

Jonathon Ackers

Privacy-Preserving Moving Object Detection

Computer Science Tripos – Part II

Girton College

Friday 13th May, 2022

Declaration of Originality

I, Jonathon Ackers of Girton College, being a candidate for Part II of the Computer Science Tripos, hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose. I am content for my dissertation to be made available to the students and staff of the University.

Signed: Jonathon Ackers

Date: 13th May 2022

Proforma

Candidate Number: **2418D**
Project Title: **Privacy-Preserving Moving Object Detection**
Examination: **Computer Science Trupos – Part II, 2022**
Word Count: **???**¹
Code line Count: **???**
Project Originator: Stephen Cummins and 2418D
Supervisor: Stephen Cummins and Francisco Vargas Palomo

Original Project Aims

An increasing number of smart-home devices are being developed to provide surveillance solutions for customers. While these devices have proved successful in improving the security of people's homes, they introduce new unprecedented privacy risks. This project aimed to investigate homomorphic encryption as a solution to privacy concerns without detracting from the security benefits. To this end, it intended to use existing homomorphic encryption libraries to implement privacy-preserving unsupervised machine learning algorithms that can extract moving objects from surveillance footage. As extensions, the project would implement a bespoke encryption algorithm, investigate further inference algorithms, and validate the security of such algorithms.

Work Completed

Overall, the project was a success. It fulfilled all core success criteria and obtained significant results from inference algorithms and in the optimisations applied to improve performance. Furthermore, the project was extended by implementing a bespoke homomorphic encryption scheme, furthering understanding and providing opportunities for specialisation optimisations. Security of encryption schemes was considered during the preparation stage before implementation, but a formal proof was not completed due to time constraints. Similarly, extending with object recognition was dismissed to provide a more thorough investigation of moving object detection algorithms.

¹Counted with TeXcount.

Special Difficulties

The majority of the project was completed with relatively few unexpected disruptions. However, contracting COVID-19 in December resulted in the project falling several weeks behind schedule. Consequently, significant extra work had to be done to get back on track and ensure the project was completed before the deadline.

Contents

1	Introduction	9
1.1	Motivation	9
1.2	Related Work	10
1.3	Aims and Contributions	10
2	Preparation	12
2.1	Preliminaries	12
2.1.1	Threat Model	12
2.1.2	Homomorphic Encryption	13
2.1.3	Moving Object Detection	16
2.2	Project Strategy	18
2.2.1	Requirements Analysis	18
2.2.2	Methodology	19
2.2.3	Testing	21
2.3	Starting Point	22
2.3.1	Knowledge and Experience	22
2.3.2	Tools Used	23
2.3.3	Computer Resources	24
3	Implementation	26
3.1	Architecture	26
3.1.1	Framework	26
3.1.2	Software Interface	27
3.1.3	Class Structure	28
3.2	Encryption	28
3.2.1	CKKS Primitives	28
3.2.2	MeKKS	34
3.3	Networking	36
3.3.1	Video Size	36
3.3.2	Transmission Rate	39
3.4	Inference	43
3.4.1	Homomorphic Encryption Adaptations	43
3.4.2	Online Mixture Model	44
3.4.3	Expectation-Maximisation Algorithm	46
3.5	Summary	48

4 Evaluation	49
4.1 Requirements Analysis	49
4.2 Homomorphic Encryption Integration	49
4.2.1 Online Mixture Model	51
4.2.2 Expectation-Maximisation Algorithm	52
4.3 Practicality	52
4.3.1 Networking	53
4.3.2 Inference	55
4.4 Summary	58
5 Conclusions	59
5.1 Project Summary	59
5.2 Lessons Learned	59
5.3 Future Directions	59
Bibliography	61
A Homomorphic Encryption Addendum	66
A.1 Group Theory	66
A.2 CKKS Operations	66
B Seam Carving	68
C LASIESTA Inference Results	70
D Project Proposal	72

List of Figures

2.1	The Threat Model	13
2.2	Homomorphic Encryption	14
2.3	The Waterfall Development Model	20
2.4	The Agile Development Model	21
2.5	Project Timeline	21
3.1	High-level implementation overview.	27
3.2	Repository Overview	29
3.3	Client UML Class Diagram	30
3.4	Client UML Class Diagram	31
3.5	MeKKS UML Class Diagram	32
3.6	Bootstrapping Procedure	35
3.7	Potential Seam Costs	37
3.8	Pixel Adjacency Graphs	38
3.9	3D Pixel Adjacency Graphs	38
3.10	Pixel to Region Conversion	39
3.11	Tuning Region Adjacency Graphs	39
3.12	Foveal Sampling	40
3.13	Abstract view of parallel processes	40
3.14	The Sliding Window Protocol	41
3.15	Packing and Unpacking	42
4.1	Moving-MNIST Inference Results	50
4.2	Homomorphic Comparison Algorithm	52
4.3	Homomorphic Division Algorithm	52
4.4	Naïve Packing and Unpacking Times	53
4.5	Optimised Packing and Unpacking Times	54
4.6	Naïve Memory Usage Compression Comparison	55
4.7	Vectorised Memory Usage Compression Comparison	56
4.8	Client and Server Running Times	56
4.9	Inference Times	57
4.10	Inference Accuracy	57
4.11	Energy Usage	58
A.1	CKKS Operations	67
B.1	Seam Carving Stages	69
C.1	LASIESTA Inference Results	71

List of Tables

2.1	Licenses	24
2.2	Computer Specifications	25
4.1	Requirements Analysis	51
4.2	Packing and Unpacking Improvements	54
4.3	Compression algorithms	55

Chapter 1

Introduction

1.1 Motivation

Computers have improved almost every aspect of modern life. Recently, home security has become the latest target of the technology revolution. Companies like Ring [RING] and Eufy [EUFY] 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 analyse footage. For example, a doorbell may recognise a visitor or alert to the presence of a stranger. However, the computational intensity of inference means footage must be transferred to more powerful servers.

To preserve security, video is encrypted before transmission 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. Consequently, malicious actors could use this information to monitor peoples' location, appraise their belongings, use the contents of footage for extortion, and more. Homomorphic Encryption (henceforth HE) may provide a solution to this.

In cryptography, HE describes encryption schemes that allow mathematical operations to be performed directly on encrypted data, or *ciphertext*, rather than on raw data, or *plaintext*. For example, consider 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 is fundamental to surveillance. Detecting when, for example, somebody enters a property allows security systems to alert their owners, possibly pre-empting a break-in. However, more than just motion must be sensed. Objects must be extracted and analysed to prevent users being notified of unimportant events like, for example, leaves blowing onto a property. Gradual changes and random noise in an environment make modelling a background for object detection a significant challenge to overcome.

1.2 Related Work

There have been many attempts at solving video inference in the encrypted domain, but none are without flaws. In 2013, Chu et al. [14] proposed an encryption scheme that supports real-time moving object detection. However, this was quickly shown to suffer from information leakage, leaving it vulnerable to chosen-plaintext attacks¹. Similarly, in 2017, Lin et al. [39] proposed an encryption scheme to achieve the same goal by only encrypting some of the bits in each pixel, but this is unprotected against steganographic² attacks. Therefore, while research has solved the weaknesses in privacy, it is yet to offer a solution that also preserves security, making them useless to real-world applications.

Likewise, researchers have been investigating inference using HE for many years. In 2012, Graepel et al. [24] introduced machine learning in the HE domain. Dowlin et al. [17] 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, Gaussian Mixture Models (GMMs) are the most common technique for background modelling. There is much less HE research into this area of unsupervised learning. The best example comes from 2013 when Pathak and Raj [47] proposed a HE implementation of a GMM for audio inference. But there do not seem to be any investigations linking HE and GMMs to video analysis.

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 by investigating the limitations of current HE implementations concerning surveillance.

1.3 Aims and Contributions

This dissertation documents the design and implementation of an investigation into HE surveillance to answer the questions posed in §1.1. In particular, the following contributions.

- Creation of a client-server application simulating the device-server stack utilised by existing surveillance devices like doorbell cameras, allowing secure transmission of video data from client to server and back again after performing inference.
- Integration of Microsoft's Secure Encrypted Arithmetic Library (SEAL) [37] to allow secure, private inference of videos encrypted using the CKKS HE scheme [12].
- 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.

¹A *chosen-plaintext attack* is a scenario in which an adversary can freely encrypt plaintexts of their choosing and analyse the resulting ciphertexts.

²*Steganography* describes the technique of securing messages through information hiding. Unlike cryptography, where the existence of a message is known, but its contents are not, steganography attempts to hide the message's existence.

- Optimisations to reduce transmission time from the perspectives of reducing memory usage of HE data and increasing the transmission rate between the client and server.
- Investigation of GMMs for HE encrypted background subtraction, beginning with the work by Stauffer and Grimson [55] then moving into more general Expectation-Maximisation GMM algorithms [16].
- Implementation of the CKKS scheme from scratch in Python, called MeKKS, based on the Homomorphic Encryption for Arithmetic of Approximate Numbers paper by Cheon et al. [12, 10] to improve understanding of HE.
- Evaluation of the efficacy of the above solutions using timing, accuracy, and 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 three domains of computer science: cryptography, graphics, 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 in places where further explanation is required.

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, exposing a range of privacy risks including identity theft, monitoring intimate behaviours of household members, observations of household contents, and more. Figure 2.1 illustrates this succinctly. The first threat can be mitigated easily enough using cryptographic protocols such as TLS [56]. However, the second threat is much more difficult to defend against, particularly because data must usually be decrypted for inference to be performed [5].

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.

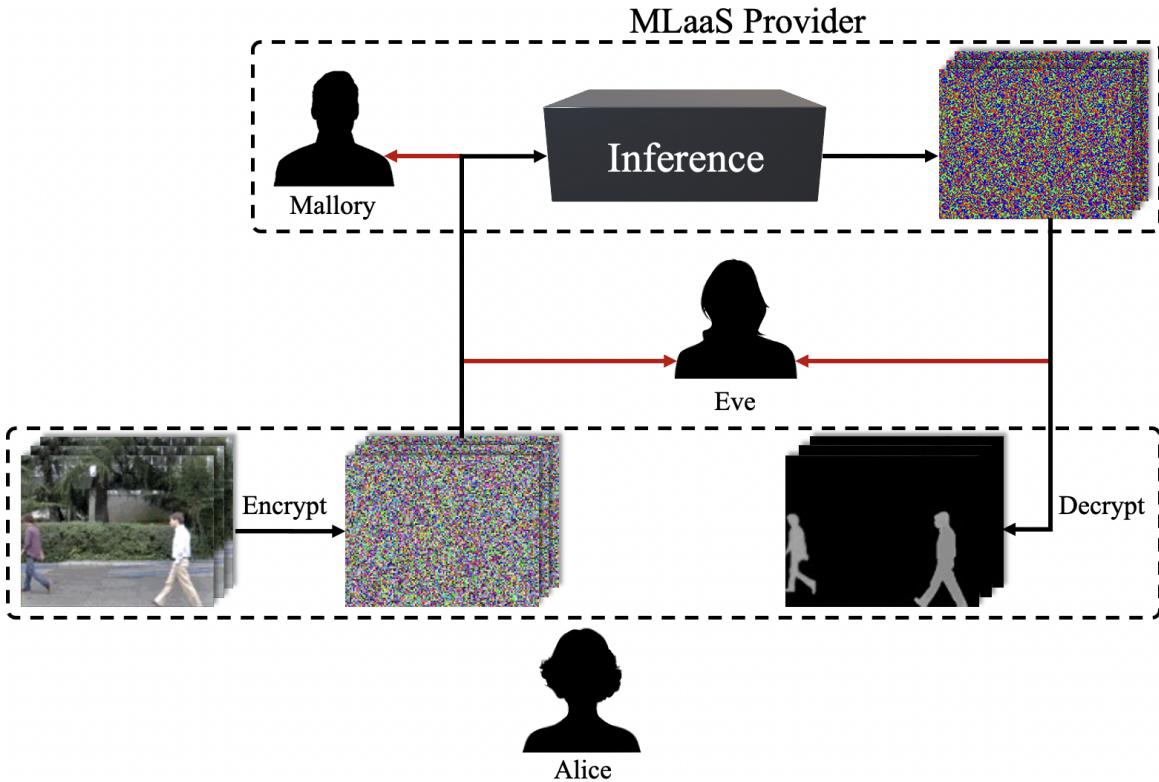


Figure 2.1: A graphical representation of the threat model. Eve is an eavesdropper able to listen to communications between user and service provider. Mallory is a malicious individual within the service provider able to access users' video data. HE is able to obstruct both Eve and Mallory, preventing 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 $\Pi = (\text{KeyGen}, \text{Enc}, \text{Dec})$. KeyGen is a function used to generate a *public key* (PK) and *private key*¹ (SK) such that $(\text{PK}, \text{SK}) \leftarrow \text{KeyGen}(1^l)$, where the security parameter, l , measures how hard it is for an adversary to break the scheme². 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 $c \leftarrow \text{Enc}_{\text{PK}}(m)$. Similarly c is decrypted back into m by $m \leftarrow \text{Dec}_{\text{SK}}(c)$.

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

$$\text{Dec}_{\text{SK}}(\text{Eval}(f, c_1, \dots, c_n)) = f(m_1, \dots, m_n) \quad (2.1)$$

¹The *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.

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

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

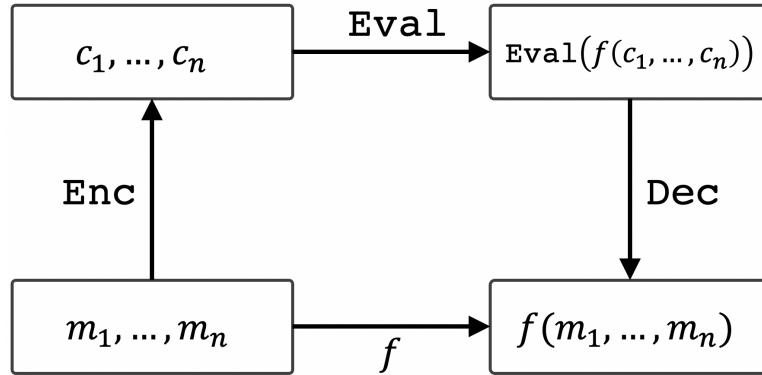


Figure 2.2: Homomorphic Encryption

Theoretically, a *fully* homomorphic scheme³ 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.

$$\text{IP}(M = m | C = c) = \text{IP}(C = c) \quad (2.2)$$

$$\text{IP}(C = c | M = m) = \text{IP}(M = m) \quad (2.3)$$

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 applications⁴. 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 large number on a classical computer - the

³The 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 [45].

⁴For example, the One-Time Pad [57]

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.

To understand the RLWE problem, some knowledge of group theory is required. The relevant knowledge is introduced in Appendix A.1. 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 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 *cyclotomic polynomial* for $\Phi(X)$, where the n -th cyclotomic polynomial, $\Phi_n(X)$, is defined as

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

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.

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

⁶A specialisation of the discrete Fourier transform, the NTT is a generalisation of the Fast Fourier transform in the case of finite fields. [7]

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 [52].

There are two broad categories of segmentation techniques: *semantic* [25] and *instance* [60]. This dissertation will focus on semantic segmentation through moving object detection because videos will be segmented into two categories: foreground and background, rather than labelling individual instances of objects.

This application is known as either *foreground extraction* or *background subtraction*. To perform this inference, the background of an image must be modelled so that changes in the scene can be detected. This is an unsolved problem. Image data can be very diverse, with variable lighting and repetitive movements (like leaves and shadows) making robust models hard to develop.

Once a background model has been developed, it can be used to detect moving objects in videos by comparing each frame to 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

Frame differencing is the most straightforward background subtraction algorithm [32]. First, a reference frame is established. There are several options for this, with the most common options being to select the first frame of the video or continually updating to the frame previously received. The challenge comes from ensuring that the model can adapt to permanent changes, such as a fence being added to a property. However, updating the reference frame too frequently could cause issues with slow-moving objects becoming occluded.

Once the reference frame, B , has been selected, the foreground can be extracted. Denoted each 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 , $P(f_t)$. The mathematical definition is given by Equation 2.5.

$$P(F_t) = P(f_t) - P(B) \quad (2.5)$$

where F represents the frames in the resultant video.

Mean Filter

A *mean filter* approach to moving object detection attempts to overcome the weaknesses of frame differencing in selecting a reference frame [61]. Instead of taking a frame directly from the video, the value of B at time t is calculated using Equation 2.6.

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

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 [61]. However, the median of the preceding N frames is used to calculate the reference frame rather than the mean.

Then, like before, 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 [WREN]. Rather than storing a simple reference image that is subtracted from each frame in the video, this method stores a mean and variance for each pixel in a frame. When a new frame is received, the likelihood of each pixel is calculated. Assuming that the background is the most common value for a pixel, the foreground can be segmented by grouping the pixels with a sufficiently low likelihood.

Rather than recalculating the mean every time a frame is received - giving an $O(n^2)$ time complexity - the algorithm can be implemented using cumulative functions for the mean and variance to achieve a linear time complexity. The functions are defined by Equation 2.7 and Equation 2.8 respectively.

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

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

where α determines the size of the *temporal window* used to fit the Gaussian model⁷, $d = |f_t - \mu_t|$ 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,

$$\frac{|f_t - \mu_t|}{\sigma_t} > k \quad (2.9)$$

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

A variant of this algorithm exists where the model is 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. However, it requires the entire frame to be initially background and struggles to cope with changes to the background.

⁷ α acts similarly to a decay factor, weighting the most recent frames more importantly. 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 [55]. 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.

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*⁸ distribution over components such that $\text{IP}(x \text{ generated by component } k) = \phi_k$. If the component weights are learned, they are known as *a-posteriori*⁹ 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 tuned to match the distributions observed in the data. If all of the data is known, it can be incorporated into this stage to achieve the most accurate fitting. However, only a subset of the data will likely be available in practice, so the GMM will have to extrapolate to the actual values. This technique can also be taken advantage of if there is limited time to perform fitting.

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 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. Consequently, the requirements subtly evolved as understanding matured. The original requirements are given in Appendix D 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 or further the investigation into HE and surveillance.

⁸A probability derived purely through deductive reasoning.

⁹From Bayesian statistics, referring to conditional probability $\text{IP}(A|B)$.

Core

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

This component provides the foundation for implementing and integrating all other components. It is essential for emulating the MLaaS software stack. While conceptually simple, incorporating HE data adds challenges that formed 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 within the HE domain.

- 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 specialisation optimisations.

- 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 [DOWLIN] 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.

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 [49]. 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.

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

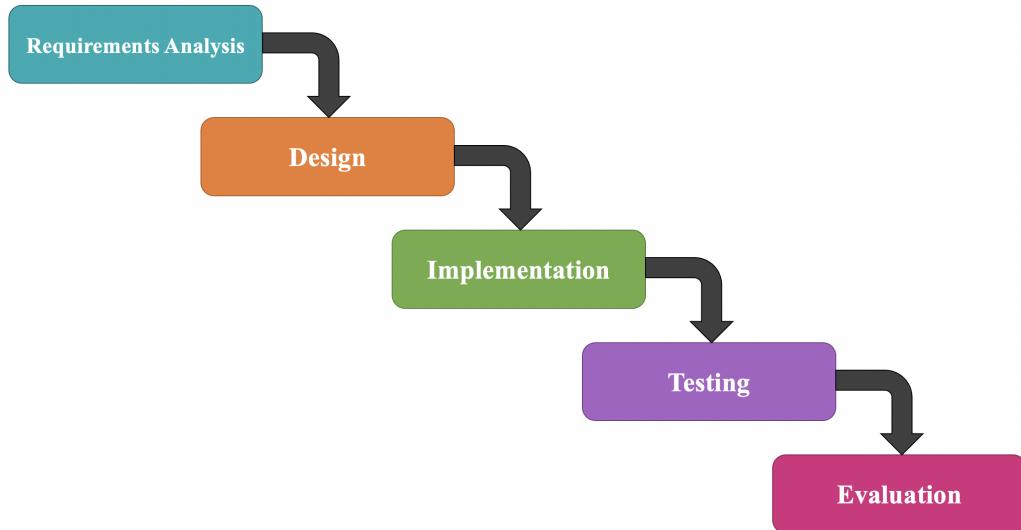


Figure 2.3: The Waterfall Development Model

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 adopted.

The *evaluation* stage replaces the *maintenance* stage of the traditional Waterfall model. This stage involves running experiments to evaluate the project.

When working on extensions, an iterative model was more appropriate. The main reason for this was that less time had been dedicated to researching these components, so implementation was riskier. Consequently, a rapid cyclical development model requiring components to be decomposed would allow any problems to be discovered sooner, limiting impacts. Therefore, the Agile model, depicted in Figure 2.4 was selected. Moreover, regular supervisor meetings allowed Agile's sprint system to be utilised so that a project prototype could be presented in each meeting to ensure thorough progression tracking.

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

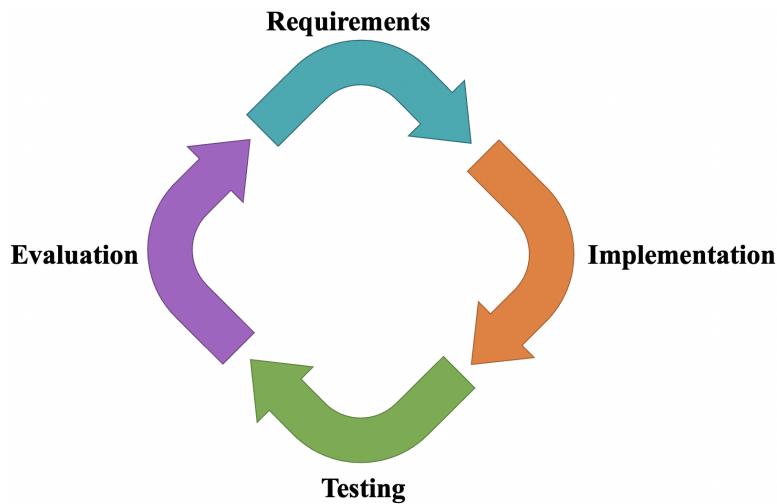


Figure 2.4: The Agile Development Model

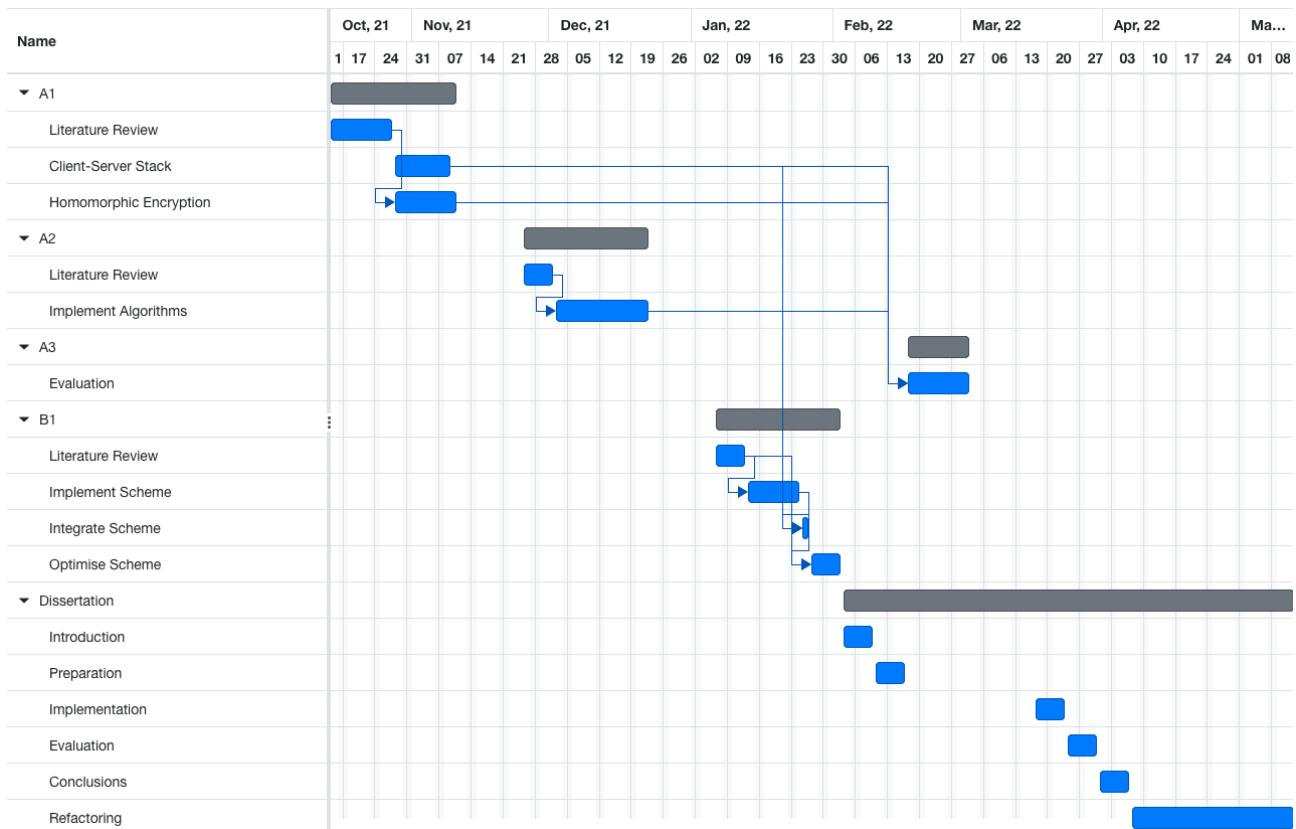


Figure 2.5: Project Timeline

2.2.3 Testing

Unlike traditional software engineering, machine learning does not provide precise criteria against which correctness 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 utilises the independent nature of components written following object-oriented principles to elements in isolation. It does this by providing known expected, boundary, and erroneous data and ensuring function results 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 completing the first extension for verifying the correctness of the encoding, encryption, and decryption functions and the HE Boolean circuits.

Integration Tests

Integration tests increase the scope of functionality covered by each test. Their goal is to ensure separate modules interact correctly. Once unit testing has been completed, these tests aggregate verified modules and provide similar 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 if 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.

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, most understanding came from academic papers; notably [12] and [37]. Although, articles such as [9] 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

[55] 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* [21]. 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* [22] and *GitHub* [27] were used for version control and source code management. *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 [12] was selected because it supports representing real numbers¹⁰. 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 [26].

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

¹⁰As opposed to, for example, the BFV scheme, which only supports integers [8, 19].

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. [54, 38].

- The LASIESTA dataset contains a variety of sequences showing people moving across static backgrounds. 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. [15].

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 PyJoules	MIT [29]
Matplotlib Python Tkinter	PSFL [20]

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 understanding the MLaaS threat model described in §2.1.1. From this, Figure 3.1a depicts a high-level layout of the project's core components. The components have been split into two categories: *online* and *offline*. Online describes inference being performed directly in response to a request from the user. Offline describes generating inference results on batches of data, independent of the front-end.

- The online components responsible for emulating the MLaaS model are where most of the discussion of this dissertation occurs. 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. The result is then passed through the *client* to the *server*. In the *inference* component, the data is 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 during implementation and evaluation. The *testing* component refers to developing and refining the inference algorithms used to extract moving objects. The *evaluation* component encompasses evaluating the application, including both inference and networking.

Figure 3.1b provides 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. A layer that may be particularly relevant to this investigation is 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 [4]. Equally, the hardware used in current surveillance implementations may produce weaker results.

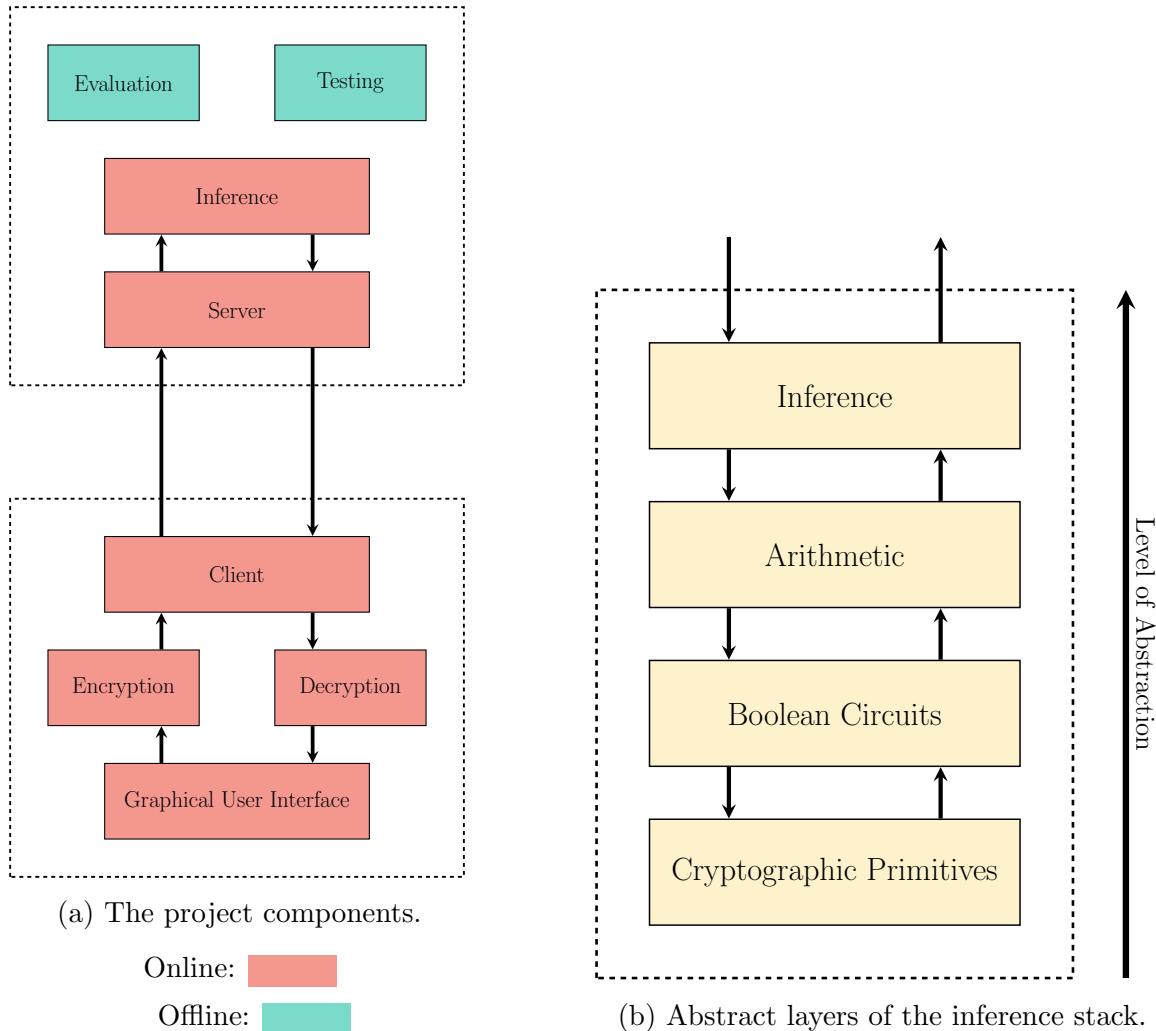


Figure 3.1: High-level implementation overview.

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 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 interacts with. It allows the encryption scheme and inference method to be configured, and videos to be uploaded.
- 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 scheme can be easily substituted.

- The lowest level contains the cryptographic primitives. Contained within the `lib` folder, the libraries `Seal-Python` and `MeKKS` contain the implementations of these primitives.

3.1.3 Class Structure

This section provides a more thorough 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 overlaps between diagrams exist, they have been separated into three figures for 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. [12], 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. Ciphertext polynomials can be decrypted to retrieve approximations of plaintext polynomials, with the accuracy depending on the number of operations performed.

To encrypt vectors of real values, they must 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 so precision is preserved by multiplying by a *scaling factor*, Δ . Once the vector has been encoded, 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 $\Delta^2(\mathbf{v} \odot \mathbf{w})$. This implies that a sequence of multiplications will 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 applied to 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 $Q = q_0\Delta^L$, 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 .

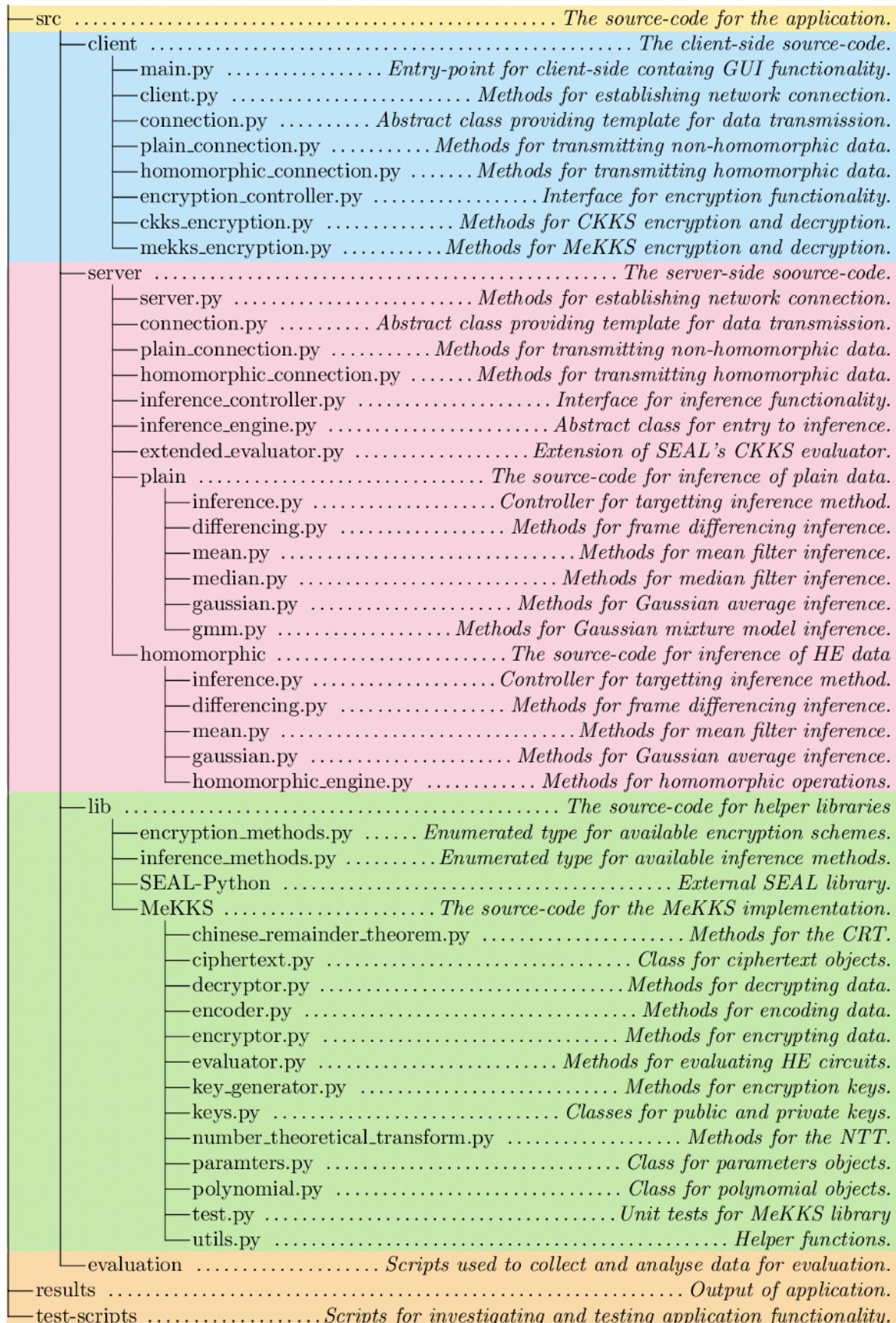


Figure 3.2: Repository Overview.

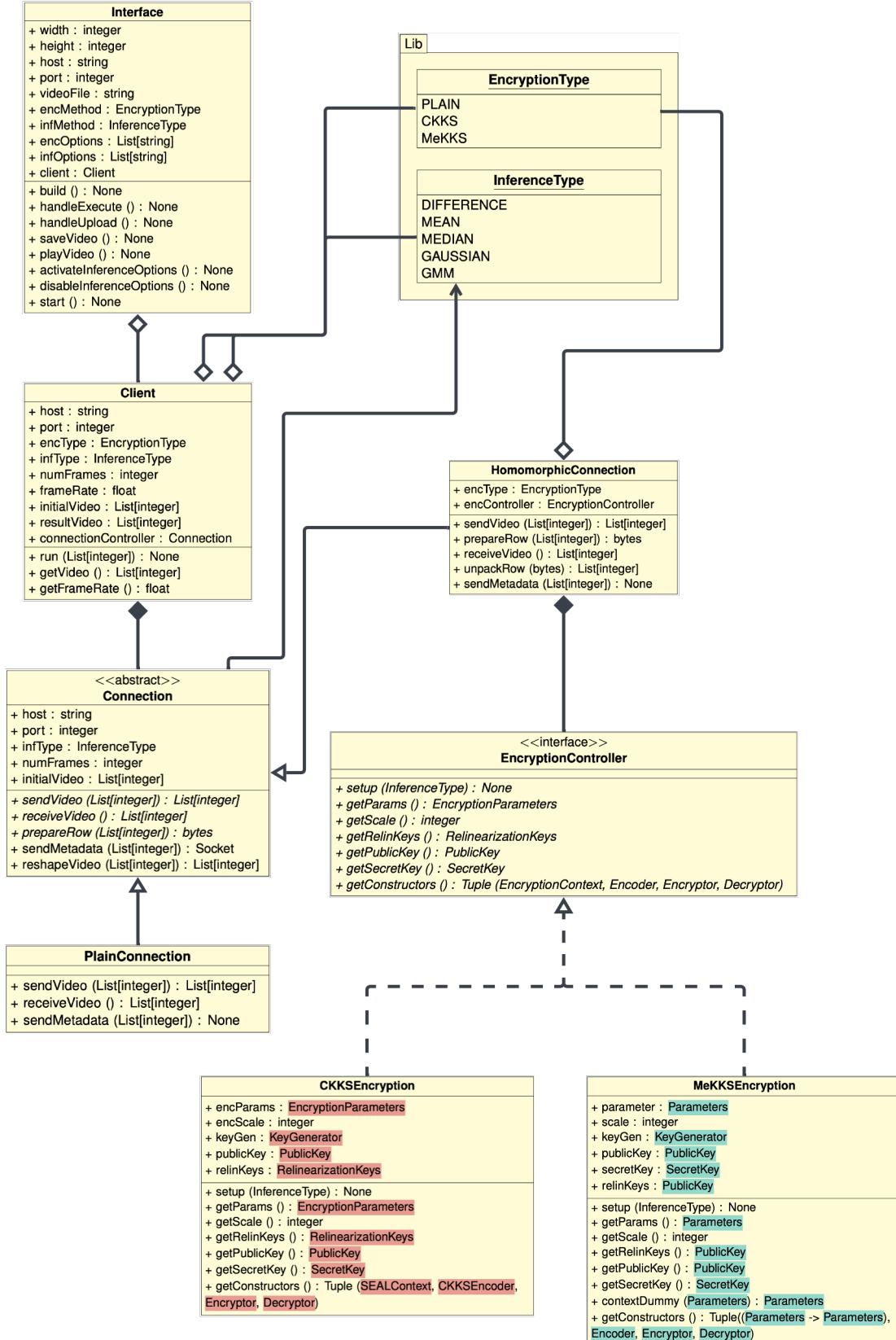


Figure 3.3: UML Class Diagram showing the components of the client side.

External CKKS classes:

External MeKKS classes:

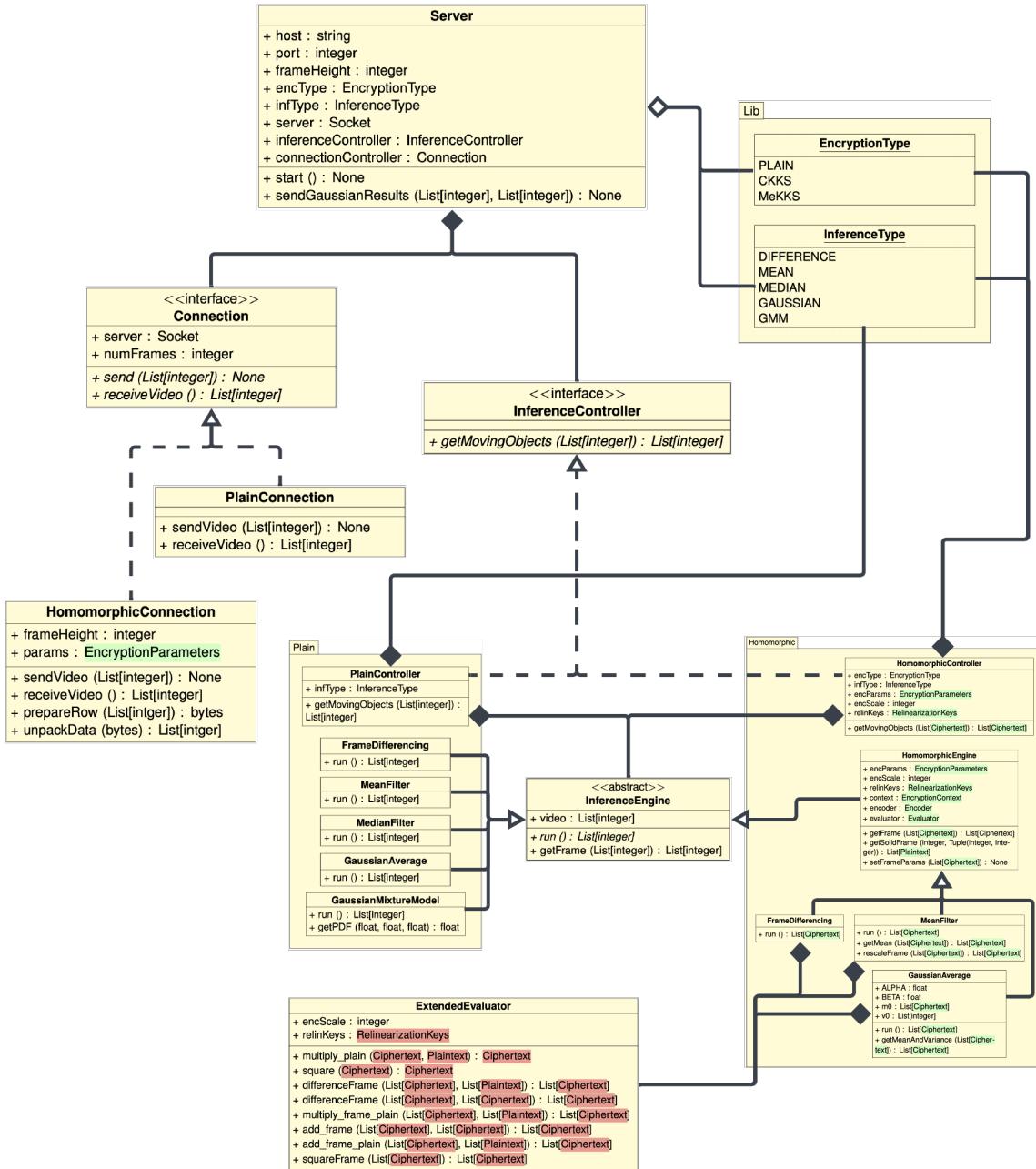


Figure 3.4: UML Class Diagram showing the components of the server side.

External CKKS classes: Abstract HE objects:

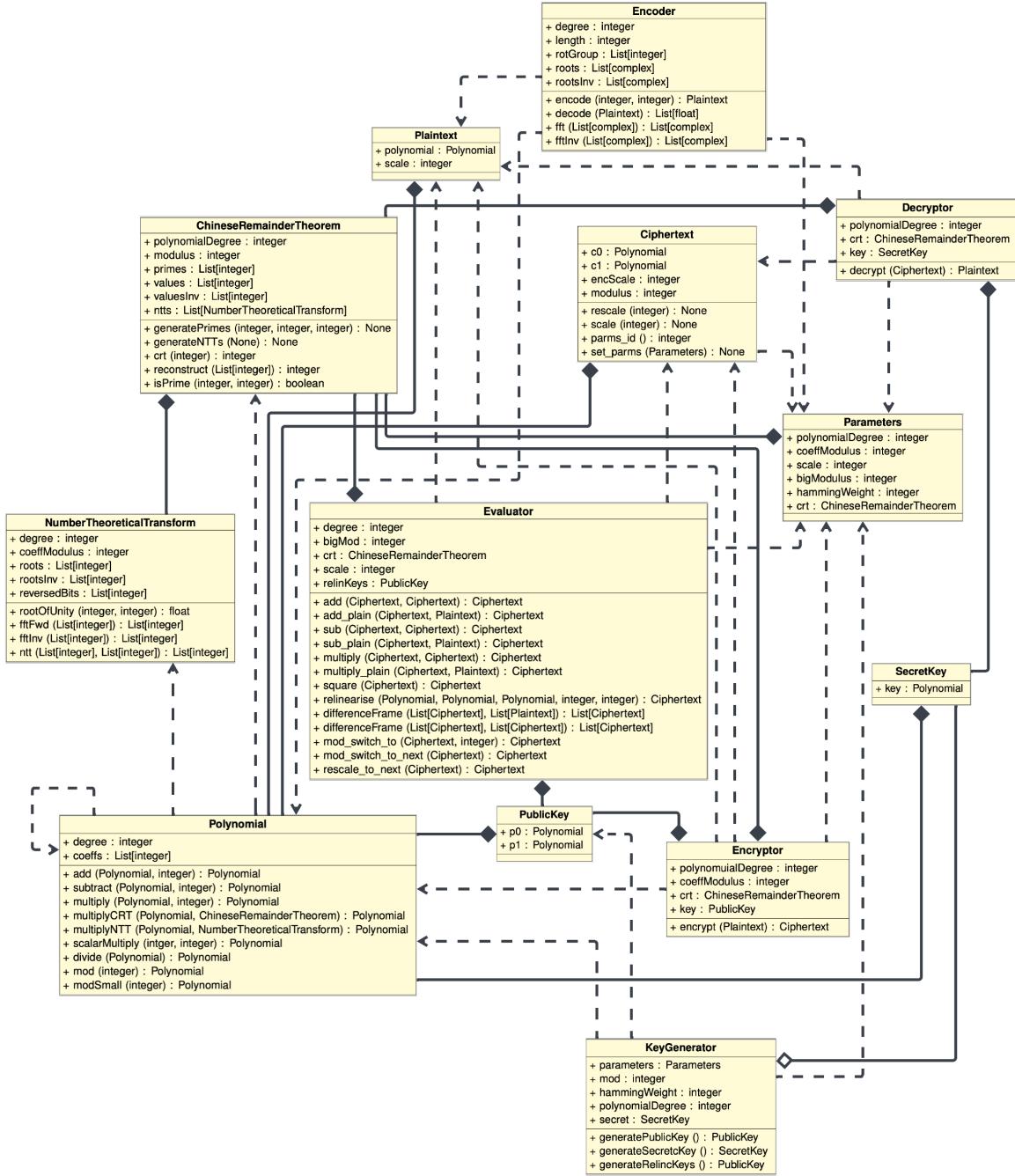


Figure 3.5: UML Class Diagram showing the components of the MeKKS encryption scheme.

Encoding and Decoding

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 it.

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 $\sigma : \frac{\mathbb{R}[X]}{X^N + 1} \rightarrow \mathbb{C}^N$. This is applied to each root of $\Phi_M(X)$ through the evaluation of r as

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

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.

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

Operations

A full list of the operations supported by CKKS and their definitions is included in Appendix A.2.

Rescaling

The rescaling function introduced above is defined by equation 3.3.

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

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 scale Δ^2 down to Δ .

Rotations

An advantage of RLWE based HE schemes over others is the ability to *rotate* ciphertext slots. Given a vector $\mathbf{V} = [v_0, v_1, v_2, \dots, v_n]$ and an offset d . A rotation would cyclically shift each element by d indices¹. 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 [31].

RNS Optimisations

SEAL utilises a *residue number system* [11] to overcome the slow computations caused by very large polynomial coefficients requiring arbitrary precision arithmetic [RNS]. This

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

exploits the Chinese Remainder Theorem (CRT) to decompose the coefficients in \mathbb{Z}_q into n smaller ones: $\mathbb{Z}_{p_1}, \dots, \mathbb{Z}_{p_n}$. To do this, the CRT uses a *moduli switching chain* p_1, \dots, 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, \dots, 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 $\mathbb{Z}_{p_1}, \dots, \mathbb{Z}_{p_n}$.

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 scaling factor must be manually reset to Δ after each multiplication and rescaling.

3.2.2 MeKKS

Initially, the goal of implementing the CKKS scheme from first principles, named MeKKS, was to understand its fundamental mathematical principles further. There were little to no expectations of providing a performance improvement. However, during implementation, it became apparent that there were advantages to a specialised HE scheme that can overcome weaknesses found in SEAL.

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 [11, 10, 33]. The opportunities for developing MeKKS are extensive. However, it was essential to define a clear endpoint for the implementation to schedule the project. Therefore, the performance expectations compared to SEAL were significantly bounded.x

Implementation

The first iteration of MeKKS implementation began using the scheme outlined by Cheon et al. in 2016 [12]. This allowed all of the functionality described in §3.2.1 to be implemented following the SEAL API, allowing for easy integration with the existing application.

However, the performance of this implementation meant it was infeasible to integrate into the rest of the application. Therefore, the bootstrapping procedure from the next HEAAN iteration was used to speed up computation [10]. Bootstrapping takes advantage of the *approximate computation* characteristic of HEAAN to evaluate the decryption formula

approximately so 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 operations - where the error induced by the approximation is small enough to maintain precision.

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

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

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

The Taylor polynomial can be used to approximate the trigonometric function so that it can be evaluated 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, 46]. However, the complexity of recryption grows exponentially with the depth of the decryption circuit, which still has a substantial impact.

Figure 3.6 provides a graphical representation of the bootstrapping approximation.

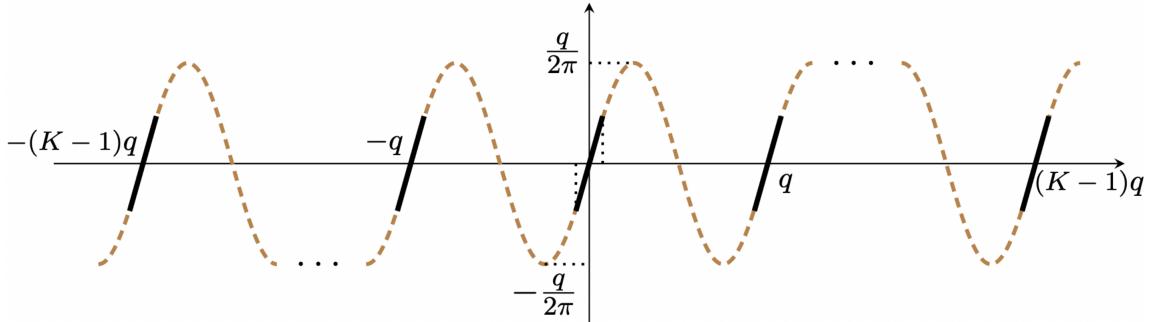


Figure 3.6: Modular Reduction and Scaled Sine Functions. Reproduced from [10].

Optimisations

The principal optimisations of MeKKS came through data representation. When integrating SEAL, a Python wrapper for a C++ implementation was used. Consequently, the application was forced to handle large C++ objects, abstracted through the wrapper. Therefore, Python struggled to manipulate these objects efficiently, specifically when serialising data for transmission. 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 schedule ensured MeKKS was only implemented once the application's core had been complete. As a result, the functionality required had been finalised and only needed operations were implemented for MeKKS. 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 details the techniques investigated for optimising communication between the client and server. An established limitation of HE is the size of ciphertexts [42]. Consequently, the *transmission time* of data is significantly impaired. The *transmission time* can be defined by Equation 3.5.

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

Transferring large volumes of data to the cloud is a critical component of the MLaaS model, so a substantial portion of the investigation was dedicated to reducing transmission time. The problem was considered from two angles: attempting to reduce the *video size* and attempting to increase the *transmission rate*.

3.3.1 Video Size

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* [3]. Consequently, an image can be resized while preserving important features, such as people or buildings. There are two categories for distinguishing these features. Firstly, *top-down* methods use tools such as *face detectors* to highlight where the features appear in the image [2]. Whereas a *bottom-up* approach uses saliency maps² to locate the most important [35].

The details of the original seam carving paper, as well as a depiction of the algorithm, are included in Appendix B. However, a more advanced algorithm was implemented for this investigation to provide more optimal results.

To quantify the importance of a pixel, seam carving defines an *energy function*. Rubinstein et al. [50] proposed the *forward energy* function using dynamic programming. This method calculates the energy of a pixel by accounting for the impact on future energies if it is removed. To achieve this, the *energy difference* function is defined by Equation 3.6. The cost of removing pixels, C , is measured as the forward differences between the pixels that would become neighbours after deletion. There are three cases for this: diagonally adjacent in each direction and orthogonally adjacent, depicted by Figure 3.7 and defined by Equation 3.7.

$$\Delta E_{i+1} = E(\mathbf{I}_{i+1}) - (E(\mathbf{I}_i) - E(C_i)) \quad (3.6)$$

$$C_L(i, j) = |\mathbf{I}(i, j + 1) - \mathbf{I}(i, j - 1)| + |\mathbf{I}(i - 1, j) - \mathbf{I}(i, j - 1)| \quad (3.7)$$

$$C_U(i, j) = |\mathbf{I}(i, j + 1) - \mathbf{I}(i, j - 1)| \quad (3.8)$$

$$C_R(i, j) = |\mathbf{I}(i, j + 1) - \mathbf{I}(i, j - 1)| + |\mathbf{I}(i - 1, j) - \mathbf{I}(i, j + 1)| \quad (3.9)$$

²a representation highlighting the regions of an image where a person's eyes are first drawn, see [53].

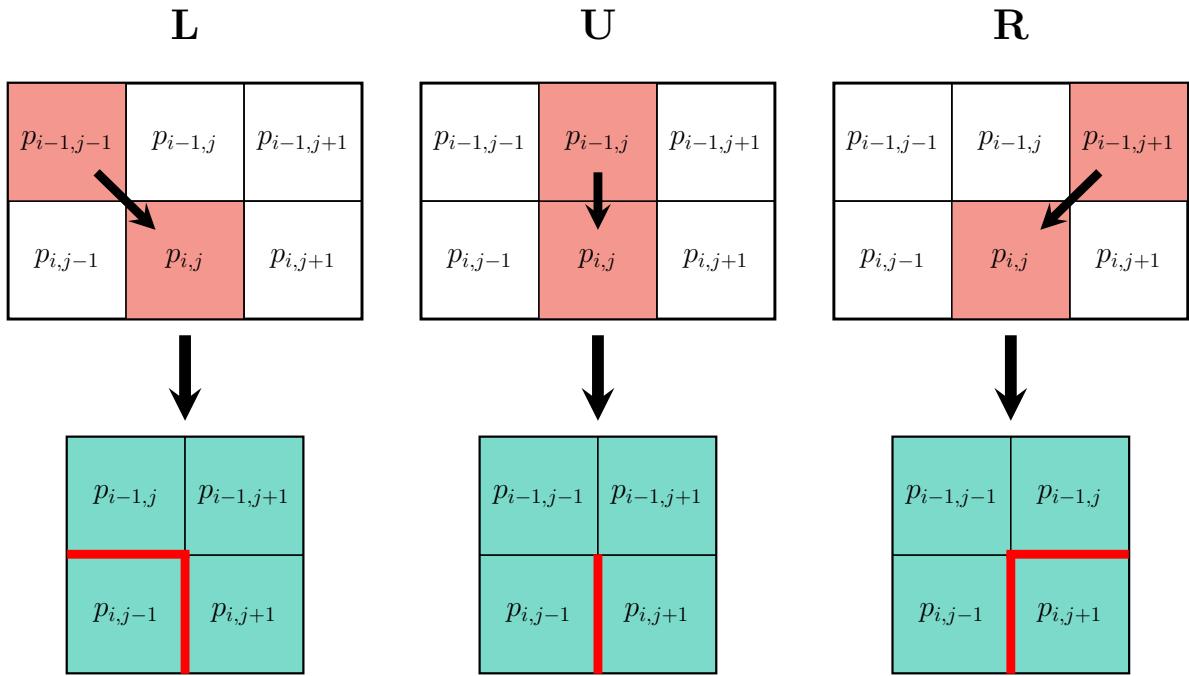


Figure 3.7: There are three possible vertical seam costs for pixel (i, j) . After removing the seam, new neighbours (blue) and new edges (red) are created.

Once the energy 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 row in the path. Likewise, a *horizontal seam* connects the left of an image to the right, such that only a single pixel from each column is included. Formally, this is defined by Equation 3.10a and Equation 3.10b respectively.

$$\mathbf{s}^x = \{s_i^x\}_{i=1}^n = \{(x(i), i) \text{ s.t. } \forall i |x(i) - x(i-1)| \leq 1\}_{i=1}^n \quad (3.10a)$$

$$\mathbf{s}^y = \{s_j^y\}_{j=1}^m = \{(j, y(j)) \text{ s.t. } \forall j |y(j) - y(j-1)| \leq 1\}_{j=1}^m \quad (3.10b)$$

for an $n \times m$ image, \mathbf{I} , where $x : [1, \dots, n] \rightarrow [1, \dots, m]$ and $y : [1, \dots, m] \rightarrow [1, \dots, n]$.

Using these definitions and the energy function, the *optimal seam* can be found. That is, the seam that minimises the *seam cost* - the sum of all pixel costs in the path. Some implementations will use variants of Dijkstra's algorithm for this. Alternatively, Equation 3.11 defines a dynamic programming approach.

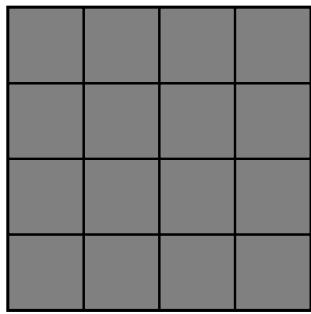
$$M(i, j) = e(i, j) + \min \begin{cases} M(i-1, j-1) + C_L(i, j) \\ M(i-1, j) + C_U(i, j) \\ M(i-1, j+1) + C_R(i, j) \end{cases} \quad (3.11)$$

It is important to note that there have been several extensions to seam carving that may apply to this project. Particularly, optimisations for videos by introducing two-dimensional seams to allow time to be accounted for, and implementations using GPUs to reduce execution time [50, 18].

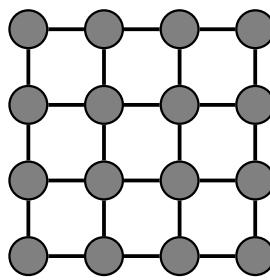
Graph Representation

Representing images using graphs has several advantages. Firstly, graphs are discrete, mathematically simple objects with an established set of provably correct algorithms. More pertinent to this investigation, graphs provide flexible representations that can be used to tune image size.

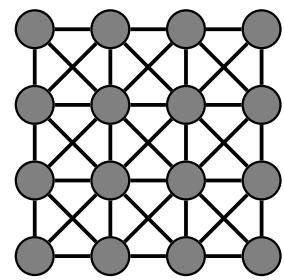
Graph-based image processing methods operate on *pixel adjacency graphs* - graphs whose vertex set is the set of image pixels and edge set defines adjacency of pixels. An example of some pixel adjacency graphs is given by Figure 3.8. Three-dimensional pixel adjacency graphs account for relationships between video frames when handling video files. An example of these is depicted by Figure 3.9.



(a) A 2D image with 4×4 pixels.

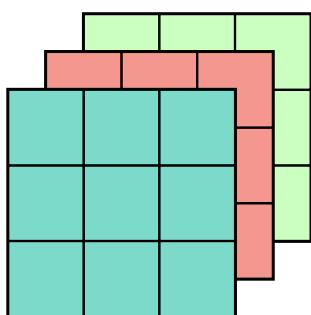


(b) A 4-connected pixel adjacency graph.

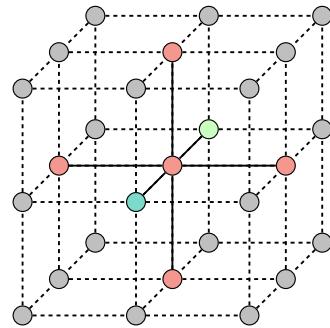


(c) An 8-connected pixel adjacency graph.

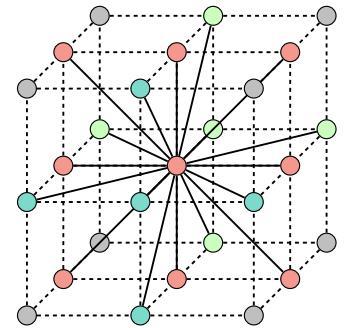
Figure 3.8: Pixel adjacency graphs.



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



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

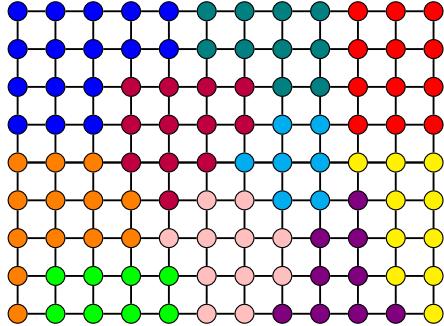


(c) An 18-connected 3D pixel adjacency graph.

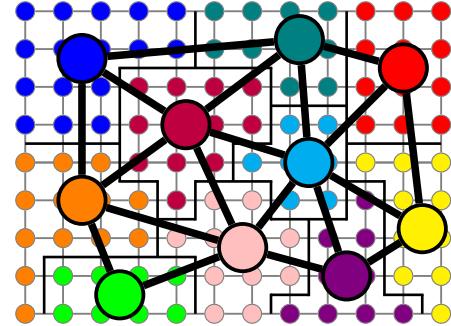
Figure 3.9: 3D pixel adjacency graphs.

However, to improve the video size, pixel adjacency graphs must be extended to *region adjacency graphs*. In this case, rather than representing each pixel with a node, pixels are amalgamated into regions represented by a single node. Figure 3.10 provides a pictorial example of this.

To achieve this, the notion of similarity between pixels must be quantified. Perhaps surprisingly, this is another image segmentation problem. Consequently, established algorithms producing valid solutions exist. Unsupervised clustering algorithms such as *the watershed transform* [6] or *k-means clustering* [41] are two such methods that have proved useful in existing works.



(a) A pixel adjacency graph.



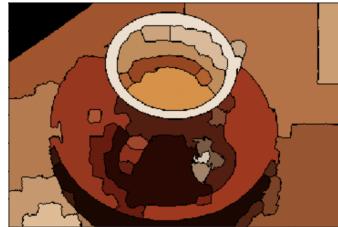
(b) The corresponding region adjacency graph.

Figure 3.10: Converting a pixel adjacency graph to a region adjacency graph.

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 when determining regions [23]. The *Fovea centralis* is a region of the retina responsible for the sharp central vision used by mammals to focus on particular objects. Consequently, its shape can be used to bias the placement of nodes of a graph to prioritise more critical areas. This allows the region budget to be used more efficiently to reduce noticeable quality reduction. Other techniques have been developed using saliency maps or similar.



(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 3.11: A demonstration of the impact of reducing the number of regions on image quality. Images taken from [23].

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 [23]. 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 overlaid 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.

3.3.2 Transmission Rate

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

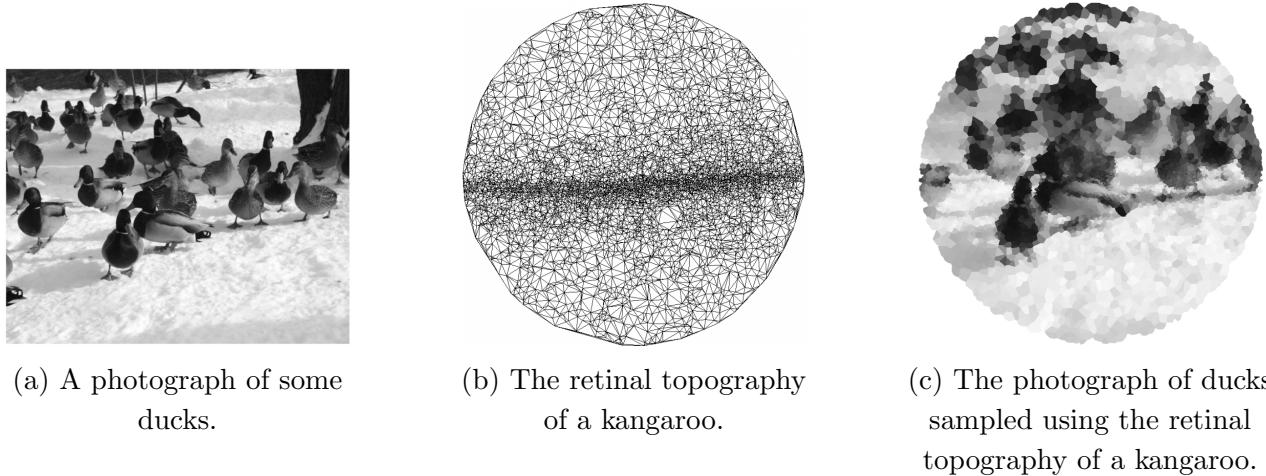


Figure 3.12: An example of foveal sampling. Images taken from [23].

system. To do this, the project investigates the application of *parallel* and *concurrent* computing.

Parallel computing is often conflated with *concurrent computing*. However, the terms are distinct. Parallel computing means a task is broken down into numerous similar sub-tasks that can be completed independently [48]. Concurrent computing means each sub-task will address unrelated processes, often requiring inter-task communication [48]. In Figure 3.13, the abstract layers of the networking processes have been coloured to indicate whether concurrent or parallel computing is used.

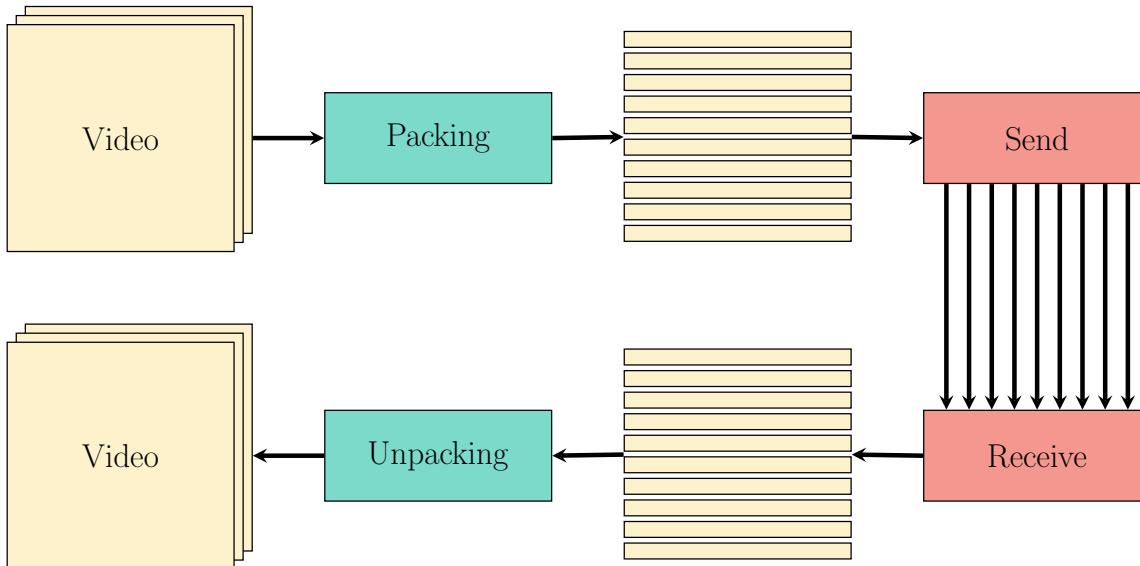


Figure 3.13: The stages required for a video to be sent across the network.

Parallel processes:

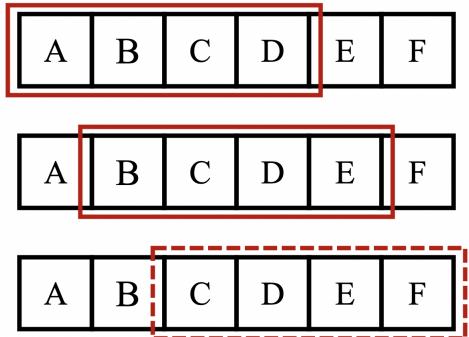
Concurrent processes:

These techniques were selected for investigation because of the growing trend of support in computer architecture. Traditionally, computer design has focussed on *sequential computation* to improve performance. However, factors such as Dennard scaling [59] mean the improvements predicted by Moore's law [51] may not continue indefinitely. Therefore,

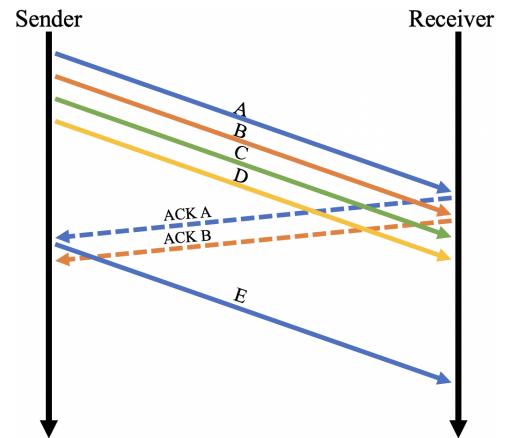
architects are utilising multiprocessing to gain similar gains. Consequently, this seemed like a viable opportunity for the investigation to consider future iterations of surveillance technology.

Communication

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 3.14 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 acknowledgements. Once an acknowledgment has been received, the next packet in the queue can be sent.

Figure 3.14: 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, a thread is assigned and establishes a connection with the server. Consequently, multiple connections will be open in parallel, allowing more data to 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. 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 make 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 transmitted, it must be prepared, or *packed*. Equivalently, when data arrives at its destination, data must be *unpacked*.

Depicted by Figure 3.15, there are three distinct stages of 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 unsurprisingly missing from these pipelines.

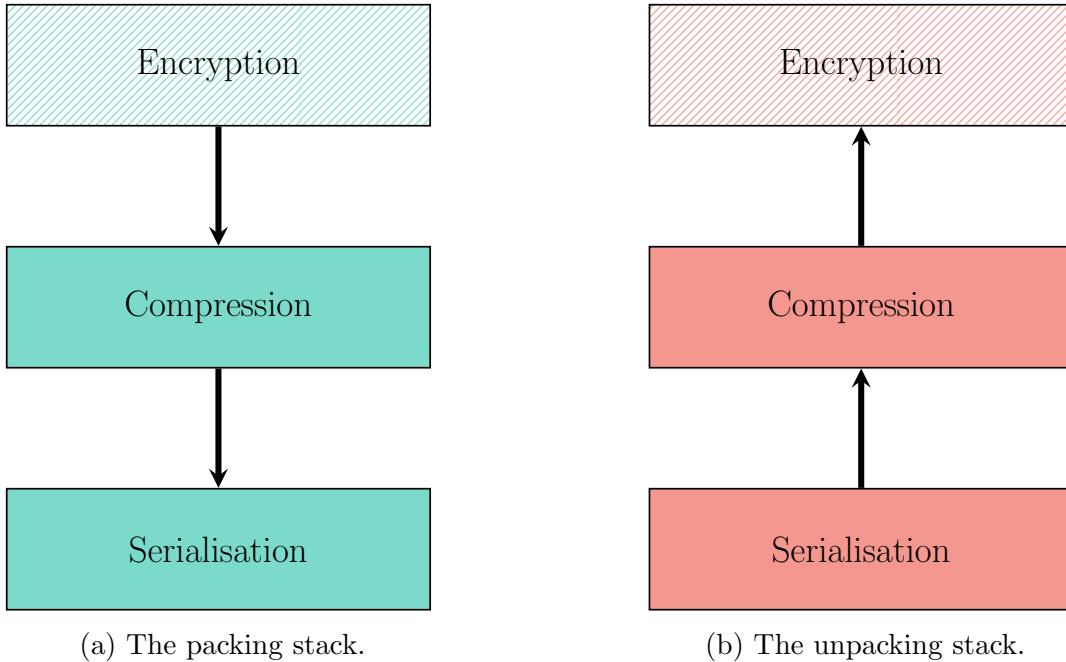


Figure 3.15: The packing and unpacking stacks. Block colours indicate processes occur in both client and server, patterned colours indicate the process only occurs in the client.

In a naïve implementation, each video pixel might be packed individually. However, this can be improved. The CKKS scheme operates on vectors of real values. Therefore, decomposing a frame into rows provides the opportunity for *vectorising* the application by encrypting each row as a single ciphertext object. Consequently, the number of ciphertexts needed is reduced - for an $n \times m$ pixel frame, the number of objects is reduced from nm to n - so the time and space complexity of encryption is reduced from quadratic to linear complexity³.

Similarly, compressing and serialising subsections of video can be parallelised to improve time and space complexity. Therefore, once encryption has completed, these operations can be performed by the same thread. No further overhead of thread creation is required, so this improvement is only limited by the need for each stage to terminate entirely before the next begins. Consequently, the smaller the quantum of data they operate on, the quicker they will complete. However, smaller quanta requires more threads, so a similar balance to that described in §3.3.2 must be achieved.

³This would fall under concurrency rather than parallelism because objects such as encoders, encryptors, and keys must be shared between processes.

3.4 Inference

This section discusses the implementation of the moving object detection algorithms. It will examine the necessary modifications to support HE video data in §3.4.1, and detail the more complex algorithms needed for unsupervised learning in §3.4.2 and §3.4.3. Adaptations were required to incorporate the HE Boolean circuits and overcome operation depth limitations.

3.4.1 Homomorphic Encryption Adaptations

There were two main challenges to overcome when implementing inference algorithms in 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 HE is more limited than is available when working with plain data. Consequently, compromises were 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 increases the memory usage of ciphertexts, so data transmission quickly becomes infeasible.

The adaptations implemented for the less complex algorithms are detailed below. A discussion of the techniques investigated for implementing GMMs is evaluated in §4.2.

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 impact the level of a ciphertext, so has no impact on ciphertext size. As well as making networking more efficient, this also makes the inference algorithm faster because the data being operated over is smaller.

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 most straightforward implementation recalculates the reference frame repeatedly by storing a list of all frames that have been observed and finding their mean each time a new frame is received. In contrast, a more advanced implementation will use an iterative function to update the mean, so does not require storing all frames.

Evidently, the second method will perform better in both time and space complexity when considering plain video data. However, when investigating HE data, the distinction is not as clear. 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 because increasing ciphertext size so far would severely increase the time complexity of operations and make transmission times impractically slow. 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 cost of recalculating the mean of plaintext data may be relatively negligible, as more frames are observed, HE implementations will become considerably slower. However, this method can be used to derive a solution. A compromise can be achieved by limiting the number of frames stored, forgetting the oldest frame whenever a new one arrives. Importantly, this may reduce inference accuracy, so a balance must be struck between running time and inference quality.

Gaussian Average

Similarly to a mean filter, a plaintext implementation of Gaussian average inference would fit the Gaussian distributions using iterative formulae for the mean and variance. Copied from §2.1.3 for convenience, Equation 3.12 and Equation 3.13 provide these formulae. Consequently, the plaintext implementation will be infeasible in the HE domain. Fortunately, it can be adapted.

$$\mu_t = \begin{cases} f_0 & \text{if } t = 0 \\ \alpha f_t + (1 - \alpha) \mu_{t-1} & \text{otherwise} \end{cases} \quad (3.12)$$

$$\sigma_t^2 = \begin{cases} c & \text{if } t = 0 \\ d^2 \alpha + (1 - \alpha) \sigma_{t-1}^2 & \text{otherwise} \end{cases} \quad (3.13)$$

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.

Firstly, it can be observed that expanding the iterative formulae highlights that the frames of a video are *multiplicatively independent*, as demonstrated by Equation 3.14 for the fifth frame of the video.

$$\begin{aligned} \mu_4 &= \alpha f_4 + \alpha(1 - \alpha) f_3 + \alpha(1 - \alpha)^2 f_2 + \alpha(1 - \alpha)^3 f_1 + (1 - \alpha)^4 f_0 \\ \sigma_4^2 &= \alpha d_4^2 + \alpha(1 - \alpha) d_3^2 + \alpha(1 - \alpha)^2 d_2^2 + \alpha(1 - \alpha)^3 d_1^2 + (1 - \alpha)^4 c \end{aligned} \quad (3.14)$$

Helpfully, the coefficient terms across calculations are identical, so computation can be shared. More importantly, the server decides the value of α before runtime. Therefore, 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. Then, results can be summed to give the mean and variance. Consequently, the number of levels required for a ciphertext is significantly reduced, so time and space complexity are also reduced.

However, this method still requires storing a list of preceding frames to recalculate terms repeatedly. Like with mean filter, this introduces a trade-off between running time and memory usage against inference accuracy.

3.4.2 Online Mixture Model

In 1999 Stauffer and Grimson proposed *adaptive background mixture models* for real-time moving object detection [55]. To overcome a single Gaussian distribution's inability to cope with the changing lighting conditions in practice, they proposed a mixture of adaptive

Gaussians. The advantage of this technique over other GMMs is that the model runs *online* because 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 3.15.

$$\{X_0, \dots, X_t\} = \{I(x, y, i) \mid 0 \leq i \leq t\} \quad (3.15)$$

where I is the image sequence.

The 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

$$\mathbb{P}(X_t) = \sum_{i=1}^K \omega_{i,t} \times \eta(X_t, \mu_{i,t}, \Sigma_{i,t}) \quad (3.16)$$

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 3.17.

$$\eta(X, \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(X-\mu)^T \Sigma^{-1} (X-\mu)} \quad (3.17)$$

To maximise the likelihood of the observed data, a *k-means* approximation was selected 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.

If none of the distributions match a pixel's value, the least likely Gaussian is replaced by a new distribution defined with the current value as its mean, an initially high variance, and low prior weight. Then, the prior weights are adjusted at time t using Equation 3.18.

$$\omega_{k,t} = (1 - \alpha)\omega_{k,t-1} + \alpha M_{k,t} \quad (3.18)$$

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 3.19, where ρ is defined by Equation 3.20.

$$\mu_t = (1 - \rho)\mu_{t-1} + \rho X_t \quad (3.19a)$$

$$\Sigma_t^2 = (1 - \rho)\Sigma_{t-1}^2 + \rho(X_t - \mu_t)^T(X_t - \mu_t) \quad (3.19b)$$

$$\rho = \alpha\eta(X_t, \mu_k, \Sigma_k) \quad (3.20)$$

Predicting

Once the parameters have been updated, each pixel must be labelled by determining the Gaussian component most likely produced by the background process. This decision assumes that there will be relatively little variance in the Gaussians when a static object is in the frame and a large variance not matching existing distributions when a new object occludes the background. Consequently, a method 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 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

$$B = \operatorname{argmin}_b \left(\sum_{k=1}^b \omega_k > T \right) \quad (3.21)$$

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* of a statistical model [16]. 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 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: 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 3.22.

$$\gamma_{Z_i=k} = \mathbb{P}(Z_i = k | X_i) = \frac{\mathbb{P}(X_i | Z_i = k)\mathbb{P}(Z_i = k)}{\mathbb{P}(X_i)} \quad (3.22)$$

$$= \frac{\omega_k \mathcal{N}(x_i, \mu_i, \sigma_i)}{\sum_c \omega_c \mathcal{N}(x_c, \mu_c, \sigma_c)} \quad (3.23)$$

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 3.24 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.

$$Q(\theta^{(t)}, \theta^{(t-1)}) = \mathbb{E} [\log \mathbb{P}(Z | \theta^{(t)}) | X, \theta^{(t-1)}] \quad (3.24)$$

$$= \sum_{k=1}^M \log \mathbb{L}(\theta_k | X, Z) \mathbb{P}(Z_k | X, \theta^{(t-1)}) \quad (3.25)$$

$$= \sum_{k=1}^M \log \mathbb{L}(\theta_k | X, Z) \gamma_{Z_i=k} \quad (3.26)$$

where $\log \mathbb{L}(\theta_k | X, Z)$ 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 3.27.

$$\theta^{(t+1)} = \operatorname{argmax}_{\theta} Q(\theta^{(t)}, \theta^{(t-1)}) \quad (3.27)$$

where

$$Q(\theta^{(t)}, \theta^{(t+1)}) = \sum_{k=1}^M \sum_{i=1}^N \log \gamma_k \mathbb{P}(Z_k | X_i, \theta^{(t-1)}) + \sum_{k=1}^M \sum_{i=1}^N \log \mathbb{P}(x_i | \theta_k) \mathbb{P}(Z_k | X_i, \theta^{(t-1)}) \quad (3.28)$$

From this, the optimal parameter values can be derived by differentiating Equation 3.28 with respect to the means, covariances, and weights, and solving when equal to zero, in turn. The results of these calculations are given Equation 3.29, Equation 3.30, and Equation 3.31, respectively. In the equations, $N_k = \sum_{i=1}^N \gamma_{Z_i=k}$.

$$\hat{\mu}_k = \frac{\sum_{i=1}^N X_i \mathbb{P}(Z_i = k | X_i, \theta^{(t-1)})}{\sum_{i=1}^N \mathbb{P}(Z_i = k | X_i, \theta^{(t-1)})} = \frac{1}{N_k} \sum_{i=1}^N \gamma_{Z_i=k} X_i \quad (3.29)$$

$$\hat{\sigma}^2_k = \frac{1}{N_k} \sum_{i=1}^N \gamma_{Z_i=k} (X_i - \mu_k)^2 \quad (3.30)$$

$$\hat{\omega}_k = \frac{N_k}{N} \quad (3.31)$$

3.5 Summary

Overall, the project’s implementation stage began with a foundational implementation of a client-server stack, followed by the integration of Microsoft’s SEAL library to allow HE data transmission, satisfying the first core success criterion. After this, an investigation into the optimisation of the network stack involving the implementation of a *seam carving* algorithm and *graph representations* of images to reduce the size of videos and *parallelisation* to increase the transmission rate between client and server.

The following implementation stage required inference algorithms to be converted to the HE domain to satisfy the second core success criterion. As part of this, several modifications had to be made to the algorithms, and investigations had to be conducted into implementing more HE Boolean circuits.

Finally, a bespoke HE scheme was implemented from first principles following the CKKS scheme initially integrated. This allowed further understanding and an investigation into specialising the implementation as an opportunity for optimisation.

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 performance on the LASIESTA dataset is included in Appendix C.

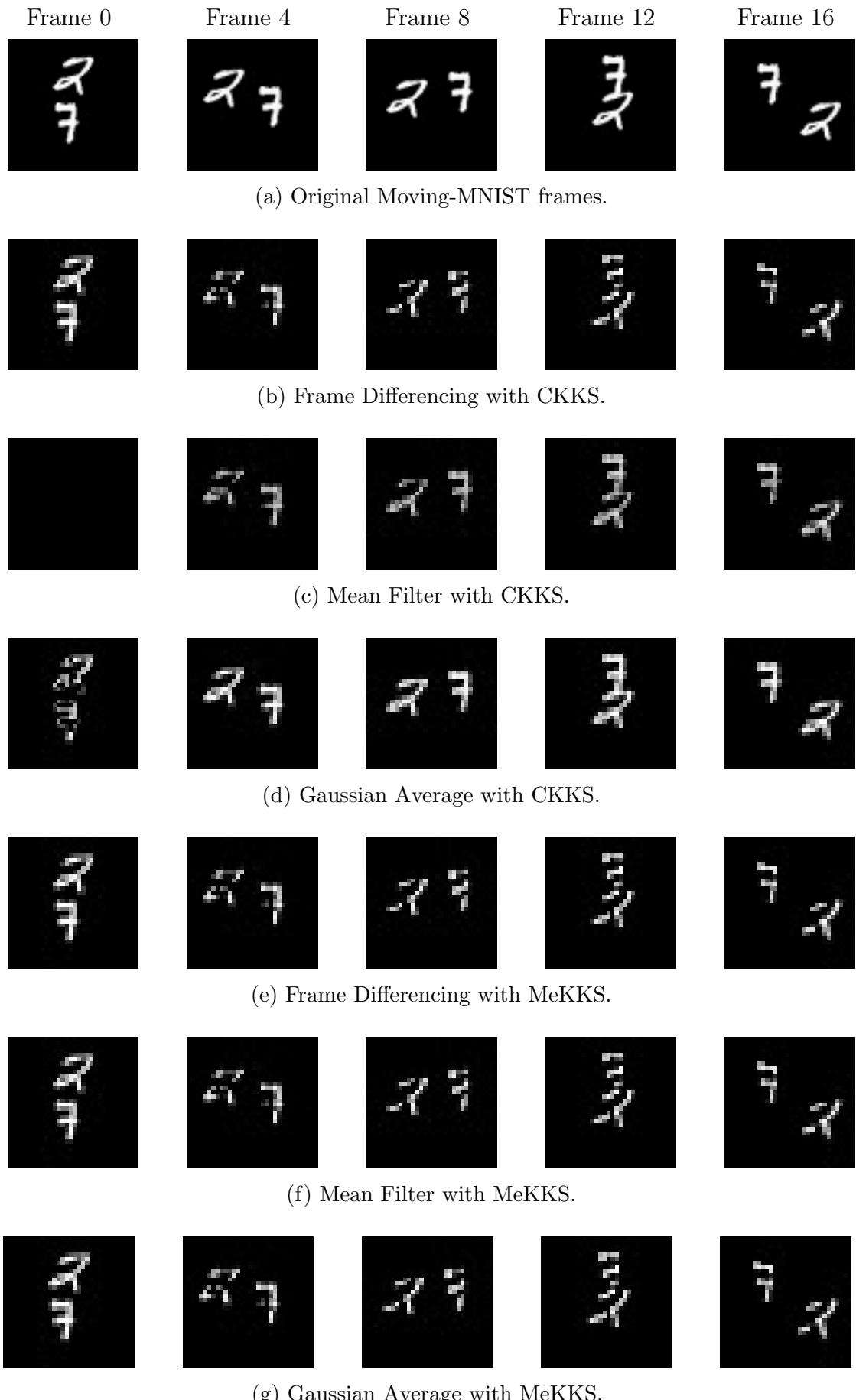


Figure 4.1: Moving-MNIST Inference Results

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 3.21 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 [13]. Unfortunately, this introduces security concerns. If the ability to compare two HE ciphertexts is added to the system, an attacker¹ 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.

¹such as Mallory in §2.1.1.

Algorithm Comp($a, b; d, d', t, m$)

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

```

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

```

Figure 4.2: Homomorphic Comparison Algorithm

Algorithm 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:    $b_{n+1} \leftarrow b_n^2$ 
5:    $a_{n+1} \leftarrow a_n \cdot (1 + b_{n+1})$ 
6: end for
7: return  $a_d$ 

```

Figure 4.3: Homomorphic Division Algorithm

4.2.2 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. [13]. 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

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 naïve 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 when the primary camera should be triggered to conserve battery life. Then, only short clips containing potential events are transferred to the server for inference. Consequently, real-time performance is not required. Although, speed cannot be ignored to ensure users are not notified too late.

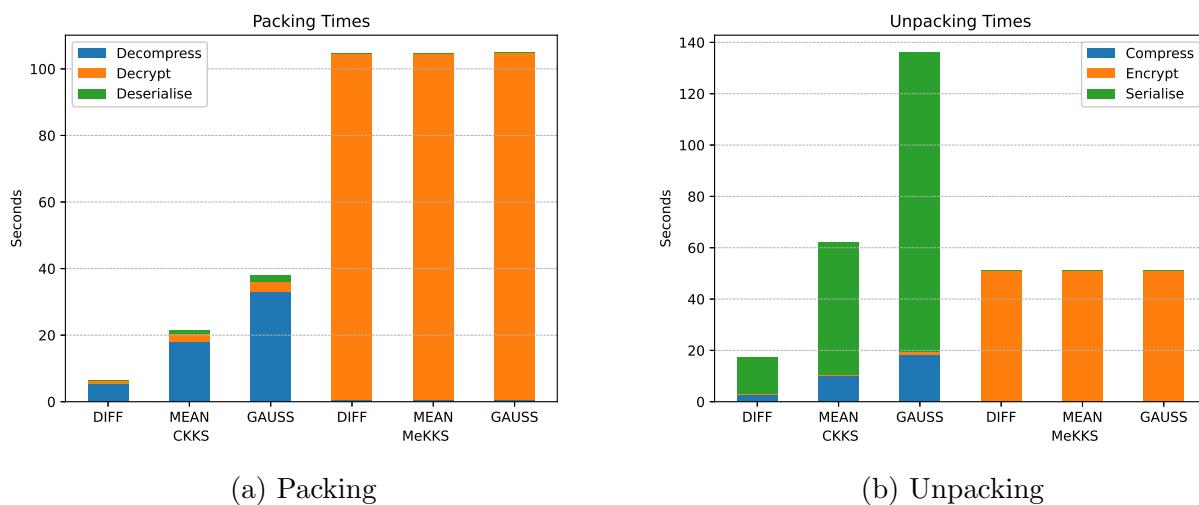
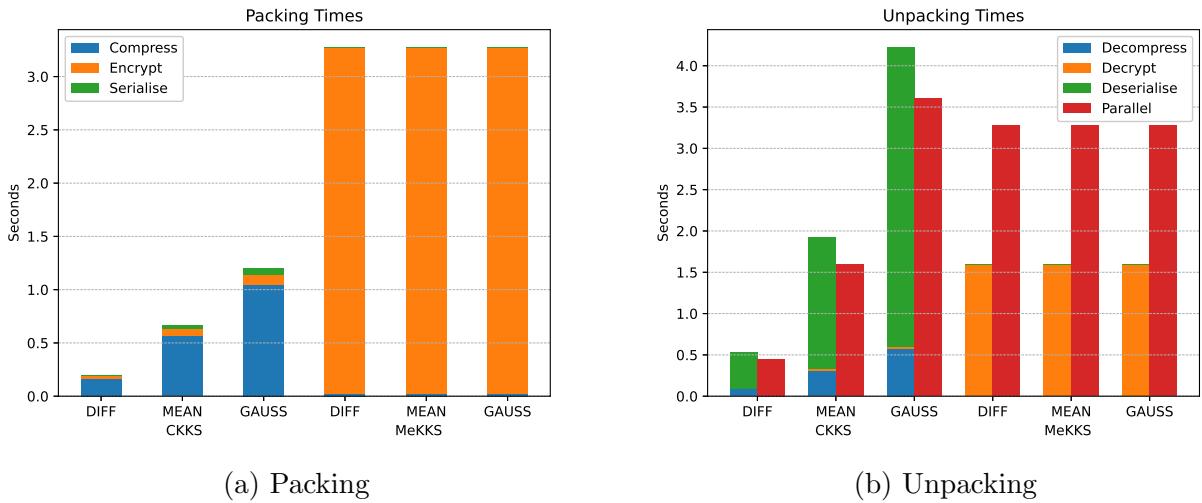


Figure 4.4: Naïve Packing and Unpacking Times



(a) Packing

(b) Unpacking

Figure 4.5: Optimised Packing and Unpacking Times

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.

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 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

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.

compressing videos is shown in Figure 4.6. MeKKS’s more basic implementation means the ciphertext size is constant for all inference methods. Despite this, it is still two orders of magnitude smaller than CKKS data, thanks to the native Python data structures used. Compression is much more significant with CKKS data, almost halving the memory usage for the Gaussian inference, making it a valuable component of the CKKS packing procedure, but largely unnecessary when MeKKS is selected.

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.

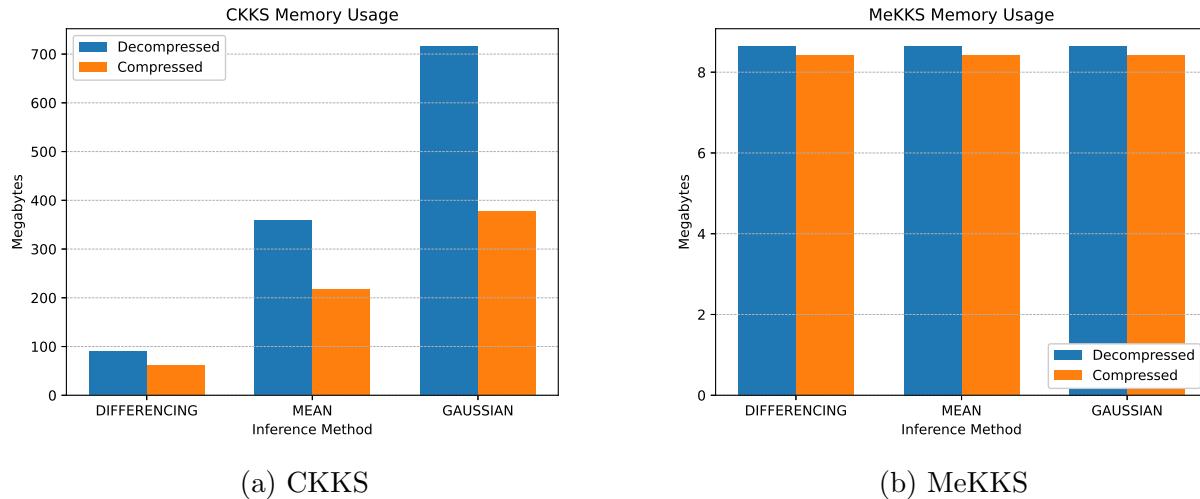


Figure 4.6: Naïve Memory Usage Compression Comparison

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

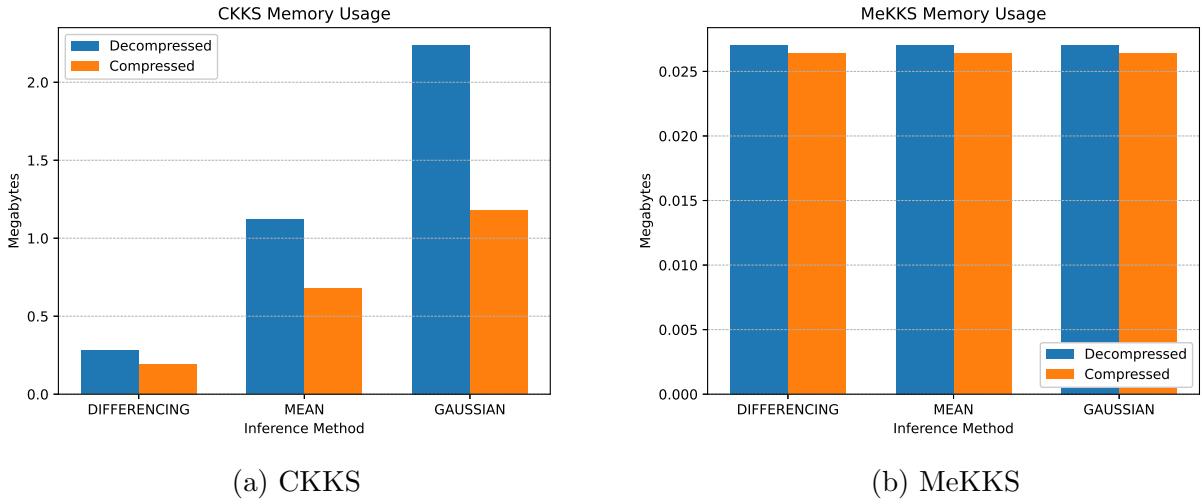


Figure 4.7: Vectorised Memory Usage Compression Comparison

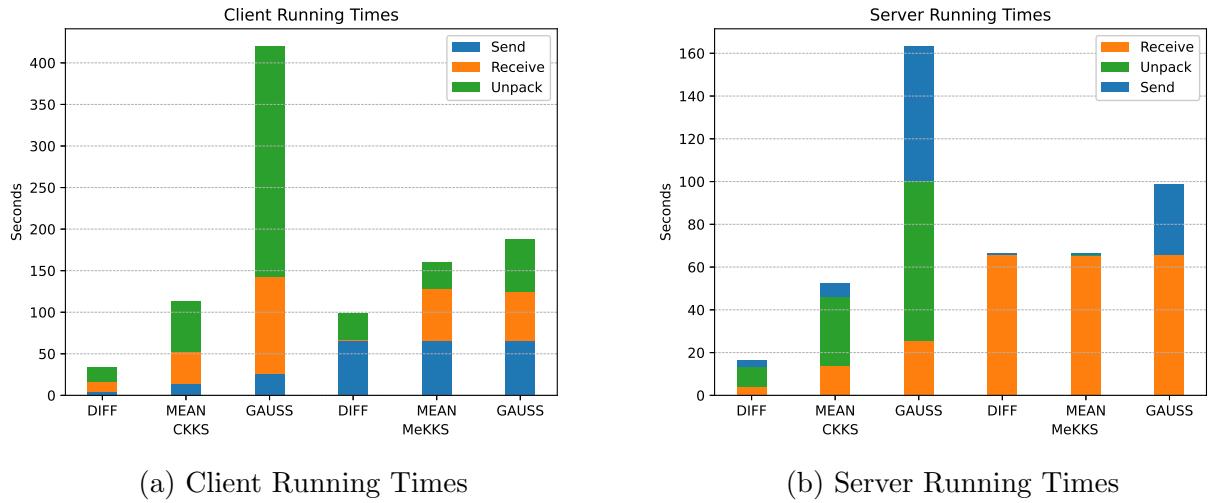


Figure 4.8: Client and Server Running Times

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. While the HE implementations do not exactly match inference in unencrypted data, they are able to produce nearly the same accuracy, producing almost identical results to the human eye. Perhaps surprisingly, MeKKS produces slightly more accurate results with frame differencing, but this is likely due to the random noise induced by HE encryption.

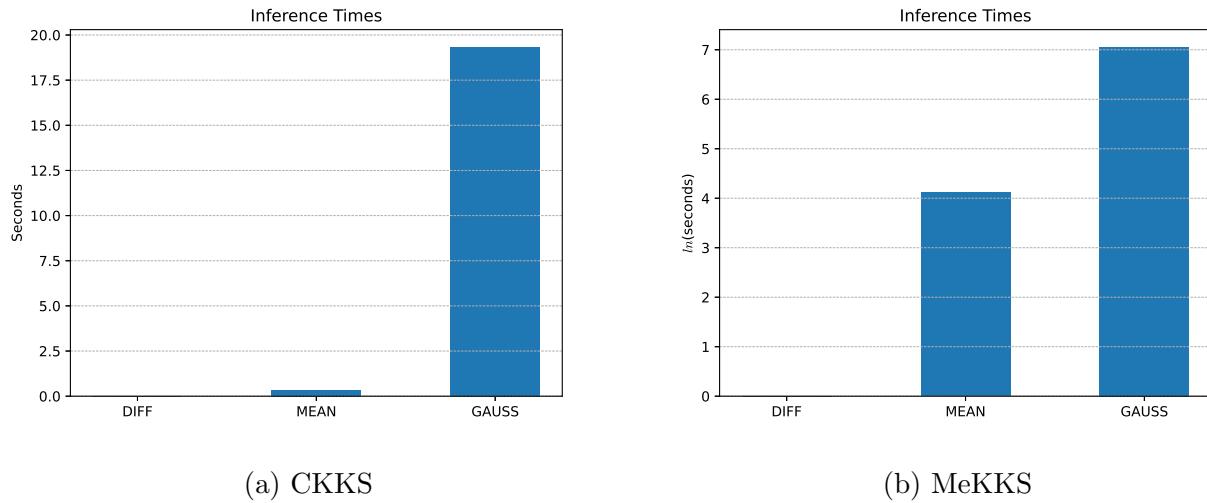


Figure 4.9: Inference Times

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 square difference* according to Equation 4.1

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

which can be normalised using (4.1)

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

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

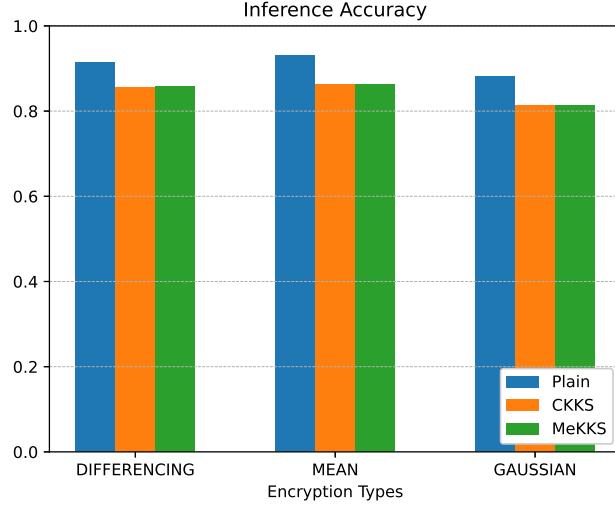


Figure 4.10: Inference Accuracy

Energy Usage

While not a particular focus of this investigation, energy usage is an important factor in determining the practicality of HE in surveillance. This is because the majority of devices being emulated - surveillance cameras or doorbells - are battery-powered. As such, they must be conservative in their energy usage in order to extend battery life as far as possible. Figure 4.11 depicts the energy usage of each combination of encryption scheme and inference method for the client and server modules. The absolute value cannot be isolated from background energy usage, so values have been normalised to allow for comparison on a log scale.

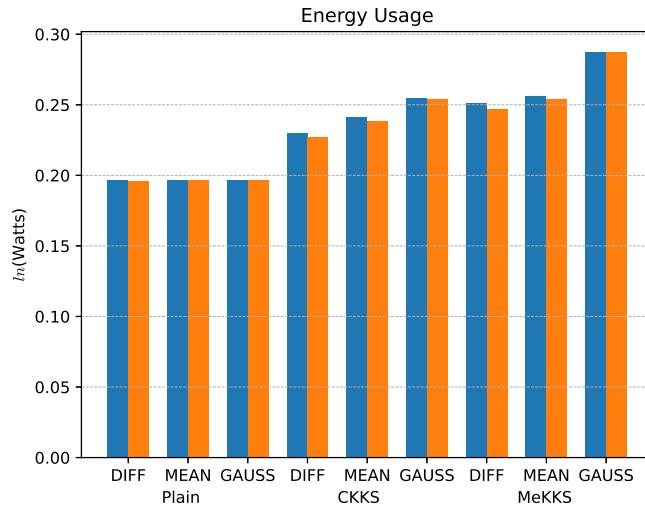


Figure 4.11: Energy Usage

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. 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

5.1 Project Summary

Overall, the project was a success. All core success criteria and one of the extensions were met, and the remaining criteria were thoroughly considered. In its final state, the project contains an investigation into the applicability of HE in ensuring user privacy is preserved when using modern MLaaS surveillance technology. For this, a client-server system allowing data to be homomorphically encrypted using the CKKS scheme and transferred across a network was implemented. Moreover, several background subtraction algorithms were implemented, and more were investigated to attempt to find the limit of HE pertinency. Also, the results of these algorithms were evaluated to compare to traditional algorithms that operate on plaintext data. Furthermore, a bespoke implementation of the CKKS scheme - called MeKKS - was created to increase understanding and highlight opportunities for application-specific optimisations.

5.2 Lessons Learned

Throughout the project, many challenges were faced. The most critical of these challenges was the requirement for research to cultivate the background knowledge of both unsupervised machine learning for image segmentation and HE - neither of which were covered in Tripos content. Specifically concerning HE, lack of documentation regarding scheme applications warranted many hours of literature reviews and deciphering research papers.

Furthermore, computations using machine learning and HE took a long time to execute - particularly when networking was involved. Consequently, debugging and evolving algorithms was a time-consuming process. This made strategic time management even more critical to allow enough time to complete a thorough investigation. However, these challenges did make the project a productive learning experience that will be helpful when exploring other areas of computer science in future or when working on any large projects generally.

5.3 Future Directions

The project encountered two recurring limitations that would benefit from further investigation. Firstly, the memory requirements enforced by HE result in a significant bottleneck during the networking stage of the application. Consequently, the practicality of HE for overcoming privacy concerns in MLaaS models is severely limited. To overcome this, either the

number of levels in a ciphertext needed to perform consecutive multiplications will need to be reduced, or more realistically, the impact of adding a new level on the size of a ciphertext must be made more affordable.

Secondly, the variety of operations that can be performed on HE ciphertexts must be expanded if more advanced moving object detection algorithms are to be implemented. Even relatively simple operations such as division are lacking from the CKKS scheme, which means either new inference algorithms will need to be developed or the HE schemes will need to be extended to be more supportive.

Bibliography

- [1] Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody, Travis Morrison, Amit Sahai, and Vinod Vaikuntanathan. Homomorphic encryption standard. *Protecting Privacy through Homomorphic Encryption*, 2022.
- [2] Paul Viola and Michael Jeffrey Jones. Rapid object detection using a boosted cascade of simple features. *Conference on Computer Vision and Pattern Recognition*, 2001.
- [3] Shai Avidan and Ariel Shamir. Seam carving for content-aware image resizing. *ACM Transactions on Graphics*, 2007.
- [4] 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, the first homomorphic cnn on encrypted data with gpus. *IEEE Transactions on Emerging Topics in Computing*, 2020.
- [5] Ho Bae, Jaehee Jang, Dahuin Jung, Hyemi Jang, Heonseok Ha, Hyungyu Lee, and Sungroh Yoon. Security and privacy issues in deep learning. *Journal of IEEE Transactions on Artificial Intelligence*, 2018.
- [6] Serge Beucher and Christian Lantuéjoul. Use of watersheds in contour detection. *International workshop on image processing, real-time edge and motion detection*, 1979.
- [7] M. Bhattacharya, R. Creutzburg, and J. Astola. Some historical notes on number theoretic transform. *International TICSP Workshop on Spectral Methods and Multirate Signal Processing*, 2004.
- [8] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. *Annual Cryptology Conference*, 2012.
- [9] Brilliant.org. Homomorphic encryption. <https://brilliant.org/wiki/homomorphic-encryption/>, Retrieved March 2022.
- [10] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. Bootstrapping for approximate homomorphic encryption. *Advances in Cryptology*, 2018.
- [11] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. A full rns variant of approximate homomorphic encryption. *International Conference on Selected Areas in Cryptography*, 2019.

- [12] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. *Lecture Notes in Computer Science*, 2016.
- [13] Jung Hee Cheon, Dongwoo Kim, Duhyeong Kim, Hun hee Lee, and Keewoo Lee. Numerical method for comparison on homomorphically encrypted numbers. *International Conference on the Theory and Application of Cryptology and Information Security*, 2019.
- [14] Kuan-Yu Chu, Yin-Hsi Kuo, and Winston H. Hsu. Real-time privacy-preserving moving object detection in the cloud. *Proceedings of the 21st ACM international conference on Multimedia*, 2013.
- [15] 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.
- [16] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society*, 1977.
- [17] 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. *ICML*, 2016.
- [18] Ronald Duarte and Resit Sendag. Accelerating and characterizing seam carving using a heterogeneous cpu-gpu system. *Proceedings of the 18th Annual International Conference on Parallel and Distributed Processing Techniques and Application*, 2012.
- [19] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *Cryptology ePrint Archive*, 2012.
- [20] Python Software Foundation. Psf license agreement for python 3.10.4. <https://docs.python.org/3/license.html#psf-license>, Retrieved March 2022.
- [21] Python Software Foundation. Python. <https://www.python.org/>, Retrieved March 2022.
- [22] Git. Git. <https://git-scm.com/>, Retrieved March 2022.
- [23] Leo John Grady. *Space-variant computer vision: A graph-theoretic approach*. PhD thesis, Boston University, 2004.
- [24] Thore Graepel, Kristin Lauter, and Michael Naehrig. MI confidential: Machine learning on encrypted data. *The International Conference on Information Security and Cryptology*, 2012.
- [25] Dazhou Guo, Yanting Pei, Kang Zheng, Hongkai Yu, Yuhang Lu, and Song Wang. Degraded image semantic segmentation with dense-gram networks. *IEEE Transactions on Geoscience and Remote Sensing*, 2021.
- [26] Huelse. Seal-python. <https://github.com/Huelse/SEAL-Python>, 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. *Journal of Heuristics*, 2004.
- [31] Chiraag Juvekar, inod Vaikuntanathan, and Anantha Chandrakasan. Gazelle: A low latency framework for secure neural network inference. *Proceedings of the 27th USENIX Conference on Security Symposium*, 2018.
- [32] Intan Kartika and Shahrizat Shaik Mohamed. Frame differencing with post-processing techniques for moving object detection in outdoor environment. *IEEE 7th International Colloquium on Signal Processing and its Applications*, 2011.
- [33] Andrey Kim, Antonis Papadimitriou, and Yuriy Polyakov. Approximate homomorphic encryption with reduced approximation error. *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. *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. *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. *2015 International Conference on Pervasive Computing (ICPC)*, 2015.
- [37] Kim Laine. Simple encrypted arithmetic library 2.3.1. *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. *Multimedia Tools and Applications*, 2017.
- [40] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. *Journal of the ACM*, 2010.
- [41] J. MacQueen. Some methods for classification and analysis of multivariate observations. *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. *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] Payal V. Parmar, Shraddha B. Padhar, Shafika N. Patel, Niyatee I. Bhatt, and Rutvij H. Jhaveri. Survey of various homomorphic encryption algorithms and schemes. *International Journal of Computer Applications*, 2014.
- [46] Mike Paterson and Larry J. Stockmeyer. On the number of nonscalar multiplications necessary to evaluate polynomials. *SIAM Journal on Computing*, 1973.
- [47] Manas A. Pathak and Bhiksha Raj. Privacy-preserving speaker verification and identification using gaussian mixture models. *IEEE*, 2013.
- [48] Rob Pike. Concurrency is not parallelism. Waza Conference (Slides: <https://talks.golang.org/2012/waza.slide#1>), 2012.
- [49] W. W. Royce. Managing the development of large software systems: concepts and techniques. *Proceedings of the 9th International Conference on Software Engineering*, 1987.
- [50] Michael Rubinstein, Ariel Shamir, and Shai Avidan. Improved seam carving for video retargeting. *The ACM SIGGRAPH conference proceedings*, 2008.
- [51] R.R. Schaller. Moore's law: past, present and future. *IEEE Spectrum*, 1997.
- [52] Linda G. Shapiro and George C. Stockman. *Computer Vision*. Pearson, 2001.
- [53] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv:1312.6034*, 2014.
- [54] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhutdinov. Unsupervised learning of video representations using lstms. https://www.cs.toronto.edu/~nitish/unsupervised_video/, Retrieved March 2022.
- [55] Chris Stauffer and W. E. L. Grimson. Adaptive background mixture models for real-time tracking. *1999 IEEE computer society conference on computer vision and pattern recognition*, 1999.
- [56] Steven Turner. Transport layer security. *IEEE Internet Security*, 2014.
- [57] Ioannis Tzemos, Apostolos P. Fournaris, and Nicolas Sklavos. Security and efficiency analysis of one time password techniques. *20th Pan-Hellenic Conference on Informatics*, 2016.
- [58] Wikipedia user Newton2. Seam carving. https://en.wikipedia.org/wiki/Seam_carving, Retreived March 2020.
- [59] Greg Yeric. Moore's law at 50: Are we planning for retirement? *International Electron Devices Meeting*, 2015.

- [60] Jingru Yi, Pengxiang Wu, Menglin Jiang, Qiaoying Huang, Daniel J. Hoeppner, and Dimitris N. Metaxas. Attentive neural cell instance segmentation. *Medical Image Analysis*, 2019.
- [61] Ruolin Zhang and Jian Ding. Object tracking and detecting based on adaptive background subtraction. *Procedia Engineering*, 2012.

Appendix A

Homomorphic Encryption Addendum

A.1 Group Theory

The RLWE problem used by the CKKS encryption scheme 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, $\bullet : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}$, 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, $\boxplus : \mathbf{R} \times \mathbf{R} \rightarrow \mathbf{R}$, and $\boxtimes : \mathbf{R} \times \mathbf{R} \rightarrow \mathbf{R}$, 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 $(\mathbb{Z}[x], +, \times)$.

A.2 CKKS Operations

Figure A.1 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.

- **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 $\mathbf{a} := -\mathbf{rs} + \mathbf{e} \pmod{qL}$ and $\mathbf{a}' := -\mathbf{r}'\mathbf{s} + \mathbf{e}' + Ps^2 \pmod{P \cdot qL}$.
 2. Return $\text{SecretKey} := (1, \mathbf{s})$, $\text{PublicKey} := (\mathbf{a}, \mathbf{r})$, $\text{EvaluationKey} := (\mathbf{a}', \mathbf{r}')$.
- **Enc_{PublicKey}(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 $(\mathbf{c}_0, \mathbf{c}_1) \pmod{qL}$.
- **Dec_{SecretKey}(c₀ + c₁ · s)** = $(\mathbf{c}_0 + \mathbf{c}_1) \pmod{q_l}$.
- **Add((c₀, c₁), (d₀, d₁))** = $(\mathbf{c}_0 + \mathbf{d}_0, \mathbf{c}_1 + \mathbf{d}_1) \pmod{q_l}$.
- **AddPlain((c₀, c₁), x)** = $(\mathbf{c}_0 + \text{Enc}_{\text{PublicKey}}(x), \mathbf{c}_1) \pmod{q_l}$.
- **Multiply((c₀, c₁), (d₀, d₁))** = $(\mathbf{r}_0, \mathbf{r}_1) + ([P^{-1} \cdot \mathbf{r}_2 \cdot \text{EvaluationKey}_0], [P^{-1} \cdot \mathbf{r}_2 \cdot \text{EvaluationKey}_1]) \pmod{q_l}$.
- **MultiplyPlain((c₀, c₁), x)** = $(\mathbf{c}_0 \cdot \text{Enc}_{\text{PublicKey}}(x), \mathbf{c}_1 \cdot \text{Enc}_{\text{PublicKey}}(x)) \pmod{q_l}$.

Figure A.1: A list of the basic operations supported by the CKKS scheme. P is a large scaling factor and $[\cdot]$ denotes rounding to the nearest integer. Recreated from [12].

Appendix B

Seam Carving

In the original seam carving paper, Avidan and Shamir proposed multiple energy functions to attempt to quantify the importance of a pixel in order to remove “those which won’t be noticed” [3].

To begin with, Equation B.1 is proposed. It is then evolved into Equation B.2 to achieve better results by using a histogram to group pixels.

$$e(\mathbf{I}) = \left| \frac{\partial}{\partial x} \mathbf{I} \right| + \left| \frac{\partial}{\partial y} \mathbf{I} \right| \quad (\text{B.1})$$

$$e_{HoG}(\mathbf{I}) = \frac{\left| \frac{\partial}{\partial x} \mathbf{I} \right| + \left| \frac{\partial}{\partial y} \mathbf{I} \right|}{\max(HoG(\mathbf{I}(x, y)))} \quad (\text{B.2})$$

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 B.1b depicts the application of an energy function to an example image.

Once the energy of each pixel has been calculated, the seams of the image need to be generated. This allows the optimal seam to be determined and deleted from the image. To do this, the cost of a seam can be quantified by Equation B.3. The dynamic programming algorithm originally proposed is defined by Equation B.4 - for vertical seams, the equation for horizontal seams is similar.

$$E(\mathbf{s}) = E(\mathbf{I}_s) = \sum_{i=1}^n e(\mathbf{I}(s_i)) \quad (\text{B.3})$$

$$M(i, j) = e(i, j) + \min(M(i - 1, j - 1), M(i - 1, j), M(i - 1, j + 1)) \quad (\text{B.4})$$

The figure below depicts each stage of the seam carving algorithm. Images are taken from [58].

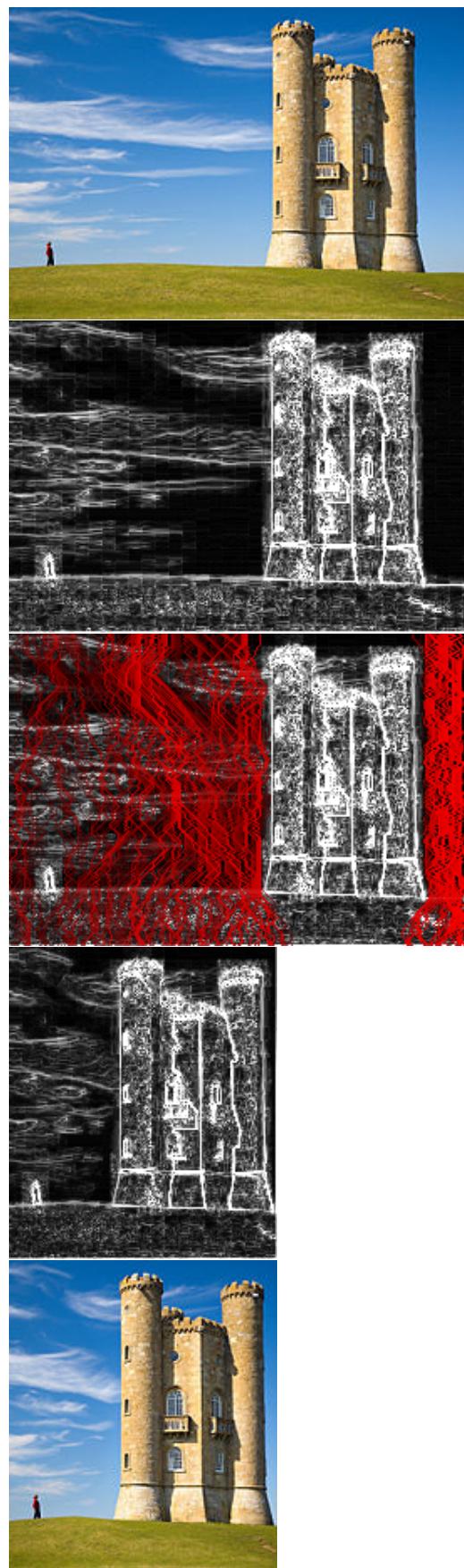
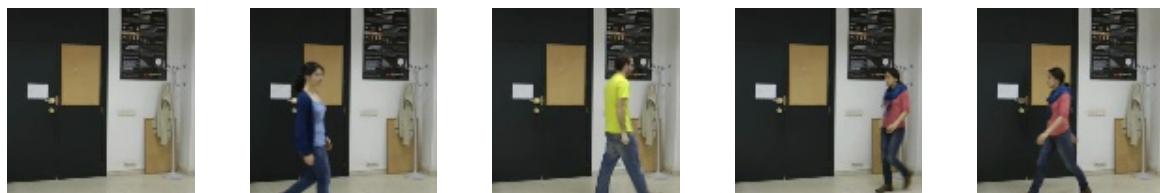


Figure B.1: Each stage of the seam carving algorithm.

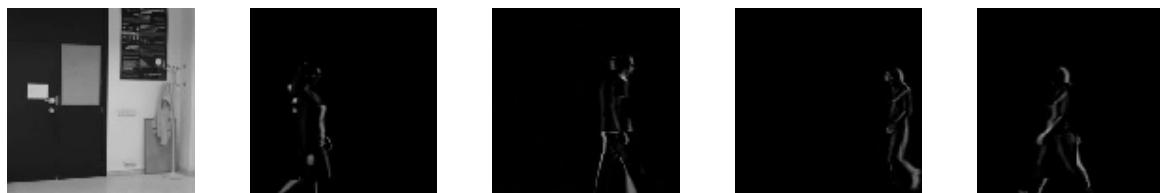
Appendix C

LASIESTA Inference Results

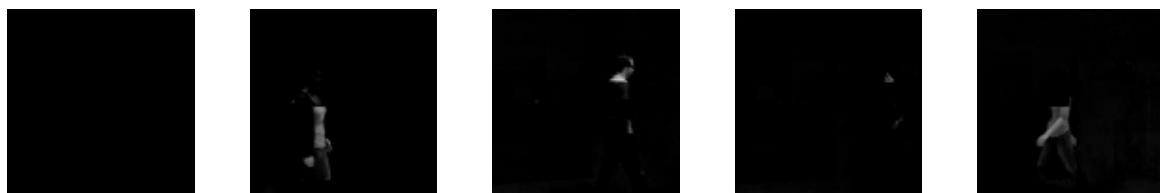
The results of each inference algorithm running on unencrypted data and data encrypted with the CKKS scheme are included in Figure C.1 on the next page.



(a) Original LASIESTA frames.



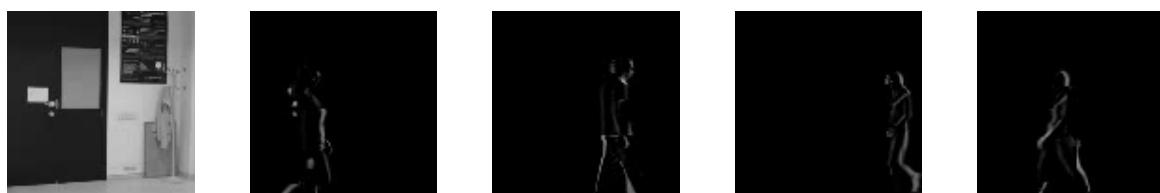
(b) Frame Differencing with no HE.



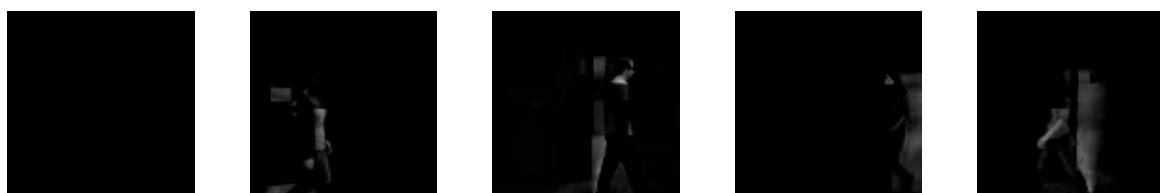
(c) Mean Filter with no HE.



(d) Gaussian Average with no HE.



(e) Frame Differencing with CKKS.



(f) Mean Filter with CKKS.



(g) Gaussian Average with CKKS.

Figure C.1: LASIESTA Inference Results

Appendix D

Project Proposal

The original proposal for the project is included from the next page.

Privacy-Preserving Moving Object Detection

Jonathon Ackers

Description

The inspiration for this project is smart home and security companies such as [Ring](#). These companies offer their customers the ability to secure their homes by selling a variety of surveillance cameras for both internal and external use. Moreover, as well as traditional surveillance, companies are now utilising the latest machine learning technologies to analyse video streams in order to interpret what is being recorded, and respond accordingly. For example, if a smart doorbell recognises who is arriving at the front door, it could automatically unlock and allow them to enter. However, while there are many benefits to this - and these systems often make peoples' lives easier, there are downsides to security cameras getting smarter.

In its current form, machine learning inference requires more hardware power than can be included in the relatively small devices. Consequently, video data must be transferred to servers where the analysis can be performed. During the transfer this data will be encrypted for security and privacy, but when it arrives on the cloud, it must then be decrypted so that the machine learning models can be applied. This means that the companies providing these services have access to raw video streams of peoples' homes. Understandably, this has become a privacy concern for many people whom don't trust these companies not to be malicious with this data. For example, service providers could use this data to monitor their customers' property and guests, then sell this to other companies. The incentive for the organisations to implement a system like this is to provide assurances to their customers that their privacy will be protected, so should encourage more people to use their products.

Therefore, the goal of my project is to create an application which can perform machine learning inference on encrypted data, rather than raw video, in order to preserve the privacy of users. The application will emulate the services provided by companies like Ring, in order to demonstrate what could be possible in real-world industry. For simplicity, and to ensure the solution is as robust as possible, I will focus on just moving-object detection. Initially, I plan to use a standard homomorphic encryption library, such as SEAL or the Paillier cryptosystem, for the encryption aspect of the project. Homomorphic encryption is a method of encryption design to permit computations to be performed on the encrypted data without first decrypting it. For example, the numbers 3 and 4 could be encrypted, then the encrypted values could be multiplied together, and decrypting the result would give 12. This will allow me to focus on building and optimising the machine learning algorithms which will perform the moving object detection. The result of this stage of the project will allow users to upload a video file, and receive a video highlighting any moving objects. Part of this phase may require training machine learning models to perform inference. Training can be very slow, so it is important that I leave enough time during my project to ensure the model has learned enough to produce sufficiently accurate results.

Once the homomorphic encryption solution has been successfully implemented, I will attempt to speed up the running time of the application. To do this, I will try to optimise the encryption algorithm, in an attempt to create a scheme which is much less computationally expensive to perform inference on. As part of this, I will draw from some published works. However, in order to increase the speed of the application, some trade-offs will have to be made. Therefore, during the final stage of the project, I will compare the faster encryption scheme to the original homomorphic scheme; allowing me to gain an understanding of them relative to each other, using metrics like running time, accuracy, memory usage, and practicalities of engineering, as well as evaluating the security of the cryptographic algorithms, possibly by simulating attacks on the system.

Timetable

Package	Start	End	Description
1	14/10/2021	27/10/2021	<i>Preparatory work.</i> E.g. experiment with hardware limitations to determine if extra resources required and if application can be entirely client-side, or if a client-server model will need to be developed.
2	28/10/2021	10/11/2021	Begin creating simple video handling application. Milestone: Complete frontend of application. <i>Introduction to Robotics Assignment 1 Deadline: 01/11/2021</i>
3	11/11/2021	24/11/2021	Implement homomorphic encryption algorithm. <i>Introduction to Robotics Assignment 2 Deadline: 22/11/2021</i>
4	25/11/2021	08/12/2021	Begin implementing inference algorithm.
5	09/12/2021	22/12/2021	Finish inference algorithm. Milestone: Full stack implemented.
6	23/12/2021	05/01/2022	<i>Buffer.</i>
7	06/01/2022	19/01/2022	Begin implementing second encryption algorithm.
8	20/01/2022	02/02/2022	Finish second encryption algorithm. Milestone: Implementation complete.
9	03/02/2022	16/02/2022	Write dissertation: Introduction and Implementation chapters, and share with supervisors. <i>Cybercrime 1 Deadline: 04/02/2022</i>
10	17/02/2022	02/03/2022	Evaluate two approaches by recording the accuracy, running time, and memory usage. <i>Cybercrime 2 Deadline: 18/02/2022</i>
11	03/03/2022	16/03/2022	<i>Buffer.</i> <i>Cybercrime 3 Deadline: 04/03/2022</i>
12	17/03/2022	30/03/2022	Write dissertation: Evaluation of approaches, and share with supervisors. Milestone: First full draft. <i>Cybercrime 4 Deadline: 18/03/2022</i>
13	31/03/2022	13/04/2022	Write dissertation: Evaluation of project and response to feedback on first draft, and share with supervisors. Milestone: Second full draft.
14	14/04/2022	27/04/2022	Write dissertation: Response to feedback on second draft, and share with supervisors. Milestone: Final draft.

Special Resources

Running machine learning on inference on video streams requires large amounts of memory. This is because the video must be loaded into memory entirely before inference can occur, rather than simply reading single frames like when images are analysed. Therefore, I will likely require more GPUs, to create a greater memory capacity, in order to meet the VRAM requirements. How much support I require will become clear once I begin implementing prototype systems.

Initially, I plan on using the [Moving MNIST](#) dataset to simplify execution of the application. However, towards the end of the project I hope to use videos more similar to real-life surveillance systems. While I should be able to synthesise this data myself, the videos could contain images of other subjects, so I may require approval from the ethics committee.

Starting Point

I have not completed any work on this project in advance of submitting this proposal. All code will be written from scratch during the time allotted in my timetable. I also have little prior knowledge about homomorphic encryption and moving object detection, so I have included time for research during the preparation stage of my timetable.

The inspiration for this project has come from several papers - which I will be using to assist me during the implementation phase of the project. I have listed all of the papers I am currently planning on using here:

- [Real-Time Privacy-Preserving Moving Object Detection in the Cloud](#) by Chu et al.
- [Moving Object Detection in the Encrypted Domain](#) by Lin et al.
- [Privacy-Preserving Watch List Screening in Video Surveillance System](#) by Sohn et al
- [Adaptive background mixture models for real-time tracking](#) by Stauffer et al.
- [Object Detection in Encryption-Based Surveillance System](#) by Zeng et al.

Success Criteria

The project will be deemed successful once the following criteria have been met.

1. The application allows users to provide a video file to the client, which can then homomorphically encrypt the file, and transfer it to the server. The server can send the encrypted data back to the frontend, where it can be decrypted and played to the user.
2. The application implements a statistical model which can detect moving objects on encrypted videos from the Moving MNIST Handwritten-Digits dataset using Gaussian Mixture Models.
3. The accuracy of the moving object detection should be evaluated: comparing inference on both unencrypted, and homomorphically encrypted video, using the same statistical modelling. Metrics such as running time and memory usage will also be compared.

Extensions

If time allows, I will consider completing the following extensions.

1. I will optimise the encryption scheme used by the application in order to speed up the running time of the application, and hopefully create a real-time solution to moving object detection. If this is successful, I will allow users to live-streamed video from sources like webcams rather than having to upload a pre-recorded video file.
2. I will provide a security analysis of the the bespoke encryption algorithm, in order to evaluate the trade-offs necessary to speed up running time. This could consist of an analysis of the mathematics used during the implementation of the algorithm, or a practical attempt to break the algorithm, such as using penetration testing.
3. I will use machine learning to perform object recognition on the encrypted video, once the moving objects have been segmented. Therefore, as a result of the inference, users will receive a video highlighting moving objects, as well as a description of what these objects are.