points to components of the model such that the likelihood of the components generating the points is maximised. When applied to a GMM, the E-step can be formalised by the below process. To begin with, the *pseudo-posterior* - the probability that an observation, X_i belongs to a component Z_k - is calculated using Equation 3.27.

$$\gamma_{Z_i=k} = \mathbb{P}(Z_i = k \mid X_i) = \frac{\mathbb{P}(X_i \mid Z_i = k)\mathbb{P}(Z_i = k)}{\mathbb{P}(X_i)} \tag{3.27}$$

$$= \frac{\omega_k \mathcal{N}(x_i, \mu_i, \sigma_i)}{\sum_c \omega_c \mathcal{N}(x_c, \mu_c, \sigma_c)} \tag{3.28}$$

where ω_k is the component weights of component k and $\mathcal{N}(x_i, \mu_i, \sigma_i)$ gives the probability of x_i under component k.

The *auxillary function* defined by Equation 3.29 can then be applied to the result, $\gamma_{Z_i=k}$, where $\theta^{(t-1)}$ is the parameter generated in the previous iteration and $\theta^{(t)}$ is the new parameter value. Using Jensen's inequality, it can be proven that this auxiliary function is the lower bound of the gain of the likelihood that is obtained by updating the parameter values, but this proof is excluded for brevity.

$$Q(\theta^{(t)}, \theta^{(t-1)}) = \mathbb{E}\left[\log \mathbb{P}(Z \mid \theta^{(t)}) \mid X, \theta^{(t-1)}\right] \tag{3.29}$$

$$= \sum_{k=1}^{M} \log \mathbb{L}(\theta_k \mid X, Z)\mathbb{P}(Z_k|X, \theta^{(t-1)}) \tag{3.30}$$

$$= \sum_{k=1}^{M} \log \mathbb{L}(\theta_k \mid X, Z)\ \gamma_{Z_i=k} \tag{3.31}$$

where $\log \mathbb{L}(\theta_k \mid X, Z)$ is the log likelihood of a Gaussian component with updated parameters and $\mathbb{P}(Z_k|X, \theta^{(t-1)})$ is the distribution of latent variables according to the current parameters.

After the auxiliary function has been generated, the M-step can begin. This means maximising the value of Q to produce the optimal parameter value in Equation 3.32.

$$\theta^{(t+1)} = argmax_\theta\ Q(\theta^{(t)}, \theta^{(t-1)}) \tag{3.32}$$

where

$$Q(\theta^{(t)}, \theta^{(t+1)}) = \sum_{k=1}^{M}\sum_{i=1}^{N} \log \gamma_k \mathbb{P}(Z_k \mid X_i, \theta^{(t-1)}) + \sum_{k=1}^{M}\sum_{i=1}^{N} \log \mathbb{P}(x_i \mid \theta_k)\mathbb{P}(Z_k \mid X_i, \theta^{(t-1)}) \tag{3.33}$$

From this, the optimal parameter values can be derived by differentiating Equation 3.33 with respect to the means, covariances, and weights, and solving when equal to zero, in turn. The results of these calculations are given Equation 3.34, Equation 3.35, and Equation 3.36, respectively. In the equations, $N_k = \sum_{i=1}^{N} \gamma_{Z_i=k}$.

$$\hat{\mu}_k = \frac{\sum_{i=1}^{N} X_i \mathbb{P}(Z_i = k \mid X_i, \theta^{(t-1)})}{\sum_{i=1}^{N} \mathbb{P}(Z_i = k \mid X_i, \theta^{(t-1)})} = \frac{1}{N_k}\sum_{i=1}^{N} \gamma_{Z_i=k}\ X_i \tag{3.34}$$

$$\hat{\sigma^2}_k = \frac{1}{N_k} \sum_{i=1}^{N} \gamma_{Z_i=k} (X_i - \mu_k)^2 \tag{3.35}$$

$$\hat{\omega}_k = \frac{N_k}{N} \tag{3.36}$$

Chapter 4

Evaluation

This chapter evaluates the project's implementation using three main criteria: the extent to which it meets the success criteria detailed in §2.2.1, the applicability of HE to inference algorithms, and its practicality regarding current, real-world surveillance technology. Consequently, the chapter is divided into three sections to tackle each criterion distinctly. Both quantitative and qualitative analysis is used throughout the chapter to analyse the implementation's performance and make predictions about the scope to which the investigation may be extended in future. Unless otherwise specified, the data presented was generated using 32×32 pixel images from the Moving-MNIST dataset.

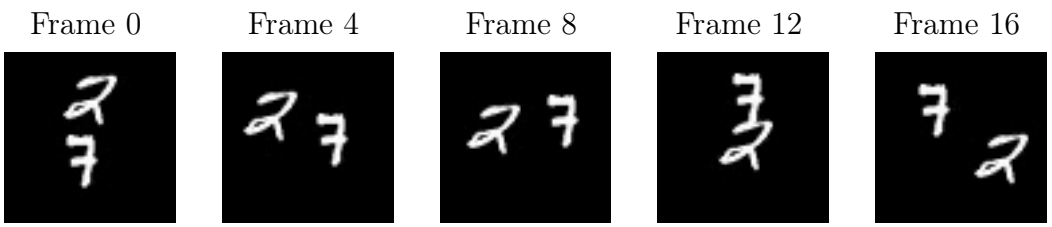
4.1 Requirements Analysis

The success criteria in §2.2.1 were split into two categories: *core* and *extensions*. As detailed in Table 4.1, All three of the core criteria were implemented, and one of the three extensions has also been completed. The open-ended nature of this project means that defining a *completed* state for some criterium was not trivial. For example, for criterium `A2`, while some algorithms have been implemented in their entirety, others require further investigation. However, it was important to have a goal for each criterium to properly plan the project and consider all aspects equally. Therefore, a justification for the state of each criterium has been included.

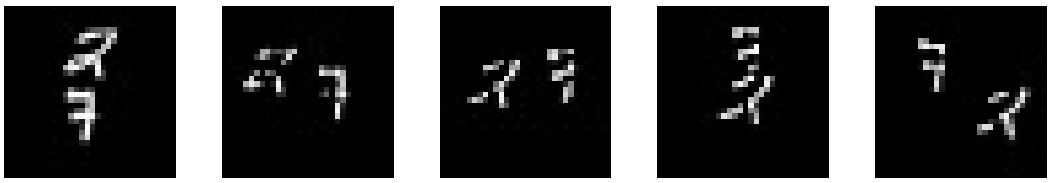
4.2 Homomorphic Encryption Integration

Adapting moving object detection algorithms for the HE domain proved to be the project's biggest challenge. The limited number of operations available, combined with the limited number of applications supported by ciphertexts, means some aspects of inference cannot be recreated. Particular challenges came when trying to implement the median filter and a GMM.

However, frame differencing, the mean filter, and the Gaussian average methods of background subtraction were successfully implemented using the techniques described in §3.4.1. A sample of the results from each of these algorithms running on the Moving-MNIST dataset is provided in Figure 4.1, and an example of what can be achieved on a more realistic dataset - the LASIESTA dataset - is provided in Figure **??**.



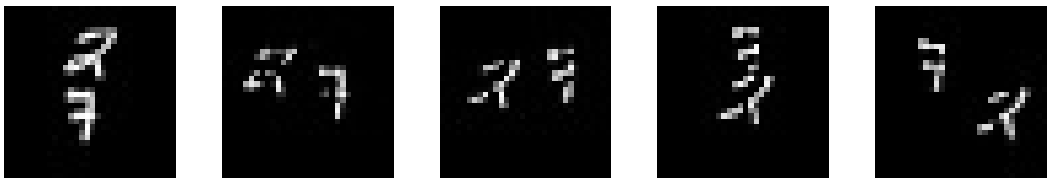
(a) Original Moving-MNIST frames.



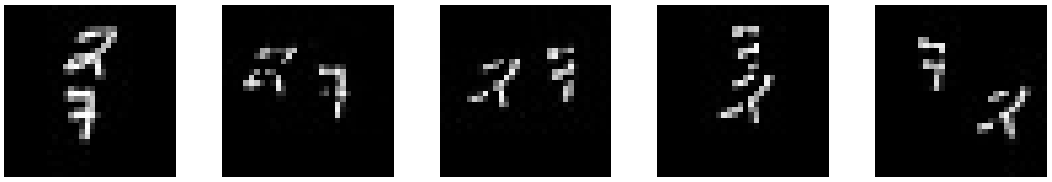
(b) Frame Differencing with CKKS.

(c) Mean Filter with CKKS.

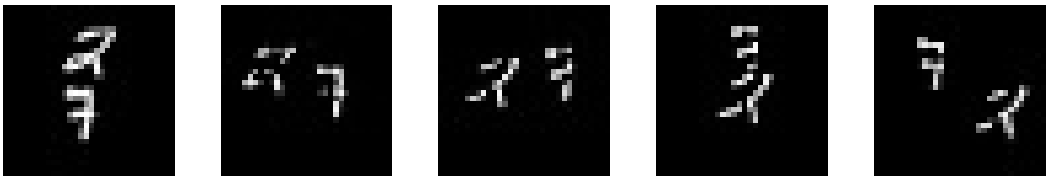
(d) Gaussian Average with CKKS.



(e) Frame Differencing with MeKKS.



(f) Mean Filter with MeKKS.



(g) Gaussian Average with MeKKS.

Figure 4.1: Moving-MNIST Inference Results

Requirement	Achieved?	Justification
A1	✓	The project contains a client-server application that allows videos to be homomorphically encrypted and transmitted across a network, with implementation techniques detailed in §3.3.
A2	✓	The project contains several algorithms that are able to extract moving objects from homomorphically encrypted videos, with implementation techniques detailed in §3.4.
A3	✓	The accuracy of HE inference algorithms are evaluated to investigate their efficacy and applicability in §4.2.
B1	✓	The MeKKS scheme, detailed in §3.2.2, provides a complete implementation of the fundamental principles of the CKKS HE scheme.
B2	✗	Due to time constraints, an independent investigation into the security of HE schemes could not be completed. However, only well-established, trusted schemes were considered; hence CKKS was selected and reimplemented over less secure schemes.
B3	✗	The implementation of moving object detection algorithms proved to be more open than expected. Consequently, more time was dedicated to further understanding this area rather than expanding into other inference paradigms.

Table 4.1: Requirements analysis.

4.2.1 Online Mixture Model

In the *online mixture model* algorithm detailed in §3.4.2, Equation 3.26 describes how a fitted model can be used to segment an image. However, inequality comparison operators are not provided by the standard CKKS implementation. To solve this, Cheon et al. proposed the algorithm in Figure 4.2 [12]. Unfortunately, this introduces security concerns. If the ability to compare two HE ciphertexts is added to the system, an attacker[1] could use it to exfiltrate information about the image. For example, with enough comparison operations, they would be able to determine the exact value of each pixel in a frame. Consequently, this algorithm was abandoned to preserve the security of the system.

For the same reason, the median filter could not be implemented. The pixel values could not be ordered without a comparison operator, so a median could not be calculated.

[1] such as Mallory in §2.1.1.

Algorithm $\texttt{Comp}(a, b; d, d', t, m)$

Input: distinct numbers $a, b \in \left[\frac{1}{2}, \frac{3}{2}\right)$, $d, d', t, m \in \mathbb{N}$
Output: an approximate value of $\text{comp}(a, b)$

1: $a_0 \leftarrow \frac{a}{2} \cdot \texttt{Inv}\left(\frac{a+b}{2}; d'\right)$
2: $b_0 \leftarrow 1 - a_0$
3: **for** $n \leftarrow 0$ **to** $t - 1$ **do**
4: $\quad inv \leftarrow \texttt{Inv}(a_n^m + b_n^m; d)$
5: $\quad a_{n+1} \leftarrow a_n^m \cdot inv$
6: $\quad b_{n+1} \leftarrow 1 - a_{n+1}$
7: **end for**
8: **return** a_t

Figure 4.2: Homomorphic Comparison Algorithm

4.2.2 Expectation-Maximisation Algorithm

The difficulty in implementing this algorithm came from the number of operations that need to be performed across the fitting and predicting stages. Calculating means and covariances repeatedly requires many successive multiplications, requiring many coefficient levels in ciphertexts. Also, like the online mixture model, not all operations are supported by the CKKS scheme. In particular, the algorithm requires several divisions to be performed when updating the Gaussian distributions. Currently, the best solution for this seems to be provided by Cheon et al. [12]. However, the algorithm, given in Figure 4.3, has a minimal domain requiring input values to be between zero and two. Normalising pixel values might provide a method for incorporating this, but the noise induced by HE means inference becomes infeasibly inaccurate

Algorithm $\texttt{Inv}(x; d)$

Input: $0 < x < 2$, $d \in \mathbb{N}$
Output: an approximate value of $1/x$

1: $a_0 \leftarrow 2 - x$
2: $b_0 \leftarrow 1 - x$
3: **for** $n \leftarrow 0$ **to** $d - 1$ **do**
4: $\quad b_{n+1} \leftarrow b_n^2$
5: $\quad a_{n+1} \leftarrow a_n \cdot (1 + b_{n+1})$
6: **end for**
7: **return** a_d

Figure 4.3: Homomorphic Division Algorithm

4.3 Practicality

This section will evaluate the implementation from the perspective of practicality in real-world surveillance systems. It will do so through two key aspects: the MLaaS client-server model and the accuracy of inference algorithms.

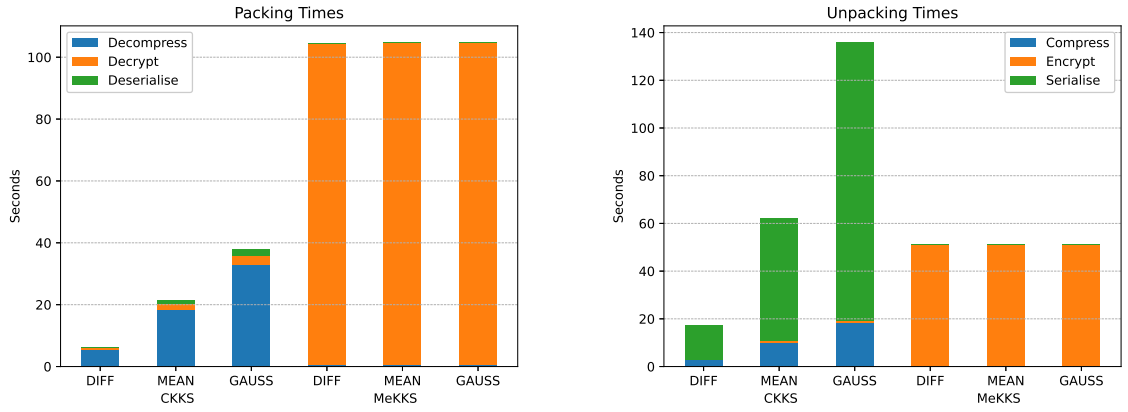
4.3.1 Networking

Data Handling

Before data can be sent across the network, it must be *packed*, and once it is received, it must be *unpacked*. This proved to be a significant bottleneck before transmitting data. During the packing phase, data must be encrypted and serialised before it is transmitted over the network. To make transmission more efficient, a compression stage is added to try and reduce the memory usage of videos. Similarly, in the unpacking phase, data must be decompressed, deserialised, and decrypted to recover the video and inference results.

As described in §3.3, several methods were investigated to try and reduce this bottleneck. By combining some of these techniques, substantial progress was made in reducing the time the packing and unpacking algorithms took to run. Figure 4.4 provides the running time of a nawïve implementation of these algorithms, and Figure 4.5 demonstrates the performance of an optimised implementation. From these charts, Table 4.2 has been derived to highlight the improvement for each category of inference and encryption scheme. Interestingly, the unpacking algorithm can be improved using parallelisation when the CKKS scheme is used, but it will worsen performance when MeKKS is used. This is due to the delays caused by deserialising data that were overcome by implementing directly in Python - although this does make the encryption and decryption functions perform dramatically worse.

While these times may appear slow, it is important to remember that surveillance companies rarely stream all video from a device. Cameras will usually contain multiple sensors to determine which video is worth performing inference on to conserve battery life. Consequently, real-time performance is not required.

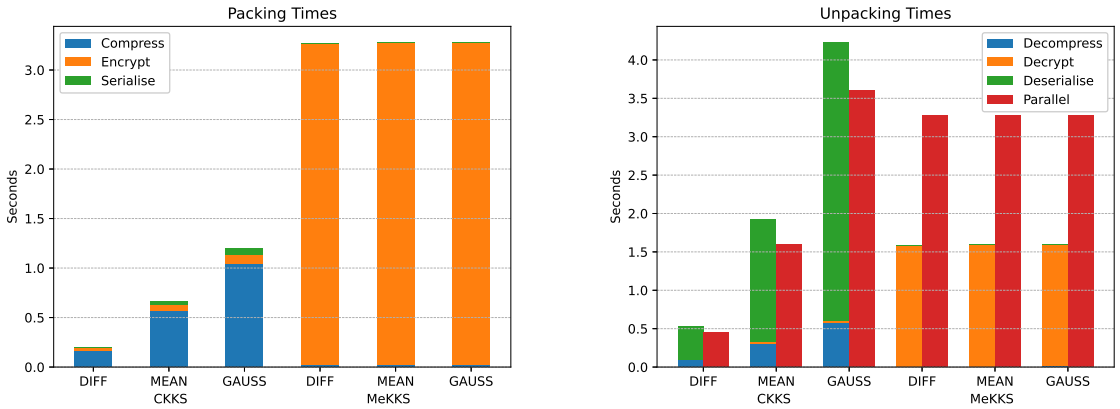


(a) Packing *(b) Unpacking*

Figure 4.4: Naïve Packing and Unpacking Times

Transmission Times

The other key aspect of the network component of the project is transmitting the data. One fundamental flaw of HE is the memory consumption inflation caused by encrypting data. Consequently, transmission times are much slower than when working with plain video data. In the final application, two main techniques were



(a) Packing *(b) Unpacking*

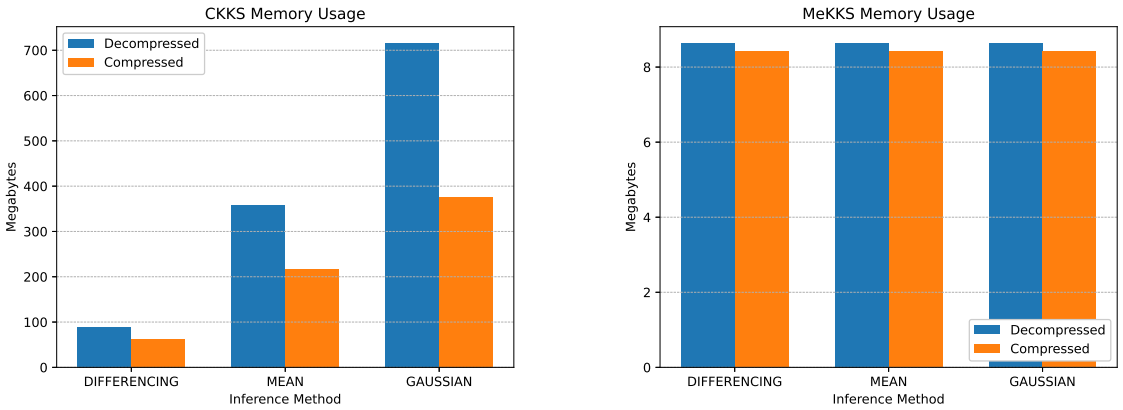
Figure 4.5: Optimised Packing and Unpacking Times

used to reduce this impact: vectorisation and compression.

For compression, several algorithms were tested - they were compared for both running time and compression ratio - and the algorithm that performed best on CKKS data was selected. The results of these tests are included in Table 4.3. From this, the impact of compressing videos is shown in Figure 4.6.

Already, this provides good improvements over raw data. However, this can be extended by encrypting rows of video frames as a single ciphertext rather than each pixel distinctly. Figure 4.7 depicts the results of this adaptation.

As a result of the above optimisations, the running times for the client and server are summarised by Figure 4.8a and Figure 4.8b respectively.



(a) CKKS *(b) MeKKS*

Figure 4.6: Naïve Memory Usage Compression Comparison

4.3.2 Inference

Another area of investigation that must be considered when discussing practicality is the performance of inference algorithms. This can be approached from two metrics. Firstly, the *running time* must be considered to evaluate if algorithms will be able to

Encryption	Inference	Process	Naïve (s)	Optimised (s)	Improvement
CKKS	Differencing	Packing	6.440	0.198	32.5×
		Unpacking	17.349	0.850	20.4×
	Mean	Packing	21.376	0.668	32.0×
		Unpacking	62.058	3.095	20.1×
	Gaussian	Packing	38.003	1.200	31.7×
		Unpacking	136.008	7.108	19.1×
MeKKS	Differencing	Packing	104.529	3.273	31.9×
		Unpacking	51.076	1.589	32.1×
	Mean	Packing	104.604	3.274	31.9×
		Unpacking	51.202	1.590	32.2×
	Gaussian	Packing	104.884	3.277	32.0×
		Unpacking	51.319	1.596	32.2×

Table 4.2: Packing and unpacking times for each encryption scheme and inference algorithm, and the improvement gained.

Algorithm	Compression (s)	Decompression (s)	Size (KB)	Percentage
None	-	-	716.18	100%
`gzip`	94.69	3.55	442.46	61.78%
`bz2`	36.28	5.02	435.29	60.78%
`lzma`	294.85	20.98	421.86	58.9%
`brotli`	871.97	0	435.78	60.85%

Table 4.3: Evaluation of compression algorithms.

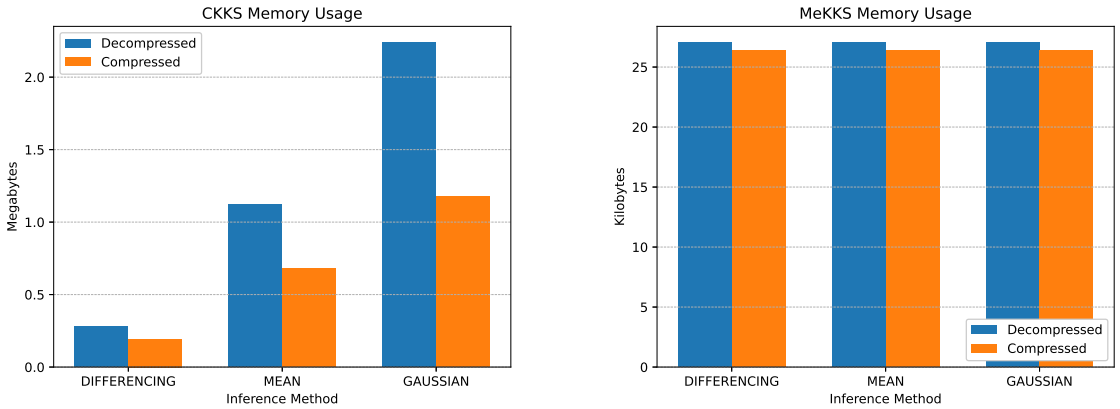
return results in a reasonable amount of time. Secondly, *accuracy* must be analysed to assess the quality of inference results compared to plain inference.

Running Time

The running time for each inference algorithm varies significantly and is severely impacted by the parameters used to tune the accuracy of each algorithm, as discussed in §3.4.1. Therefore, for this comparison, the parameters were tuned using the CKKS scheme and kept constant for a fair evaluation when testing the MeKKS scheme. The results are depicted by Figure 4.9. As expected, the CKKS scheme runs much quicker than the MeKKS scheme due to the more optimised implementation.

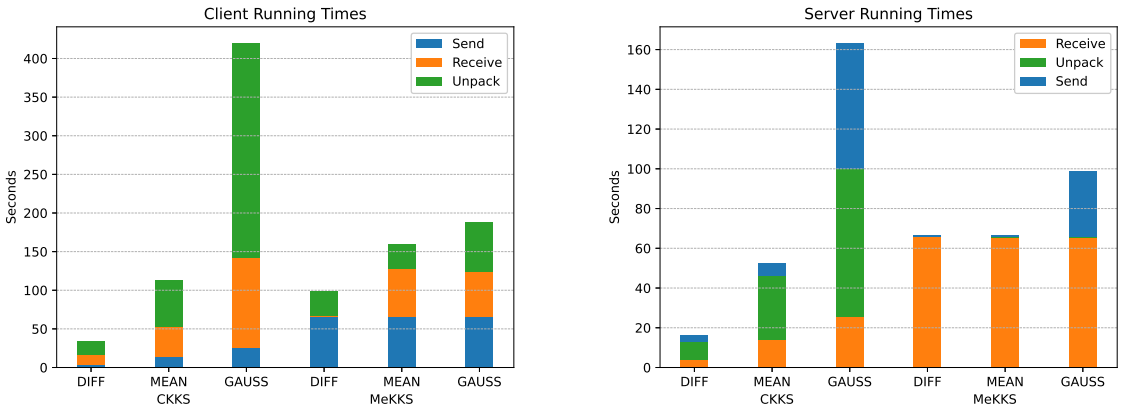
Accuracy

The accuracies of each HE inference algorithm are compared in Figure 4.10. The Moving-MNIST dataset allows accuracy to be easily calculated because it only contains white moving objects on a black background. Therefore, the similarity between the inference result and the original video can be determined by calculating the *sum*



(a) CKKS *(b) MeKKS*

Figure 4.7: Vectorised Memory Usage Compression Comparison



(a) Client Running Times *(b) Server Running Times*

Figure 4.8: Client and Server Running Times

square difference according to Equation 4.1. From this, the accuracy of each encryption scheme can be compared for each inference method.

$$S_{sq} = \sum_{n,m \in N^{N \times M}} (J[n,m] - I[n,m])^2 \tag{4.1}$$

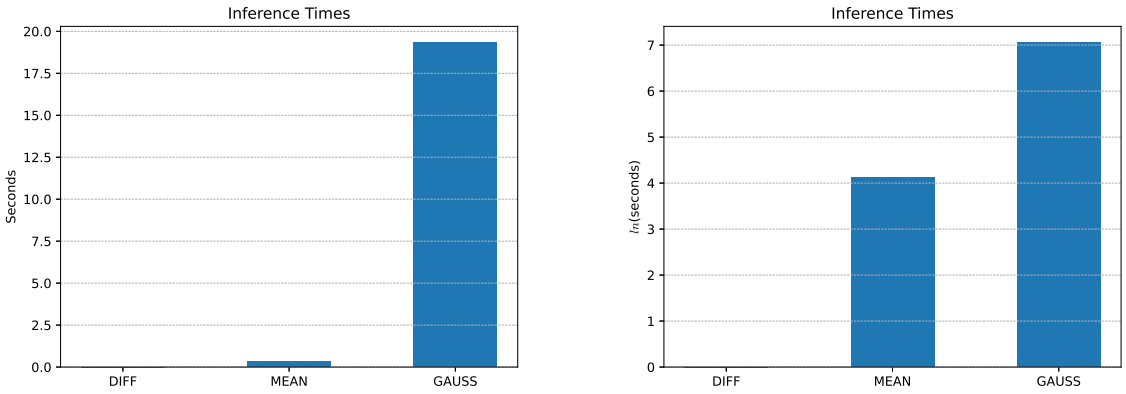
which can be normalised using

$$\frac{S_{sq}}{\sqrt{\sum J[n,m]^2 \times \sum I[n,m]^2}}$$

given two images $J[x,y]$ and $I[x,y]$ with $(x,y) \in N^{N \times M}$.

4.4 Summary

This evaluation provides evidence that combining the domains of homomorphic encryption and moving object detection can produce promising results comparable to existing algorithms extracting moving objects from unencrypted, plain video data.



(a) CKKS *(b) MeKKS*

Figure 4.9: Inference Times

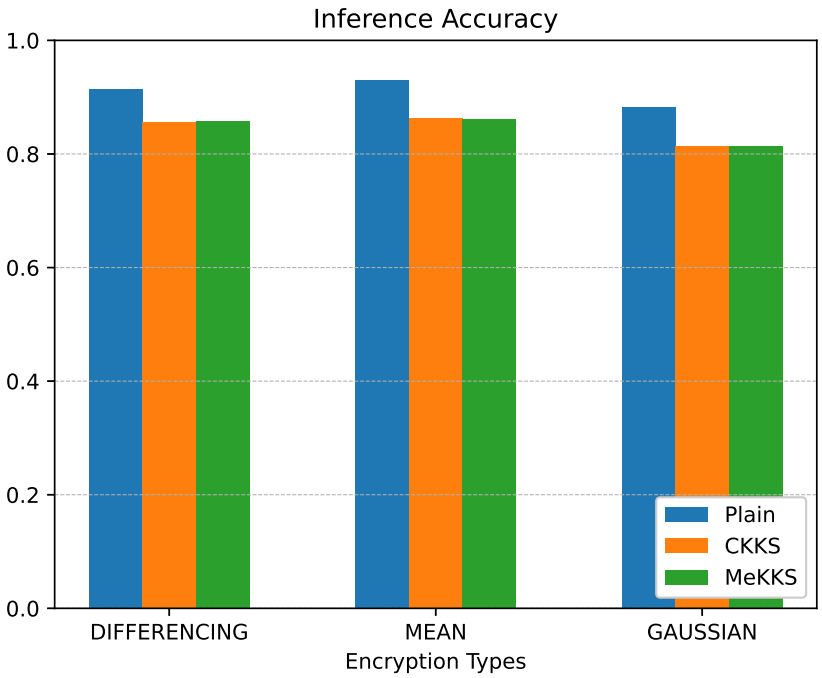


Figure 4.10: Inference Accuracy

Moreover, it demonstrates that progress can be made in improving the performance of systems incorporating these techniques. However, it acknowledges that further research is required into homomorphic primitives to provide more operations on encrypted data and reduce the space complexity of encrypted data to improve the time complexity of data transmission and inference algorithms.

Chapter 5

Conclusions

This Chapter concludes the thesis by summarizing the findings from the study, the contributions and possible limitations of the approach. It can also identify issues that were not solved, or new problems that came up during the work, and suggests possible directions going forward.

Bibliography

[1] https://homomorphicencryption.org/standard/. Retreived March 2022.

[2] Shai Avidan and Ariel Shamir. “Seam Carving for Content-Aware Image Resizing”. In: *ACM Transactions on Graphics* (2007).

[3] Ahmad Al Badawi, Jin Chao, Jie Lin, Chan Fook Mun, Jun Jie Sim, Benjamin Hong Meng Tan, Xiao Nan, Khin Mi Mi Aung, and Vijay Ramaseshan Chandrasekhar. “Towards the AlexNet Moment for Homomorphic Encryption: HCNN, theFirst Homomorphic CNN on Encrypted Data with GPUs”. In: *IEEE Transactions on Emerging Topics in Computing* (2020).

[4] Ho Bae, Jaehee Jang, Dahuin Jung, Hyemi Jang, Heonseok Ha, Hyungyu Lee, and Sungroh Yoon. “Security and Privacy Issues in Deep Learning”. In: *Journal of IEEE Transactions on Arificial Intelligence* (2018).

[5] Serge Beucher and Christian Lantuéjoul. “Use of Watersheds in Contour Detection”. In: *International workshop on image processing, real-time edge and motion detection* (1979).

[6] M. Bhattacharya, R. Creutzburg, and J. Astola. “Some historical notes on Number Theoretic transform”. In: *International TICSP Workshop on Spectral Methods and Multirate Signal Processing* (2004).

[7] Zvika Brakerski. “Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP”. In: *Annual Cryptology Conference* (2012).

[8] Brilliant.org. *Homomorphic Encryption*. https://brilliant.org/wiki/homomorphic-encryption/. Retrieved March 2022.

[9] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. “A Full RNS Variant of Approximate Homomorphic Encryption”. In: *International Conference on Selected Areas in Cryptography* (2019).

[10] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. “Bootstrapping for Approximate Homomorphic Encryption”. In: *Advances in Cryptology* (2018).

[11] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. “Homomorphic Encryption for Arithmetic of Approximate Numbers”. In: *Lecture Notes in Computer Science* (2016).

[12] Jung Hee Cheon, Dongwoo Kim, Duhyeong Kim, Hun hee Lee, and Keewoo Lee. “Numerical Method for Comparison on Homomorphically Encrypted Numbers”. In: *International Conference on the Theory and Application of Cryptology and Information Security* (2019).

[13] Kuan-Yu Chu, Yin-Hsi Kuo, and Winston H. Hsu. “Real-time privacy-preserving moving object detection in the cloud”. In: *Proceedings of the 21st ACM international conference on Multimedia* (2013).

[14] A. P. Dempster, N. M. Laird, and D. B. Rubin. “Maximum Likelihood from Incomplete Data Via the EM Algorithm”. In: *Journal of the Royal Statistical Society* (1977).

[15] Nathan Dowlin, Ran Gilad-Bachrach, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. “CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy”. In: *ICML* (2016).

[16] Ronald Duarte and Resit Sendag. “Accelerating and Characterizing Seam Carving Using a Heterogeneous CPU-GPU System”. In: *Proceedings of the 18th Annual International Conference on Parallel and Distributed Processing Techniques and Application* (2012).

[17] Eufy. *Eufy Homepage*. https://uk.eufylife.com/. Retrieved March 2022.

[18] Junfeng Fan and Frederik Vercauteren. “Somewhat Practical Fully Homomorphic Encryption”. In: *Cryptology ePrint Archive* (2012).

[19] Python Software Foundation. *PSF LICENSE AGREEMENT FOR PYTHON 3.10.4*. https://docs.python.org/3/license.html#psf-license. Retrieved March 2022.

[20] Python Software Foundation. *Python*. https://www.python.org/. Retrieved March 2022.

[21] Git. *Git*. https://git-scm.com/. Retrieved March 2022.

[22] Leo John Grady. “Space-variant computer vision: A graph-theoretic approach”. PhD thesis. Boston University, 2004.

[23] Thore Graepel, Kristin Lauter, and Michael Naehrig. “ML Confidential: Machine Learning on Encrypted Data”. In: *The International Conference on Information Security and Cryptology* (2012).

[24] Dazhou Guo, Yanting Pei, Kang Zheng, Hongkai Yu, Yuhang Lu, and Song Wang. “Degraded Image Semantic Segmentation With Dense-Gram Networks”. In: *IEEE Transactions on Geoscience and Remote Sensing* (2021).

[25] Huelse. *SEAL-Python*. https://github.com/Huelse/SEAL-Python. Retrieved March 2022.

[26] Grupo de Tratamiento de Imágenes. *Labeled and Annotated Sequences for Integral Evaluation of SegmenTation Algorithms*. https://www.gti.ssr.upm.es/data/lasiesta_database. Retrieved March 2022.

[27] Github Inc. *Github*. https://github.com/. Retrieved March 2022.

[28] Open Source Initiative. *The 3-Clause BSD License*. https://opensource.org/licenses/BSD-3-Clause. Retrieved March 2022.

[29] Open Source Initiative. *The MIT License*. https://opensource.org/licenses/MIT. Retrieved March 2022.

[30] Sheldon H. Jacobson and Enver Yücesan. “Analyzing the Performance of Generalized Hill Climbing Algorithms”. In: *Journal of Heuristics* (2004).

[31] Paul Viola andMichael Jeffrey Jones. “Rapid Object Detection using a Boosted Cascade of Simple Features”. In: *Conference on Computer Vision and Pattern Recognition,* (2001).

[32] Chiraag Juvekar, inod Vaikuntanathan, and Anantha Chandrakasan. “Gazelle: A Low Latency Framework for Secure Neural Network Inference”. In: *Proceedings of the 27th USENIX Conference on Security Symposium* (2018).

[33] Andrey Kim, Antonis Papadimitriou, and Yuriy Polyakov. “Approximate Homomorphic Encryption with Reduced Approximation Error”. In: *Topics in Cryptology - CT-RSA* (2020).

[34] Sangpyo Kim, Wonkyung Jung, Jaiyoung Park, and Jung Ho Ahn. “Accelerating Number Theoretic Transformations for Bootstrappable Homomorphic Encryption on GPUs”. In: *2020 IEEE International Symposium on Workload Characterization* (2020).

[35] Laurent Itti Christof Koch and Ernst Niebur. “A Model of Saliency-based Visual Attention for Rapid Scene Analysis”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (1998).

[36] Jaya S. Kulchandani and Kruti J. Dangarwala. “Moving object detection: Review of recent research trends”. In: *2015 International Conference on Pervasive Computing (ICPC)* (2015).

[37] Kim Laine. “Simple Encrypted Arithmetic Library 2.3.1”. In: *Microsoft Research TechReport* (2017).

[38] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. *The MNIST Database of Handwritten Digits*. http://yann.lecun.com/exdb/mnist/. Retrieved March 2022.

[39] Chih-Yang Lin, Kahlil Muchtar, Jia-Ying Lin, Yu-Hsien Sung, and Chia-Hung Yeh. “Moving object detection in the encrypted domain”. In: *Multimedia Tools and Applications* (2017).

[40] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. “On Ideal Lattices and Learning with Errors over Rings”. In: *Journal of the ACM* (2010).

[41] J. MacQueen. “Some methods for classification and analysis of multivariate observations”. In: *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability* (1967).

[42] Khalid El Makkaoui, Abdellah Ezzati, and Abderrahim Beni Hssane. “Challenges of Using Homomorphic Encryption to Secure Cloud Computing”. In: *International Conference on Cloud Technologies and Applications* (2015).

[43] Microsoft. *OneDrive Personal Cloud Storage*. https://www.microsoft.com/en-gb/microsoft-365/onedrive/online-cloud-storage. Retrieved March 2022.

[44] Microsoft. *Visual Studio Code*. https://code.visualstudio.com/. Retrieved March 2022.

[45] Wikipedia user Newton2. *Seam Carving*. https://en.wikipedia.org/wiki/Seam_carving. Retreived March 2020.

[46] Payal V. Parmar, Shraddha B. Padhar, Shafika N. Patel, Niyatee I. Bhatt, and Rutvij H. Jhaveri. “Survey of Various Homomorphic Encryption Algorithms and Schemes”. In: *International Journal of Computer Applications* (2014).

[47] Mike Paterson and Larry J. Stockmeyer. “On the Number of Nonscalar Multiplications Necessary to Evaluate Polynomials”. In: *SIAM Journal on Computing* (1973).

[48] Manas A. Pathak and Bhiksha Raj. “Privacy-Preserving Speaker Verification and Identification Using Gaussian Mixture Models”. In: *IEEE* (2013).

[49] Rob Pike. *Concurrency is not Parallelism*. Waza Conference (Slides: https://talks.golang.org/2012/waza.slide#1). 2012.

[50] Ring. *Ring Homepage*. https://en-uk.ring.com/. Retrieved March 2022.

[51] W. W. Royce. “Managing the development of large software systems: concepts and techniques”. In: *Proceedings of the 9th International Conference on Software Engineering* (1987).

[52] Michael Rubinstein, Ariel Shamir, and Shai Avidan. “Improved Seam Carving for Video Retargeting”. In: *The ACM SIGGRAPH conference proceedings* (2008).

[53] R.R. Schaller. “Moore’s law: past, present and future”. In: *IEEE Spectrum* (1997).

[54] Linda G. Shapiro and George C. Stockman. *Computer Vision*. Pearson, 2001.

[55] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. “Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps”. In: *arXiv:1312.6034* (2014).

[56] Chris Stauffer and W. E. L. Grimson. “Adaptive background mixture models for real-time tracking”. In: *1999 IEEE computer society conference on computer vision and pattern recognition* (1999).

[57] Steven Turner. “Transport Layer Security”. In: *IEEE Internet Security* (2014).

[58] Ioannis Tzemos, Apostolos P. Fournaris, and Nicolas Sklavos. “Security and Efficiency Analysis of One Time Password Techniques”. In: *20th Pan-Hellenic Conference on Informatics* (2016).

[59] *Unsupervised Learning of Video Representations using LSTMs*. http://www.cs.toronto.edu/ nitish/unsupervised_video/. Retrieved March 2022.

[60] C.R. Wren, A. Azarbayejani, T. Darrell, and A.P. Pentland. “Pfinder: real-time tracking of the human body”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (1997).

[61] Greg Yeric. “Moore’s law at 50: Are we planning for retirement?” In: *International Electron Devices Meeting* (2015).

[62] Jingru Yi, Pengxiang Wu, Menglin Jiang, Qiaoying Huang, Daniel J. Hoeppner, and Dimitris N. Metaxas. “Attentive neural cell instance segmentation”. In: *Medical Image Analysis* (2019).

Appendix A

Project Proposal