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: 13$^{\text{th}}$ May 2022

Proforma

Candidate Number:	**2418D**
Project Title:	**Privacy-Preserving Moving Object Detection**
Examination:	**Computer Science Tripos – Part II, 2022**
Word Count:	**12,000**[1]
Code line Count:	**6,322**
Project Originator:	Dr. Stephen Cummins and 2418D
Supervisor:	Dr. 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

All core requirements of this project were fulfilled. A client-server application was designed and implemented to emulate the machine learning as a service business model, and an investigation into optimising data transmission was performed. Moreover, an investigation and evaluation of moving-object detection inference algorithms in the homomorphic encryption domain was completed, highlighting the limitations of privacy-preserving machine learning. Furthermore, as an extension, a bespoke homomorphic encryption scheme was implemented to further understanding and investigate potential optimisations through specialisation.

[1] Counted 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

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Computers have improved almost every aspect of modern life. Recently, home security has become a target of the technology revolution. Companies like Ring [52] and Eufy [21] offer IoT devices like doorbells and cameras to allow their customers to continuously monitor their property. 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 provide security, video is encrypted before transmission to the server. However, the footage must be decrypted when inference is being performed. This poses significant privacy concerns. Decrypting footage on the server exposes the opportunity for employees of these companies to access video content. Consequently, malicious actors could use this information to monitor peoples' location, appraise their belongings, extort subjects, and more. Similar risks exist if the company is hacked and raw video data is exfiltrated. 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 the calculation 3×5. In a traditional encryption scheme, the plain values 3 and 5 would be multiplied and then encrypted. Using a homomorphic scheme, 3 and 5 can be encrypted, and the ciphertexts multiplied, resulting in a ciphertext that produces 15 when decrypted. However, HE is a developing research area, so has limitations. HE ciphertexts are much larger than unencrypted data, so operations' time and space complexity significantly increased. Similarly, not all operations are available in the HE domain, and those that are varies between schemes, so the choice of scheme is critical to success. An open question is, can this technique be scaled to more complex algorithms, like those required for surveillance?

More specifically, this project aims to investigate if it is possible to extract moving objects from 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 from 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 to improve video inference privacy using malleable encryption, but none are without flaws. In 2013, Chu et al. [15] 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 attacks[1]. Similarly, in 2017, Lin et al. [42] proposed an encryption scheme to achieve the same goal by only encrypting some of the bits in each pixel, but this is unprotected against steganographic[2] attacks. Therefore, while research has solved the weaknesses in privacy, it is yet to offer a solution that also preserves security, removing utility to real-world applications.

Likewise, researchers have been investigating inference using HE for many years. In 2012, Graepel et al. [27] introduced machine learning in the HE domain. Dowlin et al. [19] 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 [51] proposed a HE implementation of a GMM for audio inference. While this work can be used to establish a foundation for GMM implementation in the HE domain, audio and video analysis details differ, so its relevance is bounded. 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 the limited applicability of HE to real-time applications due to high computational complexity. However, a consequence of this is that the usefulness of HE in video inference is not well documented. Moreover, as computing capability and hardware acceleration advance, the relative difficulty of HE operations will reduce. Therefore, more insight into the applicability will become increasingly valuable, as suggested by the growing popularity of HE research. This project attempts to offer some understanding of this field by investigating HE for surveillance through techniques to optimise network transmission and modification of standard inference algorithms to support HE data, overcoming the inherent computational challenges.

1.3 Threat Model

The Machine Learning as a Service (MLaaS) framework describes a business model in which customers send data to a server for machine learning inference to be performed; then, results are returned. Specifically for this project, security companies analyse video surveillance data remotely. Suppose that a subscriber to one of these services, *Alice*, uses a camera to record activity at her front door. This exposes two critical threats: (*i*) an adversary, *Eve*, may eavesdrop on the transmitted data while it is in flight, and (*ii*) a malicious actor, *Mallory*, may exfiltrate the data while the server stores it, either by hacking the company or or by having privileged access, revealing a range of privacy risks including identity theft, monitoring intimate behaviours of household members, identifying household objects, and more. Figure 1.1 illustrates this succinctly. The first threat can be mitigated relatively easily using cryptographic protocols such

[1] A *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.

as TLS [60]. However, the second threat is much more difficult to defend against, particularly because data must usually be decrypted before inference [5].

Fortunately, HE offers a potential solution to both 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.

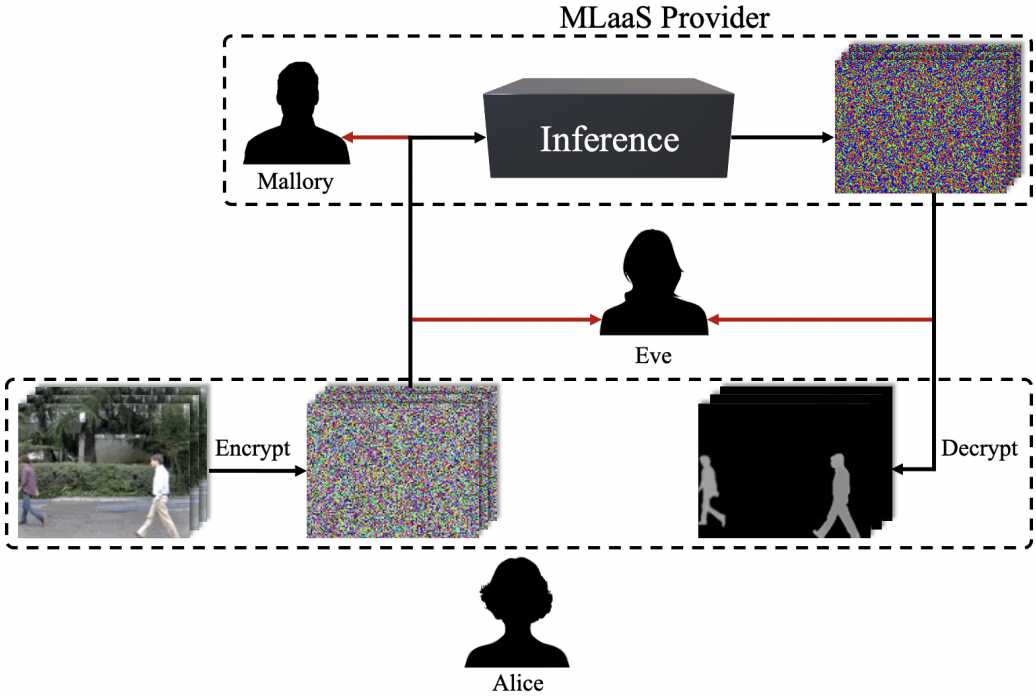


Figure 1.1: A graphical representation of the threat model. Eve is an eavesdropper able to listen to communications between user and service provider. Mallory is a malicious actor within the service provider able to access users' video data. HE is able to obstruct both Eve and Mallory, preventing them from discovering the contents of the user's video.

1.4 Project Contributions

This dissertation documents the design and implementation of a novel investigation into HE for video inference. It provides preliminary insight into the most challenging aspects of integrating the distinct fields of cryptography and computer vision to encourage further research.

- **Networking:** a client-server application simulating the device-server stack utilised by surveillance companies was constructed to enable the exploration of optimisations to increase the network throughput of videos encrypted using the CKKS HE scheme [13] provided by Microsoft's Secure Encrypted Arithmetic Library (SEAL) [40].
- **Inference:** multiple inference algorithms were implemented to permit private and plain moving object detection, including investigating online GMMs following Stauffer and Grimson [59] and the Expectation-Maximisation algorithm [18].
- **MeKKS:** a HE implementation from first principles following the Homomorphic Encryption for Arithmetic of Approximate Numbers paper by Cheon et al. [13, 11] to examine the benefits of specialising the implementation for video data by removing unneeded functionality, simplifying data structures, and vectorising ciphertexts.

Chapter 2

Preparation

This chapter discusses the preparatory work done before beginning the project's implementation. §2.1 introduces the relevant concepts, terminologies, and notations from cryptography, graphics, and unsupervised machine learning fundamental to the project, §2.2 discusses the methodologies used when approaching the design of the project's implementation, and §2.3 provides an overview of the project's foundations.

2.1 Preliminaries

2.1.1 Homomorphic Encryption

Introduction

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

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

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

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

where $m_1, \ldots, m_n$ are the plaintext equivalents of $c_1, \ldots, c_n$. This is, perhaps, better illustrated by Figure 2.1 below. Theoretically, a *fully* homomorphic scheme[3] allows the evaluation of

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

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

[3] The prefix *fully* derives from the existence of *partially* homomorphic schemes. A partially homomorphic scheme will only allow certain operations on the ciphertext - usually multiplication and division - and have existed for many years. Some examples of partially homomorphic schemes include RSA, ElGamal, and Paillier encryption [49].

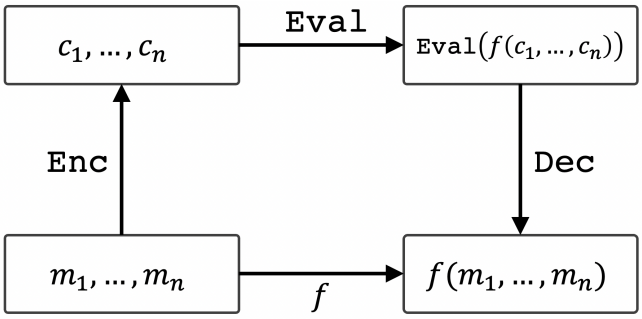


Figure 2.1: Homomorphic Encryption

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

Ring Learning with Errors

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

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

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

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

While it is possible to create perfectly secure encryption schemes[4], they are impractical in real applications. Therefore, *computational security* is considered sufficient. Relying on the hardness of certain mathematical problems, computational security means that an encryption scheme is *practically unbreakable*: the most efficient known algorithm for breaking a cipher would require more computational steps than an attacker with unlimited hardware could perform. For example, the RSA encryption scheme 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 integer factorisation problem. In complexity theory, this problem falls into the set of $\mathcal{NP}$.

Similarly, the HE schemes used in this project rely upon computational security. They utilise the hardness of the *ring learning with errors* (henceforth RLWE) problem introduced by Lyubashevsky et al. [44]. 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 upon RLWE for computational security.

To understand the RLWE problem, knowledge of group theory is required. The relevant definitions are introduced in Appendix A.1. The RLWE problem requires the ring formed by the set of polynomials modulo $\Phi(X)$ that also have coefficients in $\mathbb{Z}_q$[5]. Known as a *quotient*

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

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

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 polynomial vector computed using s and some sampled errors. An encryption scheme can be created such that, after encoding a plaintext vector, $\mathbf{v}$, as a list of polynomials and it to a polynomial using a secret polynomial, it is infeasible to recover $\mathbf{v}$ in polynomial time.

The HE schemes discussed in this dissertation rely upon 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*, the adversary's chances of calculating the correct plaintext when given a ciphertext should be no better than if they were randomly guessing.

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

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

In order to speed up computation, n is selected to be an even power of two. Consequently, $\Phi_n(X) = X^{\frac{n}{2}} + 1$. This allows the *number theoretic transform* (NTT)[6]. The advantage of this is that it can be easily accelerated using hardware [37].

The sampled errors used when deriving ciphertexts imply almost exponential error growth in the number of multiplications applied. The limited depth property of levelled HE schemes directly results from this. However, the relative growth size can be reduced by increasing the modulus q. Although, if a polynomial degree of size $n/2$ is used, efficient attacks exist against the RLWE problem for a small value of q [1]. Therefore, a fundamental trade-off is introduced between the supported depth of multiplication and the security level.

2.1.2 Moving Object Detection

Introduction

Image segmentation is a well-established problem 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 is to simplify an image representation so that it is easier to, for example, locate objects or boundaries, allowing further analysis. More precisely, image segmentation describes labelling each pixel of an image such that all pixels sharing a particular characteristic are assigned the same label [56].

There are two broad categories of segmentation techniques: *semantic* [28] and *instance* [64]. 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.

To perform *foreground extraction*, also known as *background subtraction*, the background of an image must be modelled so that changes in the scene can be detected. This is an unsolved

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

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 differences. There are several methods for achieving this. Below are five algorithms investigated for this dissertation.

Frame Differencing

Frame differencing is the most straightforward background subtraction algorithm [35]. First, a reference frame is established. There are several options for this, including selecting the first frame, or continually updating to the frame previously received. Challenges include ensuring the model can adapt to permanent changes, like a fence being added to a property, and accounting for random noise. Updating the reference too frequently can cause issues with slow-moving objects becoming occluded.

Once the reference frame, B, has been selected, the foreground can be extracted. Denoting 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 Equation2.5.

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

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 [65]. Instead of taking a frame directly from the video, the value of B at time t is calculated using Equation 2.6.

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

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

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

Median Filter

The *median filter* algorithm is similar to performing mean filter extraction. Instead of using the mean to calculate the reference frame, the median of the preceding N frames is used [65]. Then, 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 [62]. Rather than storing an image as the reference frame, this method stores a mean and variance for each pixel. 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.

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

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

where α determines the size of the *temporal window* used to fit the Gaussian model[7], $d = |f_t - \mu_t|$ gives the Euclidean distance from the pixel to the mean, and c is some constant defined by the model creator.

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

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

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

A variant of this algorithm might only update the model 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 constant changes.

Gaussian Mixture Models

Stauffer and Grimson proposed *Gaussian mixture models* (henceforth GMMs) for moving object detection in 1999 [59]. 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 labelled automatically, constituting *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 $\mathbb{P}(x \text{ generated by component k}) = \phi_k$. If the component weights are learned, they are known as *a-posteriori*[9] estimates of the component probabilities given the data.

When fitting a GMM, the Gaussian distributions are tuned to match the distributions observed in the data. A common method for this is to use *expectation maximisation* (EM), if the number of components is known. If all of the data is available, 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 extrapolate to more values. This technique can also

[7] α weights frames according to age. 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.

[8] A probability derived purely through deductive reasoning.

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

be taken advantage of if there is limited time to perform fitting.

Once a GMM has been fitted, it can be used for inference. This dissertation will focus on inference for *clustering* because it is most useful for moving object detection. The posterior component assignment probabilities can be estimated using Bayes' theorem over 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 project required the development of theoretical knowledge before implementation began so the requirements evolved as understanding matured. The original requirements are given in Appendix C for comparison. The requirements have been grouped into two categories. The first, labelled A, are the core requirements essential to the project's success. The second, labelled B, are extensions, aiming to improve understanding or further the investigation into HE and surveillance.

	Requirement	**Priority**	**Difficulty**
A1	Implement a client-server application allowing videos to be homomorphically encrypted and transmitted in both directions.	High	Medium
A2	Implement background subtraction models that can extract moving objects from homomorphically encrypted videos.	High	Medium
A3	Evaluate the accuracy of HE inference to investigate its applicability to real systems.	High	Low
B1	Implement a bespoke HE scheme and integrate it into the core application, providing the same functionality as the established scheme already used.	Medium	High
B2	Analyse the security of the encryption schemes used in the project.	Low	High
B3	Implement a deep-learning object recognition algorithm acting on HE data following Cryptonets [19].	Low	High

Table 2.1: Requirements analysis.

2.2.2 Methodology

Different stages of the project were best suited to different development methodologies. For core components, a waterfall methodology was adopted [53]. The requirements were detailed and unambiguous, so the project lent itself to a structured methodology, not requiring the flexibility of an iterative approach. The model's stages are detailed below.

The results of the *requirements analysis* stage have been provided in §2.2.1. This stage is where most of the research was performed to ensure the project's design was better informed.

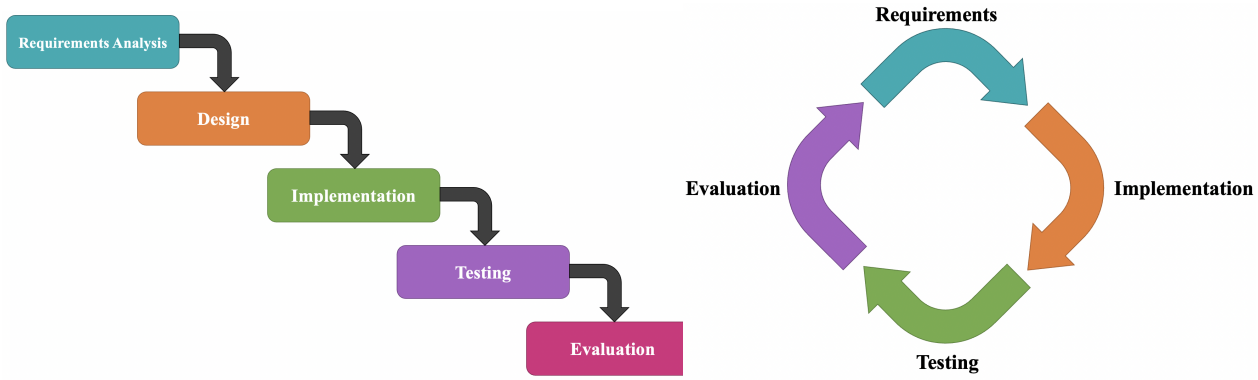
The *design* phase incorporates expanding on the requirements into a physical project; this includes, for example, creating the class diagrams shown in Figure 3.3a, Figure 3.3b, and Figure

3.4.

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 impact. Therefore, the Agile model, depicted in Figure 2.2b was selected. 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) The Waterfall Development Model (b) The Agile Development Model

Figure 2.2: Development Models

A Gantt chart of the project's timeline is shown in Figure 2.3.

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 independence resulting from the object-oriented design approach to test components in isolation. Test cases provide expected, boundary, and erroneous data to ensure function results match the expected. Unit tests can be automated to allow repeated checks 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 by ensuring separate modules interact correctly. Once unit testing has been completed, these tests aggregate verified

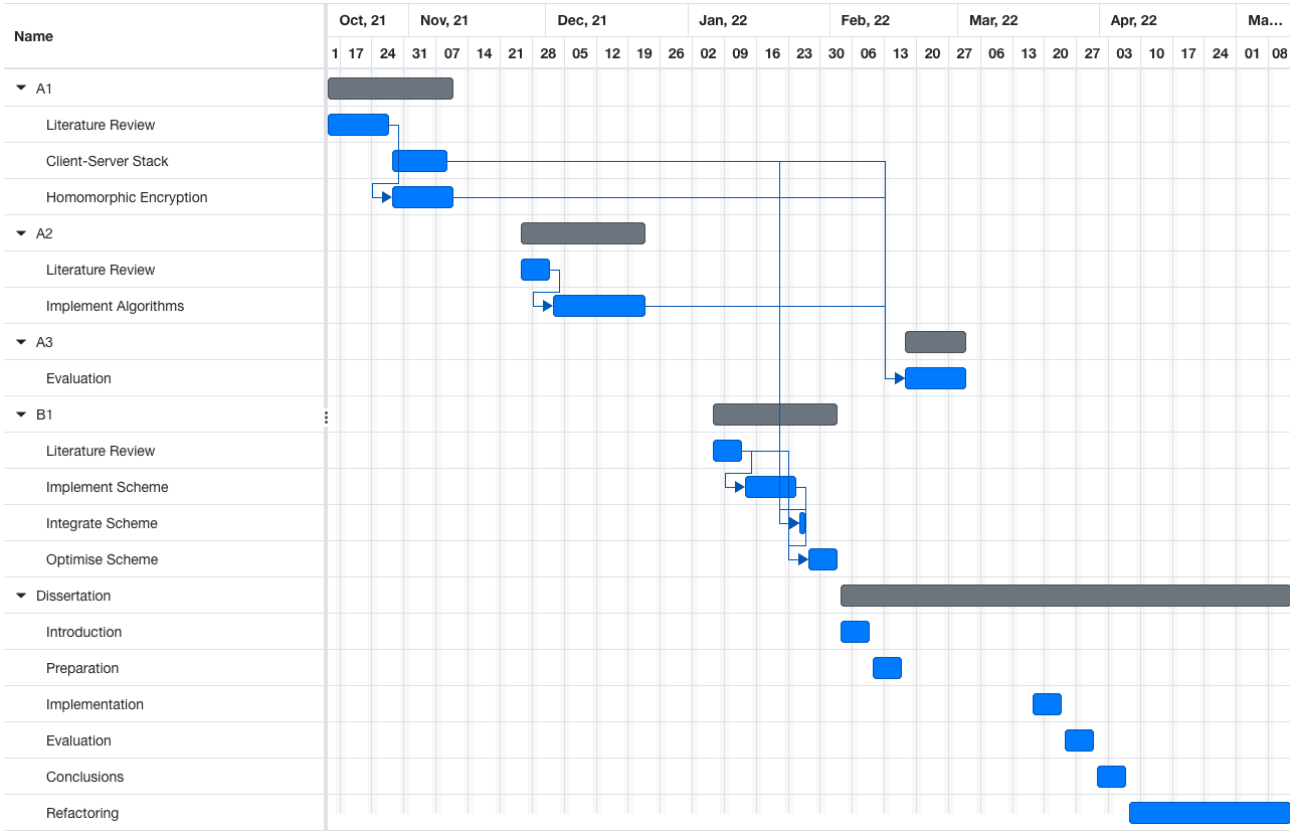


Figure 2.3: Project Timeline

modules and provide 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 comprehensive testing suite, so manual integration testing was primarily used.

Manual Verification

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

2.3 Starting Point

2.3.1 Knowledge and Experience

Prior to beginning the project, the following relevant Tripos courses 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 theory was learned independently of Tripos studies. The study of applied HE is

sparsely documented, so most understanding came from academic papers; notably [13] and [40]. Although, articles such as [10] 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. Academic papers such as [59] and [39] 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 code written for this dissertation was written in *Python* [24]. The main reasons for this were the large machine learning ecosystem, ease of use, and ease of debugging. These factors allowed for quick implementation, making Python best suited to the project's tight schedule.

However, Python is not a language traditionally used for cryptographic applications. Usually, lower-level, faster languages like C++ are favoured. Since the 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 [48] 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* [25] and *GitHub* [30] were used for version control and source code management. *OneDrive* [47] 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 availability of academic literature discussing them. In particular, the CKK scheme [13] was selected because it supports representing real numbers[10]. However, the project is designed to allow any HE scheme following the same API to be substituted in CKKS's place.

Libraries

The project uses Microsoft's SEAL library [40], 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. SEAL was integrated using a Python wrapper library [29].

Datasets

There were two publicly available datasets used in this project:

- The Moving-MNIST dataset contains ten thousand sequences of length twenty frames showing two handwritten digits from the standard MNIST dataset moving in a 64×64 pixel frame [58, 41]. 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.

[10] Versus the BFV scheme, which only supports integers [9, 22].

- The LASIESTA dataset contains sequences showing individuals moving across static backgrounds [17]. Specifically designed to evaluate segmentation algorithms, this dataset provides more realistic examples of surveillance footage so allows a truer evaluation of moving object detection in the HE domain.

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 2.2 gives the specific licenses.

Dependency	Licence

`Multiprocessing`
`NumPy`
`SymPy`
	3-Clause BSD [31]

`Microsoft SEAL`
`SEAL-Python`
`PyJoules`
	MIT [32]

`Matplotlib`
`Python`
`Tkinter`
	PSFL [23]

Table 2.2: 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 2.3.

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

Table 2.3: Computer Specifications

Chapter 3

Implementation

3.1 System 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 Software Design

The project's design began with understanding the MLaaS threat model described in §1.3. Figure 3.1a depicts an abstract layout of the project's core components. There are two categories of components: *online* describing inference performed directly in response to a user's request to emulate the MLaaS model, and *offline* describing inference results generated on batches of data, independent of the frontend. The offline components are used during the implementation and evaluation stages to develop and refine the system (labelled *testing*), and to collect and analyse the results presented in Chapter 4 (labelled *evaluation*).

An overview of the project's repository is given in Figure 3.2. Excluding SEAL, all code was written for the project. An object-oriented design methodology was used to allow separate components to be implemented independently and isolate the layers depicted in Figure 3.1. Another advantage is that it allows straightforward substitution of different inference methods and encryption schemes.

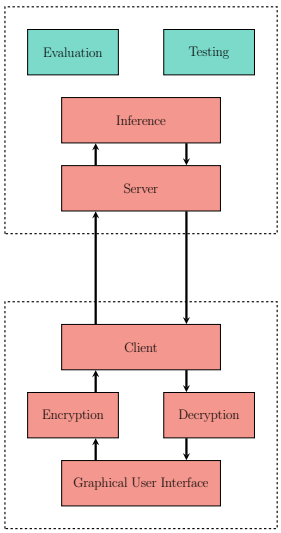
The application's online components can be split into four abstract layers, from high-level interface to low-level implementation.

1. The *graphical user interface* component allows users to configure the encryption scheme and inference method, and upload videos.

2. The *client* and *server* components are responsible for managing the network communication to allow videos to be transferred to the server for inference, and the results returned to the client.

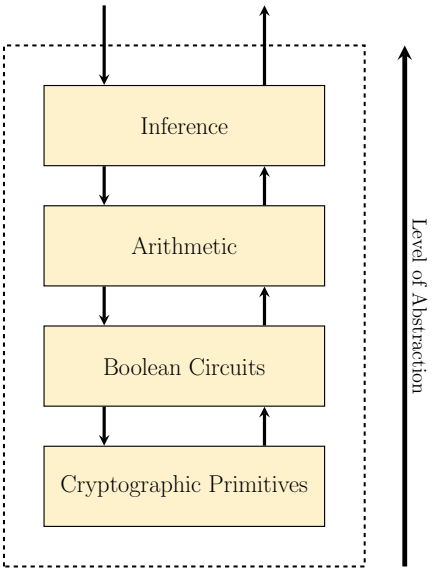
3. The *encryption* component provides an API for the cryptographic primitives to allow videos to be encrypted and decrypted in the client, and operated on by the server in the *inference* component.

4. The implementation of the cryptographic primitives provided by the CKKS scheme through the *MeKKS* and *SEAL-Python* libraries.



(a) The project components.

Online: Offline:



(b) Abstract layers of the inference stack.

Figure 3.1: High-level implementation overview.

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, lower layers exist. Another layer relevant to this investigation may be the hardware implementation. The IoT devices used for surveillance cause significant constraints on computational performance. Accelerators, such as GPUs, could be investigated to improve running times of cryptographic operations on these devices [4].

3.1.2 Class Structure

This section provides a more thorough insight into the project's structure. Figure 3.3a details the arrangement of the client-side, Figure 3.3b contains the classes composing the server-side, and Figure 3.4 depicts the structure of the MeKKS library. While overlaps between diagrams exist, they have been separated into three figures for clarity.

3.2 Encryption Schemes

This section briefly discusses the fundamentals of the CKKS HE scheme described by Cheon et al. [13], and the reimplementation completed for this project. Also, to provide an investigation into specialising HE schemes, a bespoke CKKS implementation, called MeKKS, is detailed.

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 are 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 $\mathbf{w}$. Before encoding, the vectors are scaled to $\Delta\mathbf{v}$ and $\Delta\mathbf{w}$.

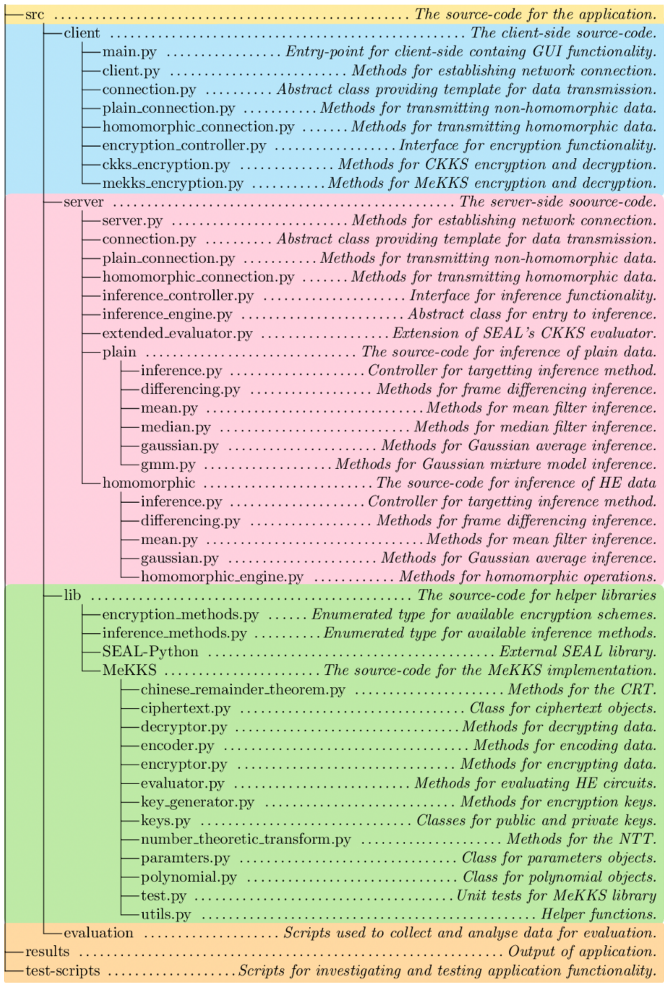


Figure 3.2: Repository overview. In total, 6,322 lines of code were written.

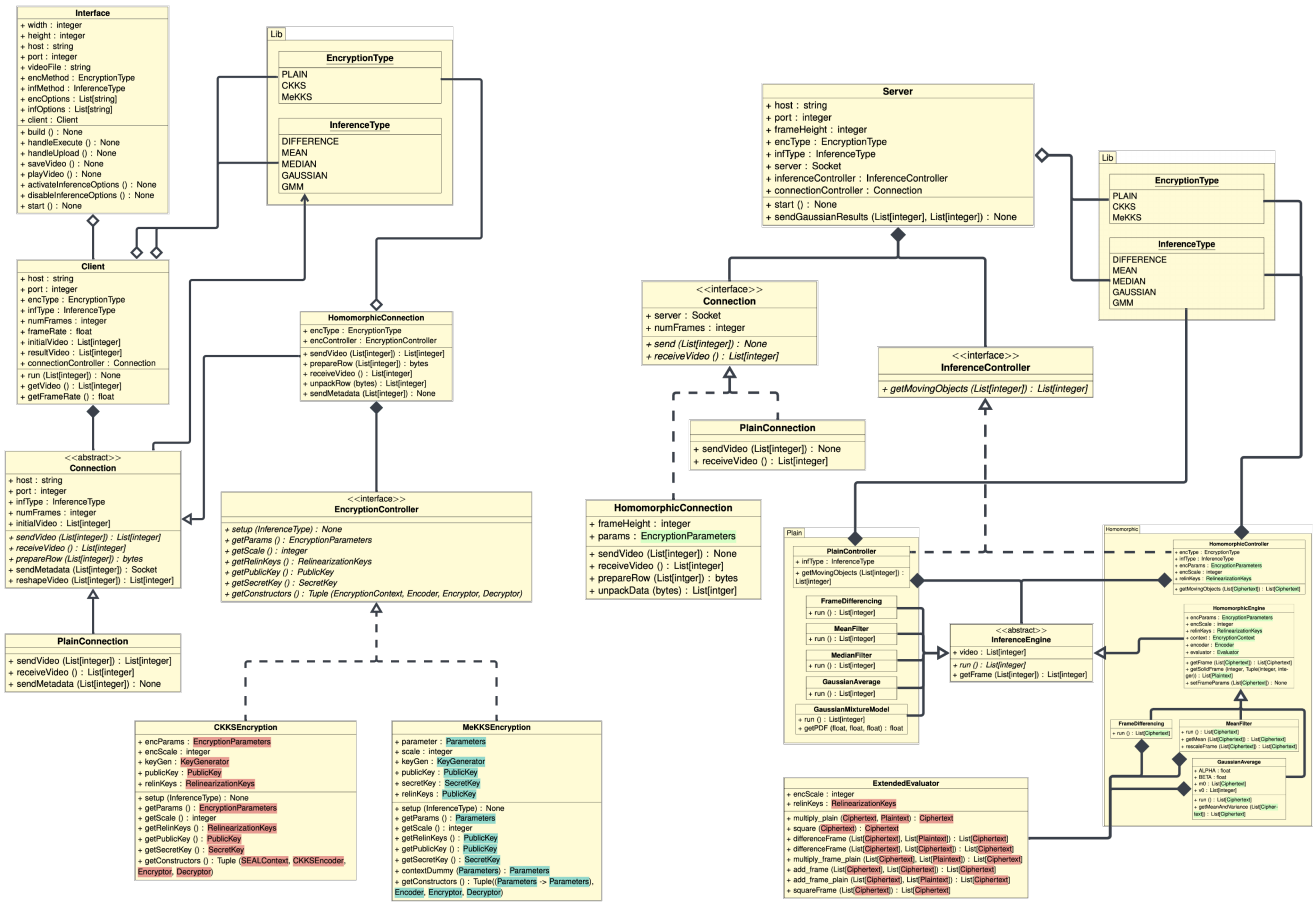
They can then be encrypted and multiplied to produce ciphertext equivalents of $\Delta^2(\mathbf{v} \bigodot \mathbf{w})$. This implies that a sequence of multiplications will continuously increase the scale factor. To overcome this, CKKS introduces a *rescaling* procedure, which can be understood as dividing the ciphertext by Δ.

However, rescaling cannot be repeatedly 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 first encrypted, a polynomial exists in the maximum level, L, with coefficient modulus $Q = q_0\dot{\Delta}^L$, for a *base modulus* q_0. When a ciphertext is rescaled, the coefficient modulus is divided by Δ, reducing it to $q_0 \cdot \Delta^{L-1}$. Hence, the ciphertext is lowered to the next level. If a ciphertext reaches level 0, no further multiplications can be applied to preserve the encrypted values. To ensure decryption produces correct results, a polynomial's coefficients cannot exceed q_0. Therefore, practical implementations will use a q_0 much larger than Δ.

Encoding and Decoding

The plaintext message space is the *cyclotomic polynomial ring* $\mathcal{R} = \mathbb{Z}[X]/\Phi_M(X)$, for the M-th cyclotomic polynomial, where M is a power of two. Encoding describes 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 $\sigma : \frac{\mathbb{R}[X]}{X^N+1} \to \mathbb{C}^N$. This is applied to each root of $\Phi_M(X)$ through the



(a) Client

(b) Server

External CKKS classes:

External MeKKS classes:

External CKKS classes:

Abstract HE objects:

Figure 3.3: UML Class Diagrams showing the core project components.

evaluation of r as

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

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

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

Hence, the CKKS encoding function is given by Equation 3.2.

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

Operations

Figure 3.5 lists the operations supported by CKKS and their definitions. An important observation is that ciphertext multiplication requires polynomials to be multiplied. This makes multiplication a much more computationally expensive operation than addition.

Rescaling

The rescaling function introduced above is defined by equation 3.3.

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

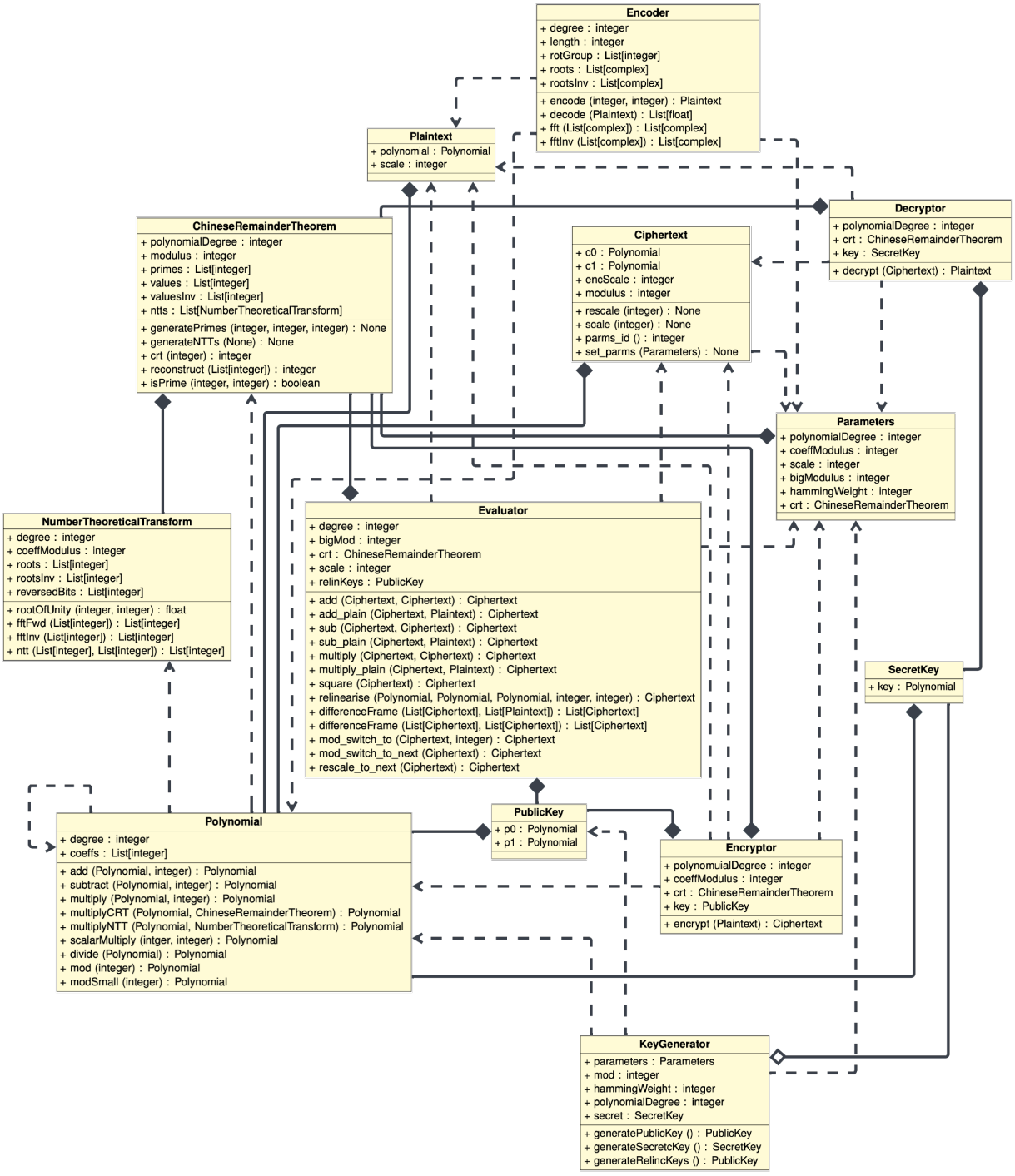


Figure 3.4: UML Class Diagram showing the components of the MeKKS encryption scheme.

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 $\mathbf{V} = [v_0, v_1, v_2, \ldots, v_n]$ and an offset d. A rotation would cyclically shift each element by d indices[1]. This is achieved by performing a *Galois automorphism* on the

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

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

Figure 3.5: A list of the basic operations supported by the CKKS scheme. P is a large scaling factor and $\lfloor \cdot \rceil$ denotes rounding to the nearest integer. Recreated from [13].

ciphertext using standard results from *Galois theory*. However, these operations are the most expensive operations offered by RLWE-based schemes [34].

RNS Optimisations

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

Unfortunately, as a result of the RNS optimisation, $q_l = q_0 \Delta^l$ cannot always be maintained due to the difficulty of finding every q_l as the product of n primes. To overcome this, SEAL instead defines $q_l = q_0 \prod_{i=1}^{l} p_i$ where q_0 must be prime. Where the original CKKS scheme divides by Δ to rescale, SEAL's implementation divides by p_l. Consequently, the scaling factor is not exactly equal to Δ. In practice, the scaling factor must be manually reset to Δ after each rescaling.

3.2.2 Exploiting Specialisation

As an extension to the project, the CKKS scheme was reimplemented from first principles to create MeKKS. This allowed the investigation of a HE scheme specialised for this application through simplification of data structures and removal of unnecessary functionality.

Limitations

The primary limitation of MeKKS was the language used for implementation. Traditionally, encryption schemes are written in C or C++ to expose low-level functionality. Consequently, the more computationally expensive operations, such as large prime number multiplications, would run much quicker than in Python.

Also, in contrast to CKKS, MeKKS is severely limited by optimisations that have been applied. This extension began by focussing on improving HE understanding, so the foundations of CKKS were implemented. However, CKKS has been significantly optimised since it was originally proposed. For example, using residue number systems, bootstrapping extensions, and reduced approximation error [12, 11, 36]. The project's time constraints meant that defining a clear implementation endpoint was essential for scheduling. Therefore, MeKKS is significantly less optimised than SEAL's CKKS, limiting performance expectations.

Extending MeKKS

The first iteration of the MeKKS implementation began with the scheme presented by Cheon et al. in 2016 [13]. This allowed all of the functionality described in §3.2.1 to be implemented following the SEAL API, enabling easy integration into the core project.

However, this implementation's performance meant it was infeasible for recording results. Therefore, the bootstrapping procedure from the next HEAAN iteration was implemented [11]. Bootstrapping takes advantage of the *approximate computation* characteristic of HEAAN to evaluate the decryption formula approximately so a ciphertext can be obtained in a large modulus. Hence, an approximation of the *modular reduction* formula was implemented to enable efficient evaluation 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 $t = \langle \texttt{Ciphertext}, \texttt{SecretKey} \rangle$ is close to a multiple of q (the ciphertext modulus). Specifically, using the *sine* function given by Equation 3.4.

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

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

The Taylor polynomial can be used to approximate the trigonometric function to enable evaluation 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 $O(K \cdot q)$ to ensure the error term is small enough on the interval $(-K \cdot q, K \cdot q)$. Cheon et al. proposed using the *Paterson-Stockmeyer* method to reduce the calculation complexity [11, 50]. 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.

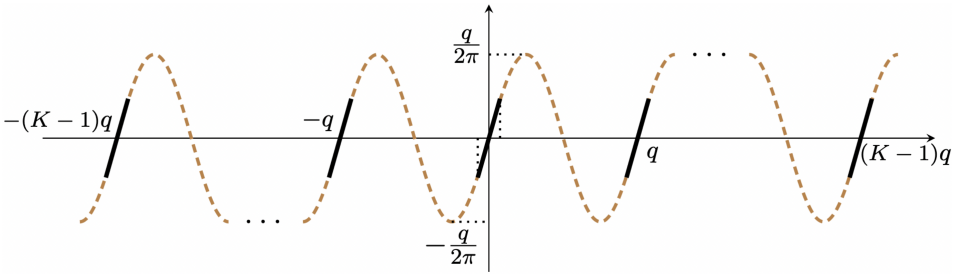


Figure 3.6: Modular Reduction and Scaled Sine Functions. Reproduced from [11].

Specialising MeKKS

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. Python struggled to manipulate these objects efficiently, specifically when serialising data for transmission. However, since MeKKS was written in Python, object attributes and functionality were exposed, providing much more efficient manipulation.

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. Therefore, the required functionality had been finalised, so only necessary operations were implemented for MeKKS. This reduced the size of objects and removed any unused computation, making execution more efficient. For example, the most expensive operations offered by CKKS are rotations. However, since the application does not use them, they were not included in MeKKS.

3.3 Optimising Network Throughput

This section details the techniques investigated for optimising communication between the client and server. An established limitation of HE is the size of ciphertexts [46]. Consequently, the *transmission time* of data is significantly impaired. The *transmission time* can be defined by Equation 3.5.

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

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: reducing the *video size* and increasing the *transmission rate*.

3.3.1 Reducing 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 of techniques for distinguishing these features. *Top-down* methods use tools such as *face detectors* to highlight features in the image [2]. *Bottom-up* approaches use saliency maps[2] to locate important areas [38].

The details of the original seam carving paper, and a depiction of the algorithm, are included in Appendix B.1. However, a more advanced algorithm was implemented to investigate more optimal results.

To quantify the importance of a pixel, seam carving defines an *energy function*. Rubinstein et al. [54] 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

[2]a representation highlighting the regions of an image where a person's eyes are first drawn [57].

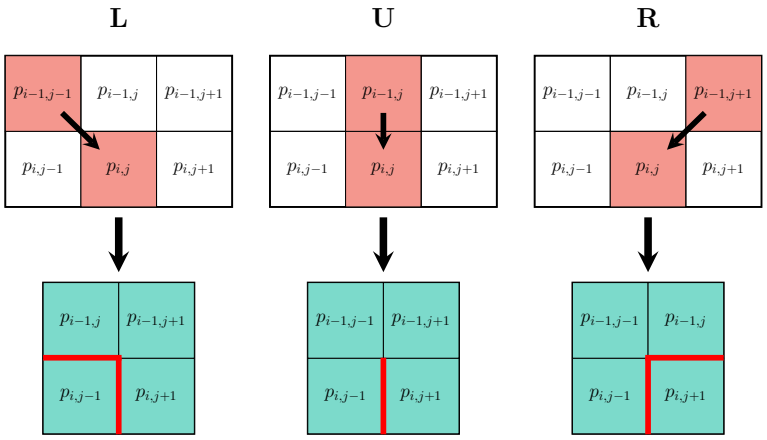


Figure 3.7: There are three possible vertical seam costs for pixel (i, j). After removing the seam, new neighbours (blue) and new edges (red) are created.

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.

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

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

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

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

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, only including a single pixel in each row. Likewise, a *horizontal seam* connects the left of an image to the right, only including a single pixel from each column. These is defined by Equation 3.10a and Equation 3.10b respectively.

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

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

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

From this, 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.

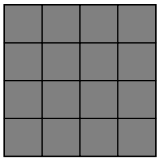
$$M(i,j) = e(i,j) + \min \begin{cases} M(i-1, j-1) + C_L(i,j) \\ M(i-1, j) + C_U(i,j) \\ M(i-1, j+1) + C_R(i,j) \end{cases} \tag{3.11}$$

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 [54, 20].

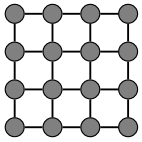
Graph Representation

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

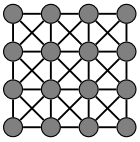
Graph-based image processing methods operate on *pixel adjacency graphs* - graphs whose vertex set is the set of image pixels and edge set defines adjacency of pixels. An example of some pixel adjacency graphs is given by Figure 3.8. Three-dimensional pixel adjacency graphs can account for relationships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



(a) A 2D image with 4×4 pixels.

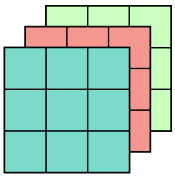


(b) A 4-connected pixel adjacency graph.

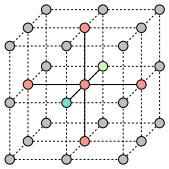


(c) An 8-connected pixel adjacency graph.

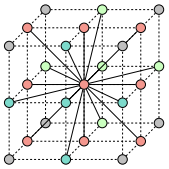
Figure 3.8: Pixel adjacency graphs.



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



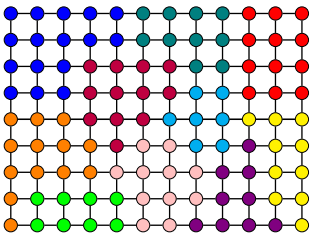
(b) A 6-connected 3D pixel adjacency graph.



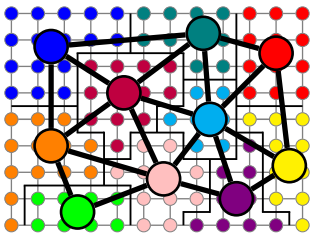
(c) An 18-connected 3D pixel adjacency graph.

Figure 3.9: 3D pixel adjacency graphs.

However, to improve the video size, pixel adjacency graphs must be extended to *region adjacency graphs*. 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.



(a) A pixel adjacency graph.



(b) The corresponding region adjacency graph.

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

To achieve this, similarity between pixels must be quantified. This constitutes another image segmentation problem. Consequently, established algorithms producing valid solutions exist. Unsupervised clustering algorithms such as *the watershed transform* [6] or *k-means clustering* [45] are two methods that have proved useful in existing works.

Importantly for this investigation, the number of regions in the image will directly impact

the transmission time. Reducing the number of nodes in the graph is advantageous because it reduces the amount of data transmitted. However, this also reduces image resolution. Consequently, removing too many nodes from the graph will remove clarity, making inference worthless. Figure 3.11 depicts this. Therefore, a balance must be struck heuristically to maximise video size reduction while minimising impact on video quality. This optimal point is likely to be different for every image, adding a further layer of complexity.

(a) A photograph of a cup of coffee.

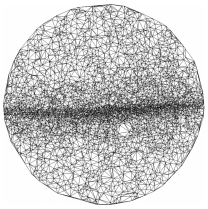
(b) The picture of coffee split into approximately 400 regions.

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

Figure 3.11: A demonstration of the impact of reducing the number of regions on image quality. Images taken from [26].

Using similar techniques to seam carving, it is possible to limit the severity of this trade-off. For example, *Foveal sampling*, depicted in Figure 3.12, utilises the visual activity of the eye to determine regions [26]. 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 some ducks.



(b) The retinal topography of a kangaroo.

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

Figure 3.12: An example of Foveal Sampling. Images taken from [26].

3.3.2 Increasing Transmission Rate

This section targets the bottlenecks limiting the transmission rate of the system by investigating the application of *parallel computing*.

Figure 3.13 depicts the abstract layers of the application's networking processes. The components are split into two categories: the *client* and *server* components handling communication, and the *packing* and *unpacking* components manipulating data before and after transmission. Parallelisation was selected for investigation because of the growing support in computer architecture. Traditionally, computer design has focussed on *sequential computing* to improve performance. However, factors such as Dennard scaling [63] mean the improvements predicted

by Moore's law [55] may not continue indefinitely. Therefore, architects are utilising multiprocessing to achieve similar gains. Consequently, this seemed like a viable opportunity for the investigation to consider future iterations of surveillance technology.

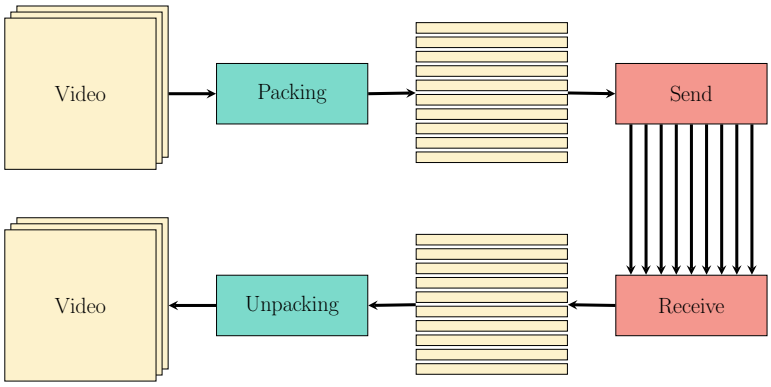


Figure 3.13: The stages required for a video to be sent across the network.

Communication

Parallelisation already exists in networking protocols. The *Transmission Control Protocol* uses a *sliding window protocol* to send groups of data packets concurrently, ensuring they are reordered correctly when received. More information is provided in Appendix B.2. These protocols exist in the *data-link* layer of the *OSI network model* [16]. This section of the investigation aimed to evaluate the result of moving parallelisation higher up this abstract stack.

Taking inspiration from sliding windows, instead of using a single network socket to send data in a single stream, videos are split into frames, and each frame is divided into packets. Meanwhile, a pool of threads is created, analogous to the window, and mapped to packets, enabling multiple sockets to be opened in parallel, increasing the amount of data sent simultaneously.

However, there are limitations to this technique. Firstly, more data must be transmitted than during sequential communication. The algorithm is non-deterministic, so the order of packet delivery cannot be guaranteed. Consequently, further information must be attached to packets 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. This cost means creating too many threads will countermand parallelisation benefits and make transmission slower. Therefore, an optimal balance between the cost of parallelisation and the volume of data to send must be achieved to maximise improvements.

Data Manipulation

Splitting videos into small packets has further advantages when preparing data for transmission. Before data can be transmitted, it must be prepared, or *packed*. When data arrives at its destination, it must be *unpacked*. There are three distinct stages to packing: *encryption*, *compression*, and *serialisation*, which are reversed during unpacking. The encryption stage is missing from the server-side pipeline.

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 complexity.

Moreover, decomposing a video into independent packets allows the packing and unpacking pipelines to be executed in parallel. Therefore, the cost of encryption, compression, and serialisation can be amortised by preparing multiple packets concurrently. This process can take advantage of the multithreading described in §3.3.2 so that no further overhead of thread creation is required. Consequently, potential improvement is only limited by the need for each stage to terminate entirely before the next begins, so reducing the size of packets allows pipelines to terminate faster. However, smaller quanta require more communication threads, so this cannot be indefinitely exploited.

3.4 Moving Object Detection Inference Algorithms

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.

3.4.1 Adapting Inference Algorithms for Homomorphic Encryption

There were two main challenges to overcome when implementing inference algorithms for HE. 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 for 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 size of ciphertexts, making data transmission infeasible.

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

Frame Differencing

Frame differencing required few modifications to accommodate HE data. Only a single operation is required: *subtraction*, and is provided by the standard CKKS implementation. Moreover, subtraction does not impact a ciphertext's level, so does not require size modifications. As well as making networking more efficient, this makes the inference algorithm faster because the data being operated over is minimised.

Therefore, the only modification required to support HE data is to replace the subtraction function in the traditional algorithm with the subtraction circuit of a HE library.

Mean Filter

A naïve mean filter implementation recalculates the reference frame repeatedly by storing a list of all observed frames and computing their mean each when a new frame is received. Alternatively, a more advanced implementation will use an iterative function to update the mean, removing the requirement to store preceding frames.

Evidently, the second method will produce better 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 when updating. Each time a multiplication circuit is applied, the

ciphertext will be reduced. Therefore, the ciphertext must have the same number of levels as frames in the video. This quickly becomes infeasible; increasing 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 issue is the space complexity of storing all frames in the video. Since HE data is very large, storing multiple copies of each frame significantly increases memory usage. Similarly, HE operations are noticeably slower than plaintext operations. Therefore, while the cost of recalculating the mean of plaintext data may be negligible, HE implementations will become increasingly slower. However, a solution can be derived. 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 Gaussian average implementation might fit distributions using iterative formulae for the mean and variance. Copied from §2.1.2 for convenience, Equation 3.12 and Equation 3.13 provide these formulae. Consequently, the plaintext implementation will be infeasible in the HE domain. Fortunately, adaptations can be made.

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

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

where α determines the temporal window, d represents the Euclidean distance between a pixel and the mean, and c is some constant.

Firstly, 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 a video.

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

Helpfully, the coefficient terms across calculations are identical, so computation can be shared. The server decides the value of α before runtime, so 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 enable recalculating terms. 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 [59]. To overcome a single Gaussian distribution's inability to model natural pixel variation, they proposed a mixture of adaptive Gaussians. The advantage of their

technique over other GMMs is that the model runs *online*: the model can be fitted, and results returned in a single phase. While this is useful for real-time inference acting on a constant data stream, it has the disadvantage of producing less accurate results earlier in the execution sequence.

Fitting

For a particular pixel, the values that it records 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.

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

where I is the image sequence.

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

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

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.

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

To maximise the likelihood of the observed data, a *k-means* approximation was selected to engender an online implementation. Each pixel in a new frame is compared against the existing 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 pixel as its mean, an initially high variance, and low prior weight. Then, the prior weights are adjusted at time t using Equation 3.18.

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

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, μ and σ are unchanged. The parameters of the matching distributions are updated according to Equation 3.19, where *rho* is defined by Equation 3.20.

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

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

$$\rho = \alpha\eta(X_t, \mu_k, \Sigma_k) \tag{3.20}$$

Predicting

Once the parameters have been updated, each pixel is labelled by determining the Gaussian component most likely produced by the background process. This decision assumes that there will be little variance in the Gaussians when the frame is static and a large variance when a new object occludes the background. Consequently, a function defining the proportion of the GMM representing the background process is required.

First, the Gaussians are ordered by 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

$$B = argmin_b \left(\sum_{k=1}^{b} \omega_k > T \right) \tag{3.21}$$

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 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* in statistical models [18]. The algorithm contains two stages: 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*, the EM algorithm does not guarantee convergence on a maximum likelihood estimator, but rather a local maximum. To overcome this, techniques such as *random-restart hill climbing* can be employed [33].

Focussing on GMMs, the algorithm can assign observed data points to components of the model such that the likelihood of the components generating the points is maximised. The below process can formalise the E-step. First, the *pseudo-posterior* - the probability that an observation, X_i belongs to a component Z_k - is calculated using Equation 3.22.

$$\gamma_{Z_i=k} = \mathbb{P}(Z_i = k \mid X_i) = \frac{\mathbb{P}(X_i \mid Z_i = k)\mathbb{P}(Z_i = k)}{\mathbb{P}(X_i)} \tag{3.22}$$

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

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, $\gamma_{Z_i=k}$, where θ_{t-1} is the parameter generated in the previous iteration and θ_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.

$$Q(\theta^{(t)}, \theta^{(t-1)}) = \mathbb{E}\left[\log \mathbb{P}(Z \mid \theta^{(t)}) \mid X, \theta^{(t-1)}\right] \tag{3.24}$$

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

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

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

After the auxiliary function has been generated, the M-step can begin. This means maximising the value of Q to produce the optimal parameter value in Equation 3.27.

$$\theta^{(t+1)} = argmax_\theta\ Q(\theta^{(t)}, \theta^{(t-1)}) \tag{3.27}$$

where

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

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. The results of these calculations are given Equation 3.29, Equation 3.30, and Equation 3.31, respectively, where $N_k = \sum_{i=1}^{N} \gamma_{Z_i=k}$.

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

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

$$\hat{\omega}_k = \frac{N_k}{N} \tag{3.31}$$

3.5 Summary

The implementation stage began by implementing a client-server application and then integrating Microsoft's SEAL library to allow HE data transmission, satisfying the first core success criterion. Then, an investigation into optimising the network stack by implementing the *seam carving* algorithm and *graph representations* of images to reduce video size and *parallelisation* to increase the transmission rate.

Afterwards, a novel investigation into adapting moving object detection algorithms for the HE domain was completed. This required modification of algorithms to reduce the number of operations required, development of HE Boolean circuits for more complex operations, and implementation of unsupervised machine learning models.

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

Chapter 4

Evaluation

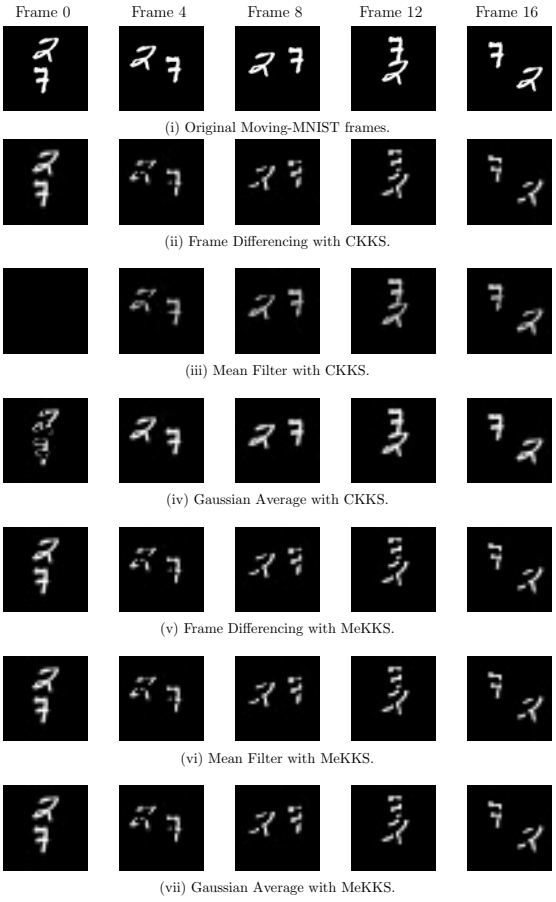
This chapter evaluates the project's implementation using three main criteria: the extent to which it meets the success criteria detailed in §2.2.1, the applicability of HE to inference algorithms, and its practicality regarding current, real-world surveillance technology. Consequently, the chapter is 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 64×64 pixel images from the Moving-MNIST dataset.

4.1 Requirements Analysis

Requirement	Achieved?	Justification
A1	✓	The project contains a client-server application that allows videos to be homomorphically encrypted and transmitted across a network.
A2	✓	The project contains several algorithms that are able to extract moving objects from homomorphically encrypted videos.
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 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.

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



(i) Original Moving-MNIST frames.

(ii) Frame Differencing with CKKS.

(iii) Mean Filter with CKKS.

(iv) Gaussian Average with CKKS.

(v) Frame Differencing with MeKKS.

(vi) Mean Filter with MeKKS.

(vii) Gaussian Average with MeKKS.

(a) Moving-MNIST

(i) Original LASIESTA frames.

(ii) Frame Differencing with no HE.

(iii) Mean Filter with no HE.

(iv) Gaussian Average with no HE.

(v) Frame Differencing with CKKS.

(vi) Mean Filter with CKKS.

(vii) Gaussian Average with CKKS.

(b) LASIESTA

Figure 4.1: Results of inference running on different datasets.

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 attempting to implement the median filter and GMM algorithms.

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 4.1a, and an example of performance on the LASIESTA dataset is included in Figure 4.1b.

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

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

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

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

(a) Homomorphic Comparison Algorithm

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

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

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

(b) Homomorphic Division Algorithm

Figure 4.2: Homomorphic Circuits

not provided by the standard CKKS implementation. To solve this, Cheon et al. proposed the algorithm in Figure 4.2a [14]. Unfortunately, this introduces security concerns. If the ability to compare two HE ciphertexts is added to the system, an attacker[1] could use it to exfiltrate information about the image. For example, with enough comparison operations, they would be able to determine the exact value of each pixel in a frame. Consequently, this algorithm was abandoned to preserve the security of the system.

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

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. [14]. However, the algorithm, given in Figure 4.2b, 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 Evaluating Real-World Applicability

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 Network Throughput Results

Data Handling

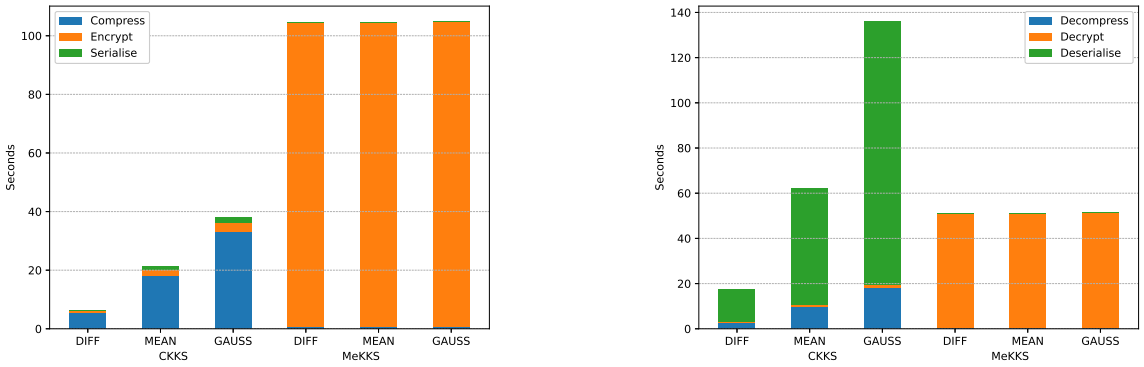
Before data can be sent across the network, it must be *packed*, and once it is received, it must be *unpacked*. This proved to be a significant bottleneck before transmitting data. During the packing phase, data must be encrypted and serialised before it is transmitted over the network. To make transmission more efficient, a compression stage is added to try and reduce

[1] such as Mallory in §1.3.

the memory usage of videos. Similarly, in the unpacking phase, data must be decompressed, deserialised, and decrypted to recover the video and inference results.

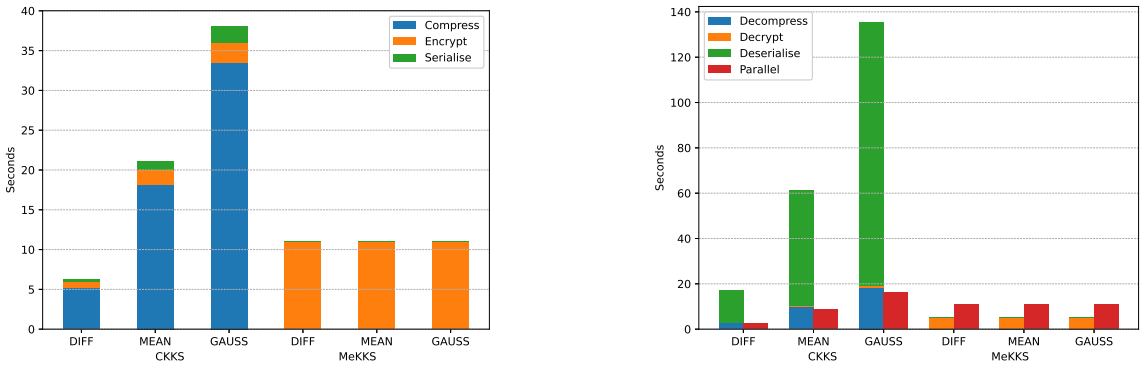
As described in §3.3, several methods were investigated to try and reduce this bottleneck. By combining some of these techniques, substantial progress was made in reducing the time the packing and unpacking algorithms took to run. Figure 4.3 provides the running time of a naïve implementation of these algorithms, and Figure 4.4 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.



(a) Packing (b) Unpacking

Figure 4.3: Naïve Packing and Unpacking Times



(a) Packing (b) Unpacking

Figure 4.4: 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 compressing videos is shown in Figure 4.5. 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 4.6 depicts the results of this adaptation.

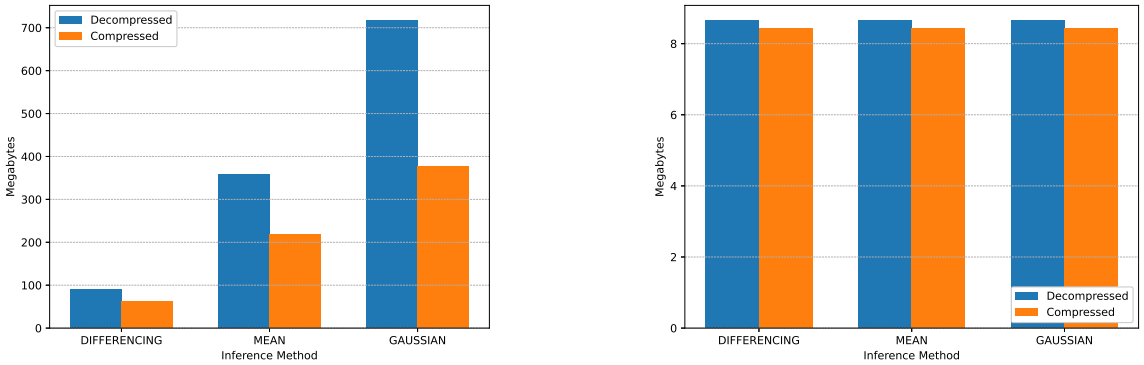
As a result of the above optimisations, the running times for the client and server are summarised by Figure 4.7a and Figure 4.7b respectively.

4.3.2 Moving Object Detection Results

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.

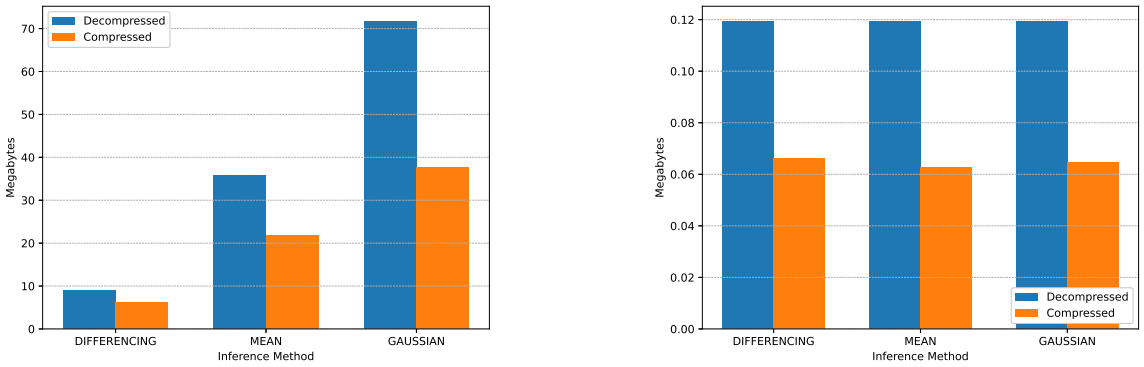
Algorithm	**Compression** (s)	**Decompression** (s)	**Size** (KB)	**Percentage**
None	-	-	716.18	100%
`gzip`	94.69	3.55	442.46	61.78%
`bz2`	36.28	5.02	435.29	60.78%
`lzma`	294.85	20.98	421.86	58.9%
`brotli`	871.97	0	435.78	60.85%

Table 4.3: Evaluation of compression algorithms.



(a) CKKS (b) MeKKS

Figure 4.5: Naïve Memory Usage Compression Comparison

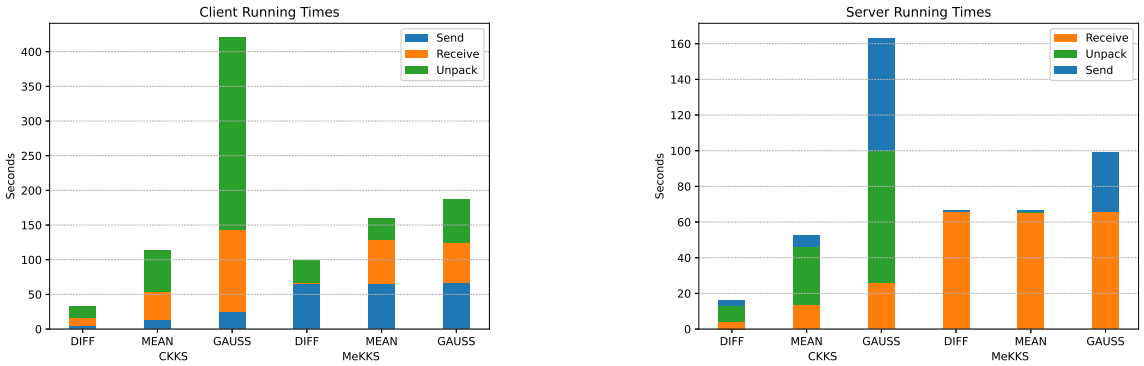


(a) CKKS (b) MeKKS

Figure 4.6: Vectorised Memory Usage Compression Comparison

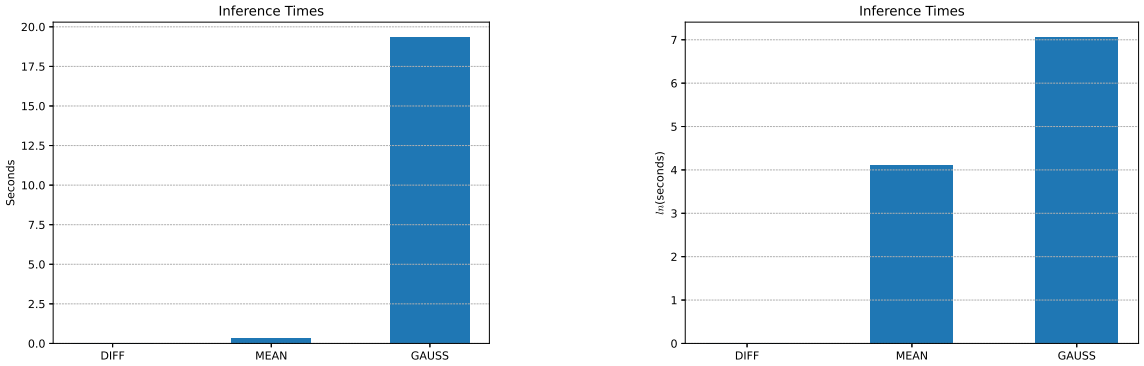
Running Time

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 4.8. Where Figure 4.7 demonstrates that the optimisations through specialisation and vectorisation can produce an improved implementation in terms of network throughput, this figure emphasises the further optimisations that are available for HE primitives. Unsurprisingly, the state-of-the-art CKKS scheme runs much quicker than the MeKKS scheme due to the inclusion of these optimisations.



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

Figure 4.7: Client and Server Running Times



(a) CKKS (b) MeKKS

Figure 4.8: Inference Times

Accuracy

The accuracies of each HE inference algorithm are compared in Figure 4.9. 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.

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 $(x,y) \in N^{N \times M}$.

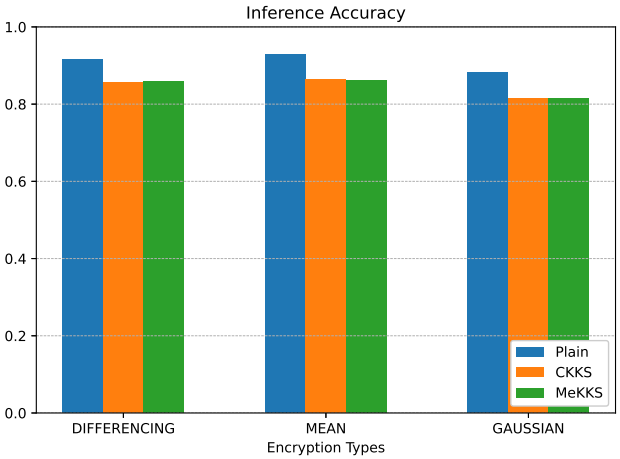


Figure 4.9: Inference Accuracy

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 4.10 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.

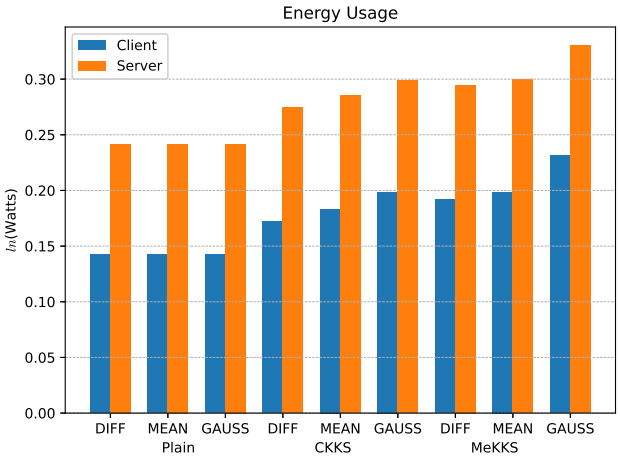


Figure 4.10: Energy Usage

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 extensions were thoroughly considered. In its final state, the project contains a novel 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 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 against traditional algorithms operating on plain data, revealing similar accuracy but significantly increased running time due to increased computational complexity. Similarly, network activity was compared between plain and encrypted data, highlighting the cost of increased ciphertext size through increased data manipulation running times and transmission times between client and server. 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 project investigated a novel integration of HE and unsupervised machine learning for image segmentation - neither of which are covered by Tripos content. Consequently, the most critical challenge was the need for research to cultivate background knowledge. However, there was little overlap between these topics found in published work, so significant time had to be invested in deriving potential solutions to encountered problems. 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 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 of HE that would benefit from further investigation. Firstly, the memory requirements resulting from ciphertext size resulted in a significant bottleneck during the networking stage of the application. Consequently, research into reducing this cost would make HE more applicable to MLaaS applications. Secondly, the variety of operations available in the HE domain must be expanded if more advanced moving object detection algorithms are to be implemented. This would enable GMMs to produce much more accurate inference results.

Regarding security, other methods could be investigated to preserve privacy. For example, techniques exist for storing private keys in hardware to prevent visibility to users [43]. This would reduce the risk of malicious actors performing unauthorised access. Moreover, *functional cryptography* is an asymmetric cryptographic protocol that allows properties of ciphertexts to be extracted [8]. An investigation into this could produce a potential alternative to HE, and compare the advantages and disadvantages of the two approaches.

More specifically for surveillance applications, further research considering the hardware aspect of the problem would be beneficial for obtaining a more precise measure of practicality. Some potential topics include: evaluating and reducing energy usage to accommodate battery-powered devices, improving computational complexity for execution on lower-powered processors, or developing specialised accelerators to design devices specific to HE computations.

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 andMichael 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 Arificial 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] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: a new vision for public-key cryptography. *Communications of the ACM*, 2012.

[9] Zvika Brakerski. Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP. *Annual Cryptology Conference*, 2012.

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

[11] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. Bootstrapping for Approximate Homomorphic Encryption. *Advances in Cryptology*, 2018.

[12] 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.

[13] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic Encryption for Arithmetic of Approximate Numbers. *Lecture Notes in Computer Science*, 2016.

[14] 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.

[15] 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.

[16] John D. Day and Hubert Zimmermann. The OSI Reference Model. *Proceedings of the IEEE*, 1983.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] Eufy. Eufy Homepage. https://uk.eufylife.com/, Retrieved March 2022.

[22] Junfeng Fan and Frederik Vercauteren. Somewhat Practical Fully Homomorphic Encryption. *Cryptology ePrint Archive*, 2012.

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

[24] Python Software Foundation. Python. https://www.python.org/, Retrieved March 2022.

[25] Git. Git. https://git-scm.com/, Retrieved March 2022.

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

[27] Thore Graepel, Kristin Lauter, and Michael Naehrig. ML Confidential: Machine Learning on Encrypted Data. *The International Conference on Information Security and Cryptology*, 2012.

[28] 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.

[29] Huelse. SEAL-Python. https://github.com/Huelse/SEAL-Python, Retrieved March 2022.

[30] Github Inc. Github. https://github.com/, Retrieved March 2022.

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

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

[33] Sheldon H. Jacobson and Enver Yücesan. Analyzing the Performance of Generalized Hill Climbing Algorithms. *Journal of Heuristics*, 2004.

[34] 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.

[35] 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.

[36] Andrey Kim, Antonis Papadimitriou, and Yuriy Polyakov. Approximate Homomorphic Encryption with Reduced Approximation Error. *Topics in Cryptology - CT-RSA*, 2020.

[37] 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.

[38] 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.

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

[40] Kim Laine. Simple Encrypted Arithmetic Library 2.3.1. *Microsoft Research TechReport*, 2017.

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

[42] 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.

[43] Markus Lorch, Jim Basney, and Dennis Kafura. A hardware-secured credential repository for Grid PKIs. *IEEE International Symposium on Cluster Computing and the Grid*, 2004.

[44] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On Ideal Lattices and Learning with Errors over Rings. *Journal of the ACM*, 2010.

[45] J. MacQueen. Some methods for classification and analysis of multivariate observations. *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, 1967.

[46] 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.

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

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

[49] 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.

[50] Mike Paterson and Larry J. Stockmeyer. On the Number of Nonscalar Multiplications Necessary to Evaluate Polynomials. *SIAM Journal on Computing*, 1973.

[51] Manas A. Pathak and Bhiksha Raj. Privacy-Preserving Speaker Verification and Identification Using Gaussian Mixture Models. *IEEE*, 2013.

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

[53] W. W. Royce. Managing the development of large software systems: concepts and techniques. *Proceedings of the 9th International Conference on Software Engineering*, 1987.

[54] Michael Rubinstein, Ariel Shamir, and Shai Avidan. Improved Seam Carving for Video Retargeting. *The ACM SIGGRAPH conference proceedings*, 2008.

[55] R.R. Schaller. Moore's law: past, present and future. *IEEE Spectrum*, 1997.

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

[57] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. *arXiv:1312.6034*, 2014.

[58] 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.

[59] 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.

[60] Steven Turner. Transport Layer Security. *IEEE Internet Security*, 2014.

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

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

[63] Greg Yeric. Moore's law at 50: Are we planning for retirement? *International Electron Devices Meeting*, 2015.

[64] Jingru Yi, Pengxiang Wu, Menglin Jiang, Qiaoying Huang, Daniel J. Hoeppner, and Dimitris N. Metaxas. Attentive neural cell instance segmentation. *Medical Image Analysis*, 2019.

[65] 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, $\bullet : \mathbb{G} \times \mathbb{G} \to \mathbb{G}$, such that the following properties hold:

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

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

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

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

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

Appendix B

Networking Addendum

B.1 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.

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

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

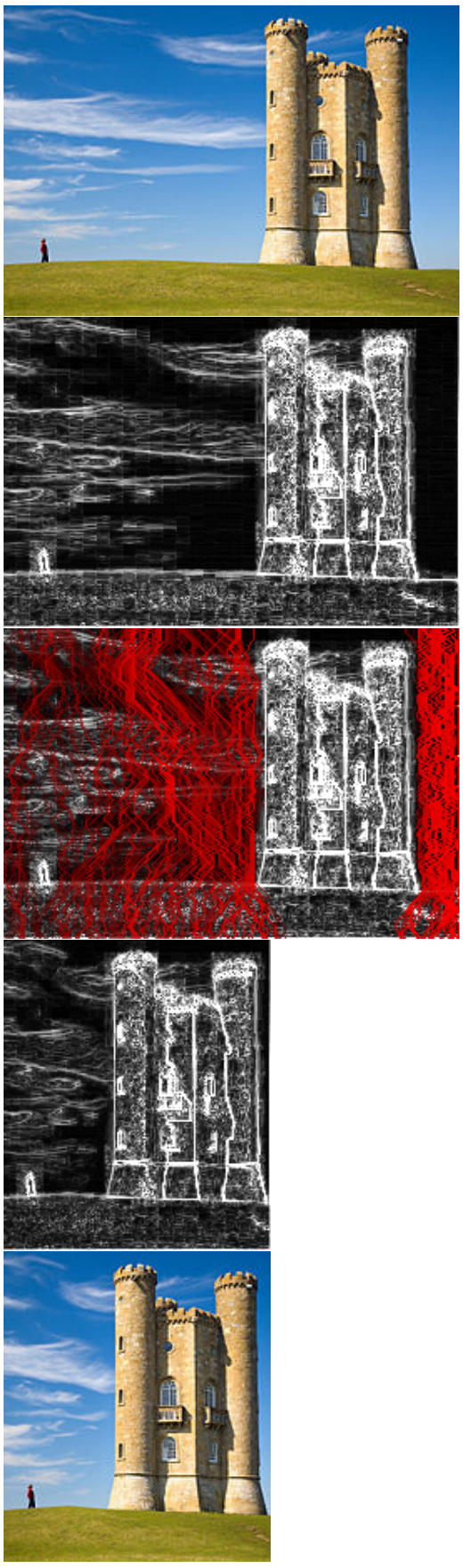
where $HoG(\mathbf{I}(x,y))$ is a histogram of oriented gradients at every pixel. The paper recommended an eight-bin histogram over an eleven-pixel square window around each pixel. Figure B.1b depicts the application of an energy function to an example image.

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.

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

$$M(i,j) = e(i,j) + \min\left(M(i-1,j-1), M(i-1,j), M(i-1,j+1)\right) \tag{B.4}$$

The figure below depicts each stage of the seam carving algorithm. Images are taken from [61].



(a) Original image.

(b) The energy of each pixel.

(c) Potential seams to delete.

(d) Low energy seams removed.

(e) Resulting image.

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

B.2 Sliding Window

A sliding window protocol is a common approach to achieving reliable, in-order delivery in packet-based data transmission protocols. Their main advantage over other techniques is that they can improve transmission efficiency in a channel with high latency. The first stage of the algorithm requires a buffer to be created storing all packets, and each packet to be assigned an identifier. Then, a window size, n, is defined. This value determines the number of packets that will be in flight between the sender and receiver simultaneously. Initially, the first n packets will be sent in a single batch.

Whenever a packet arrives at its destination, an acknowledgement containing the identifier of the packet is returned to the sender. This causes the sender to increment the sliding window, and the next packet in the buffer transmitted. If the acknowledgement is not received - possibly because the original packet was lost in transmission, or because the acknowledgement was - the sender will retransmit a packet after a timeout has expired. Two techniques for ensuring reliable transmission are included below.

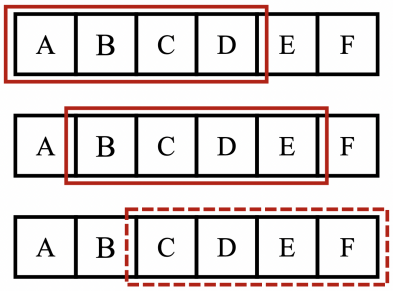
B.2.1 Go-Back-N ARQ

The Go-Back-N Automatic Repeat Request protocol recovers from packet loss or corruption by resending the target packet as well as all subsequent packets in the window. The N in the name of this protocol refers to the size of the window. This method minimises the complexity of the network, so is often preferred.

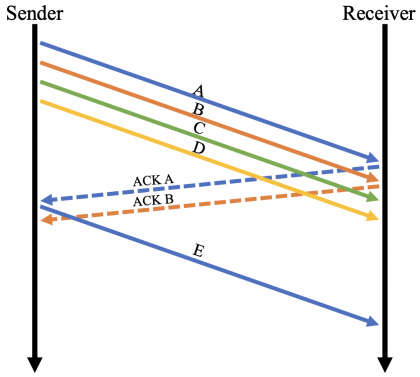
B.2.2 Selective Repeat ARQ

The Selective Repeat Automatic Repeat Request protocol recovers from packet loss or corruption by resending the target packet alone. This method is more appropriate if lots of loss is expected because the traffic on the network is minimised.

Figure B.2 summarises the sliding window approach to data transmission succinctly.



(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 for F to be sent (bottom).



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

Figure B.2: A high-level view of TCP's sliding window protocol.

Appendix C

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 makes 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 accuracy 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	*Preparatory work*. 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. *Introduction to Robotics Assignment 1 Deadline: 01/11/2021*
3	11/11/2021	24/11/2021	Implement homomorphic encryption algorithm. *Introduction to Robotics Assignment 2 Deadline: 22/11/2021*
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	*Buffer.*
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 disseration: Introduction and Implementation chapters, and share with supervisors. *Cybercrime 1 Deadline: 04/02/2022*
10	17/02/2022	02/03/2022	Evaluate two approaches by recording the accuracy, running time, and memory usage. *Cybercrime 2 Deadline: 18/02/2022*
11	03/03/2022	16/03/2022	*Buffer.* *Cybercrime 3 Deadline: 04/03/2022*
12	17/03/2022	30/03/2022	Write dissertation: Evaluation of approaches, and share with supervisors. Milestone: First full draft. *Cybercrime 4 Deadline: 18/03/2022*
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.