Secure License Plate Matching using Homomorphic Encryption

Thesis submitted in accordance with the requirements of

the Vrije Universiteit for the degree of Masters in Science

by

Archana Bindu Sunil

Parallel and Distributed Computer Systems

Supervisors:

Prof. Dr. Rob de Jeu (Vrije Universiteit)

Arjen Veenhuizen (TNO)

Dr. Ir. Thijs Veugen (TNO/TUDelft)

Asst. Prof. Dr. Zekeriya Erkin(TUDelft)

Second Reader:

Assoc. Prof. Dr. Thilo Kielmann (Vrije Universiteit)

July 2015

Abstract

License plate matching plays an important role in applications like law enforcement and road pricing where the plate is first recognized and then compared to a database of authorized vehicle plates. If an entity with a license plate wants to find a match in the license plates database which is in the possession of another entity; matching in the encrypted domain ensures that the privacy of data of both the involved entities is preserved. This work explores the field of license plate matching in a secure way so that the sensitive information associated with it is protected.

This thesis proposes a character recognition scheme which does not require any software, classifier or training data for identifying characters and yet yields a success rate of 98.5%. The recognized characters of the license plate is represented as an integer and the matching is performed on the encrypted integer values representing the number plates. Secure protocols using homomorphic encryption are designed for exact matching of the license plates. The matching procedure is error-free and the performance of matching for twenty one database sizes using two algorithms - Paillier encryption and extended Gentry's fully homomorphic system, are evaluated.

Acknowledgements

I would like to express my sincere gratitude to my supervisors Dr. Zekeriya Erkin and Dr. Thijs Veugen of TU Delft for accepting my candidature and for the insightful suggestions, guidance and support extended, without which this work would not have been completed.

The guidance and support I received from Dr. Rob De Jeu, internal supervisor and Dr.Thilo Kielmann of Vrije Universiteit (VU) Amsterdam is immense and their timely help has contributed to the completion of this thesis in right time.

My sincere thanks goes to Arjen Veenhuizen of TNO for providing me an opportunity to join their team as an intern. The helpful discussions and suggestions helped me in all the time of research and writing of this thesis. I extend my sincere gratitude to TNO and the members associated for the help I received through the internship in terms of technical guidance and otherwise.

My study in VU Amsterdam was a wonderful learning experience and was supported by Vrije Universiteit Fellowship Programme. I sincerely thank the faculty members of Parallel and Distributed Computer Systems (PDCS) and my friends for giving me this wonderful experience.

I would like to acknowledge Juan Ramón Troncoso Pastoriza for extending his help in completion of this thesis.

I would also like to thank Dr. Krishnasree Achuthan, Dr. Rafael Fourquet and Faculty of Cyber Security Systems and Networks department of Amrita University for the motivation and support provided to me.

Last but not the least, I would like to thank my family members for the unconditional love and support given to me.

Contents

Abstract ii

Acknowledgements iii

List of Tables vi

List of Figures vii

Listings viii

1 Introduction **1**

1.1 Scope of research 3

1.2 Outline of thesis 5

2 Related Work **6**

2.1 License plate character recognition 6

2.2 Secure processing 8

2.3 License plate matching 9

3 Preliminaries **11**

3.1 Cryptographic primitives 11

3.1.1 Paillier cryptosystem 12

3.1.2 Extended version of Gentry's fully homomorphic cryptosystem 13

3.2 Image processing primitives 16

3.2.1 Thresholding 16

3.2.2 Vertical projection histogram 17

3.2.3 Feature extraction 17

4 License Plate Matching **18**

4.1 Extraction of characters from plate 18

4.2 Exact number matching 21

4.2.1 Protocols using Paillier Encryption 24

4.2.2 Protocols using extended Gentry's fully homomorphic cryptosystem 30

5 Implementation **32**

5.1 Character recognition 32

5.1.1 Constraints 33

5.1.2 Process description 33

5.2 Secure matching 38

5.2.1 Paillier encryption 38

5.2.2 Extended Gentry's cryptosystem 40

6 Results and Discussions **42**

6.1 Character recognition 42

6.2 Secure integer matching 44

6.3 Limitations and future work 47

7 Conclusions **49**

A Sample database of license plate values **51**

Bibliography **53**

List of Tables

1.1 Primary purposes for license plate recognition implementation 2

6.1 Character recognition success rates of different algorithms 43

6.2 Execution time for matching data in encrypted domain 45

List of Figures

4.1 Processes for converting license plate image to character string 19

5.1 Example of a Dutch vehicle license plate 33
5.2 License plate image after thresholding 33
5.3 Range of continuous black pixels in thresholded image 34
5.4 Segmented character from range (32,81) 34
5.5 Blocks for character '3' for block-size 100 in x and y direction 34
5.6 Feature array for character '3' for block-size 100 in x and y direction . 35
5.7 The standard character set for our implementation 36
5.8 Character predictions for Fig 5.1 36
5.9 Integers representing the alphabets and the concatenated integer representing the characters of the license plate in Fig 5.1 37
5.10 Execution time of two test plate values for Paillier encryption 39
5.11 Execution time of two test plate values for extended Gentry's system 41

6.1 Graph for the execution time taken for secure matching 44

Listings

5.1 Conversion to feature array 35
5.2 Constructing the concatenated integer 37
5.3 Paillier encryption in C++ 38
5.4 Paillier encryption in Python 38
5.5 Encryption in C++ using extended Gentry's system 40

Chapter 1

Introduction

Every individual has an identity which distinguishes him from others. Similarly automobiles have license plates which identifies itself from others. Hence license plate recognition plays an important role in systems like traffic management, road pricing and for security and crime investigation. There is an increasing use of license plate recognition techniques by law enforcement agencies worldwide for tracking vehicles of interest like stolen and wanted vehicles. These technologies are also adopted for efficient traffic management and speed control purposes. Statistics from the Dutch Institute for Traffic Care (ITC) related to use of license plate recognition systems indicates that the number of speeding violations have dropped by 90% and average speed on the monitored stretches of road has dropped from 115km/h to 105km/h. A research report submitted to the U.S. Department of Justice [22] indicates that license plate recognition technologies are essential for law enforcement and public safety. Various law enforcement agencies throughout the United States were selected for the study and the results in Table 1.1 indicated that locating and recovering stolen vehicles was the primary purpose of license plate recognition implementation.

Primary purposes for ALPR	**No. of agencies**	**Percentage**
Auto theft	25	63%
Vehicle and traffic enforcement	11	28%
Investigations	10	25%
Identifying vehicles of interest	5	13%
Warrants	2	5%
Intelligence/homeland security	2	5%
Others	5	13%

Table 1.1: Primary purposes for license plate recognition implementation as per study in [22]

License plate recognition systems automates a tedious and manual process that is done at various organizations for different purposes like identifying owners of vehicles, tracking location of vehicles and people, for tracking stolen cars, for charging road tolls and to monitor border crossings. These systems improve the efficiency in identifying vehicles of interest and thus contributes to the economy. We see that a license plate itself may not be private but the use of the collected information can reveal sensitive information about people and vehicles and hence the security of this system and the privacy of the information has to be guaranteed for the system to be effectively used. This information can raise privacy issues related to location and identity of individuals.

The necessity for ensuring privacy of data has become stronger and this calls for secure encryption schemes to maintain confidentiality in applications. Guarding the vehicle number and details is important as privacy of people have to be respected and also this information could be used for criminal activities. By handling the data and processes in encrypted domain, the system becomes secure and robust.

Cryptography was initially related to concealing of messages to avoid interceptors from leaking any information. In recent decades, this field has expanded to include

secure computations, integrity checking, authentication, digital signatures and many others to it. With homomorphic encryption, computing on encrypted data is possible and this enables computations to be done in untrusted environments. The information is hidden from the service provider and still encrypted data can be processed by the service provider without having the keys. The results are deciphered by the user alone, who is in possession of the private keys. Over the years several homomorphic cryptosystems have been proposed. Some of the approaches for encrypted data processing are based on Paillier [19], Goldwasser-Micali [9], ElGamal [3] and RSA [21]. There are different schemes which utilize encrypted processing for applications including signal processing applications [16], cloud computing [25], recommender systems [4] and distributed applications [6].

1.1 Scope of research

The focus of this thesis is to develop a system for securely matching license plates so that the license plate and the information associated with it is protected. In this thesis we investigate scenarios involving multiple parties, for exact matching of licence plate numbers.

Consider a scenario which involves an office parking lot that has the database of employee and visitor vehicle license plates and a security agency with a license plate of a potential criminal's vehicle that they are trying to track. The security agency wants to find if that particular vehicle has entered the lot. But both entities would like to keep their data hidden from the other, yet perform the required processing to obtain the results. The license plate value, whose existence in the database is to be checked, is encrypted and sent to the office for the matching to be done. In such a setting, the office does not learn any information about the license plate they

receive from the agency as it is in encrypted format. The agency might get to know if the vehicle has entered the office parking lot or not but no details about the other license plates in the database are leaked. Thus the information belonging to both the entities are secured and the required operations are performed using homomorphic encryption.

The major contributions of this work include:

1. We present a character recognition system which could be easily translated to encrypted domain.
2. We compare the proposed character recognition algorithm with some of the state-of-the-art techniques and prove that it has high success rates and is comparable to the currently used techniques.
3. We design secure protocols for exact matching of license plates for three different scenarios using two different algorithms.
4. We implement the protocols and analyze the results and their performances are compared for license plate databases of twenty one different sizes.

A license plate can directly or indirectly reveal sensitive information and hence it is important to use secure methods when dealing with applications involving license plates. This work proposes a novel approach for representing the license plates and also for matching in a secure way. The character recognition system does not require any optical character recognition software or trainers but uses a standard set of characters for character prediction. The proposed character recognition system works with an accuracy of 98.5% and the execution times for the secure matching algorithms are recorded for twenty one database sizes.

1.2 Outline of thesis

The remaining chapters of this thesis are structured as follows:

Chapter 2 discusses the works related to secure processing and licence plate detection techniques. In this chapter, the state-of-the-art with respect to these fields are covered and this helps in understanding the need for a secure system for license plate matching.

Chapter 3 introduces the cryptographic primitives based on which the protocols described in the later chapters are developed. It also gives a brief description about the image processing techniques using which the character recognition algorithm works.

Chapter 4 presents the technique used for the extraction of characters from the license plate image and the algorithms for matching of the license plate characters. Even though numerous techniques have been developed for character recognition, they involve operations which are difficult to realize in the encrypted domain whereas the technique described in this work can be easily realized in the encrypted domain.

Chapter 5 deals with the implementation details of the character recognition and the secure matching sections of the system. The processes involved in the secure license plate matching system is explained step-by-step in this chapter.

Chapter 6 analyzes the results obtained from the study of the algorithms mentioned in the previous chapters. It also presents the limitations and future scope of this research line.

Chapter 7 discusses the conclusions that are drawn from this thesis.

Chapter 2

Related Work

Numerous techniques integrating cryptography and signal processing have been devised over the last few decades for the purpose of preserving privacy of data in communication and storage. This chapter deals with the different techniques used for license plate detection and character recognition. It also looks into processing done in encrypted domain.

2.1 License plate character recognition

In this section we discuss about the techniques currently being used for the license plate character segmentation and recognition. So far, character segmentation has been accomplished by techniques such as projection [30] [28], morphology [13] and connected components [26]. The character recognition techniques being used includes classifiers [23], artificial neural networks [12], support vector machine [28] and optical character recognition (OCR) software [31] [30].

A multi-national license plate recognition system where the license plate characters lie on a single row is discussed in [26]. A version of Fisher/Otsu algorithm is used

for discriminating the text pixels from background and character extraction is done using connected component labeling. For character recognition, an OCR using a combination of statistical and structural methods was used. The statistical part of OCR consists of extracting Fourier descriptors from the contours of the image and using Hierarchical Neural Network (HNN) for classification. A structural classifier using AdaBoost algorithm [24] is used to distinguish ambiguous characters. The results shows a success rate of about 95.2% for the license plate recognition.

A success rate of 98.7% is obtained in [31] by using blob extraction for distinguishing character blocks and ABBYY OCR software [1] for character recognition. K-means algorithm was used in [30] along with image binarization, vertical edge detection, horizontal and vertical image projections for character segmentation and Tesseract OCR software was used for character recognition. A character recognition success rate of 94.03% was obtained.

In [28] projection technique was used after tilt correction and image enhancement to segment the license plate into blocks and support vector machine (SVM) integration as a character recognition algorithm where SVM was employed as a classifier to recognize characters. An overall performance of 93.54% was achieved.

In [12] end and corner points are extracted from characters and a hybrid system made of neural networks, template matching and genetic algorithms are used for recognition. In this character segmentation consisted of noise reduction and image binarization using thresholding. For character recognition brightness and shape features were extracted. The results of character recognition for license plates had an accuracy of 95.2% using neural networks alone and 96.8% using hybrid system. Image scissoring for character segmentation and statistical feature extraction for character recognition yields about 82% accuracy for the work described in [15].

Character recognition of plates in [23] uses connected regions and bounding box drawn using minimum boundary rectangle for separating the characters and then a simple nearest neighbor classifier for classifying the separated characters. An offline training set was used to train the classifier and average detection accuracy was 94%.

According to [11], accuracy of character recognition relies on the accuracy of character segmentation and the work briefs about segmentation techniques used in other works. Blob extraction [29] has 97.2%, template matching [10] has 98.8%, combination of projection [20] has 97% and morphological and partition based method [13] has 94% accuracy. The study in [14] compares algorithms for license plate detection and recognition and shows that dynamic programming algorithm is the fastest and the Gabor transform has the highest accuracy. In dynamic programming-based algorithm, blob extraction is used to detect the alphanumeric character of the plate. Gabor transform method uses Gabor filters as license plate detectors and vector quantization for segmentation.

2.2 Secure processing

This section gives an overview about the works done in encrypted domain processing. Signal processing applications for privacy protection is discussed in [16]. The different techniques used for encrypted signal processing like homomorphic cryptosystem, two-party computation and garbled circuits (for secure function evaluation) are presented along with applications of these primitives such as face recognition, K-means clustering and fingerprinting.

The fingerprint-based authentication protocol presented in [2] uses fingercode for representing fingerprint images and finds all the identities in the database whose distance is under a given threshold to the user's fingercode. The construction

extensively uses homomorphic encryption for identification of matching identities. The distance between user and database fingerprints are computed utilizing the homomorphic properties of the Paillier cryptosystem whereas the protocol for finding the identities below the threshold uses additive ElGamal over Elliptic Curves as the underlying homomorphic encryption scheme.

Recommendations generated by encrypting private data and processing them is discussed in [5]. The active participation of the user is eliminated by introducing a semi-trusted third party and a comparison protocol for encrypted and packed data is presented. Data packing reduces computation and communication costs as multiple encrypted data elements can be compared in a single operation.

In [27] an extended version of Gentry's fully homomorphic scheme [8] was used to implement a face verification system working with encrypted feature vectors. The cryptosystem has the ability to homomorphically calculate low-degree polynomial functions and extends the plaintext size to allow homomorphic additions and multiplications. A Support Vector Machine (SVM) classifier is used for the verification algorithm. The non-interactive fully-private outsourced face verification system was evaluated in terms of time and communication complexities and the execution times are comparable to those obtained with a Paillier-based system.

2.3 License plate matching

In [17] the integration of license plate and face recognition for security during entry and exit in parking areas is evaluated. The matching is between a decrypted plate/face image from database and a plain test plate/face image. The face and car plate are stored in database after encrypting using Hill Cipher matrix manipulation and Fast Fourier Transform (FFT). The conversion from time to frequency domain using FFT

was performed to reduce the computational time for processing. For entry to parking the Peak to Side lobe Ratio (PSR) value of car plate is checked; the PSR value is obtained using Unconstrained minimum average correlation energy (UMACE) filters. PSR of plate and face are checked and the fusion should match for vehicle to exit. This technique had a total success rate of 96% during parking entry and 99.5% during parking exit.

License plate matching technique in [18] deals with matching plate readings captured by a dual setup of license plate readers. The matching utilizes edit distance technique of text mining which measures the closeness of two strings. The matching technique uses an association matrix which has the conditional probabilities of observing one character at one place for a given observed character at another place. The association matrix is estimated using a self-learning algorithm which improves and corrects itself over time. The results show that a system with two license plate reading units have matching rates in the range of 95-96% and false matching rates in the range of 1%.

From the existing works, we can see that cryptography is not used for license plate matching; cryptography and license plates recognition are combined in [17] but it is not used for matching purposes. There are no existing works which utilizes homomorphic encryption for license plate matching and we propose such a system through this thesis. The character recognition system is designed such that it does not require any classifiers or OCR software to recognize the characters.

Chapter 3

Preliminaries

The building blocks which help in the construction of the protocols for a secure license plate matching system are described in this chapter. It comprises of cryptographic algorithms and image processing techniques.

3.1 Cryptographic primitives

Homomorphic encryption enables computations to be performed on encrypted data. In this form of encryption, operations in plain domain have certain analogous operations in encrypted domain which would yield the same results, i.e. the operations when performed on ciphertext yields an encrypted result which when decrypted would match the result obtained by performing the analogous operations on the plaintext. There are partially homomorphic (allows some operations on the ciphertexts) and fully homomorphic (supports multiple types of operations on ciphertexts) systems. The partially homomorphic cryptosystems comprises of additively (Paillier [19], Goldwasser-Micali [9]) and multiplicatively (ElGamal [3],

RSA [21]) homomorphic schemes, among others. The first plausible construction of a fully homomorphic encryption scheme [7] was described by Craig Gentry in 2009.

This section deals with the important cryptographic algorithms, namely Paillier and a version of Gentry's fully homomorphic cryptosystems, that are used for designing the data matching systems explained in Chapter 4. These are algorithms for public key cryptography or asymmetric cryptography which requires two separate mathematically linked keys - private and public keys.

3.1.1 Paillier cryptosystem

It is a probabilistic algorithm for public key cryptography created by Pascal Paillier and its security is based on the difficulty of computing n^{th} residue classes. The decisional composite residuosity assumption (DCRA) is a mathematical assumption used in the proof of the Paillier cryptosystem and states that given a composite n and an integer z, it is hard to decide whether there exists y such that $z \equiv y^n \bmod n^2$.

Key Generation

1. Compute $n = pq$ where p and q are prime numbers of equal length.
2. Set $g = n + 1$, $\lambda = \phi(n)$ and $\mu = \phi(n)^{-1} \bmod n$ where $\phi(n) = (p-1)(q-1)$.

Public key pk = (n , g) and private/secret key sk = (λ, μ).

Encryption

1. Let m be the message where $m \in \mathbb{Z}_n$.
2. Select a random r where $r \in \mathbb{Z}_n^*$.
3. Compute ciphertext $c = g^m r^n \bmod n^2$.

Decryption

1. Ciphertext $c \in \mathbb{Z}_{n^2}^*$.
2. Compute plaintext message as $m = L(c^\lambda \bmod n^2) \cdot \mu \bmod n$, where $L(\alpha) = \frac{\alpha-1}{n}$.

$[[.]]$ is used to represent Paillier encrypted data.

Background

Certain discrete algorithms can be computed easily. By binomial theorem,

$(1+n)^w = \sum_{k=0}^{w} \binom{w}{k} n^k = 1 + nw + \binom{w}{2} n^2 + \binom{w}{3} n^3 + \cdots$

$(1+n)^w \equiv (1+nw) \bmod n^2$

If $v = (1+n)^w \bmod n^2$ then $w \equiv \frac{v-1}{n} \bmod n$

$L((1+n)^w \bmod n^2) \equiv w \bmod n$ where $L(\alpha) = \frac{\alpha-1}{n}$ and $w \in \mathbb{Z}_n$

Paillier cryptosystem is additively homomorphic. In this the decryption of product of two ciphertexts is the sum of the corresponding plaintexts.

$$D(E(m_1) \cdot E(m_2)) = m_1 + m_2$$

The decryption of a ciphertext raised to the power of a plaintext/constant is the product of the two values.

$$D(E(m_1)^{m_2}) = m_1 \cdot m_2 \text{ and } D(E(m_1)^k) = k \cdot m_1$$

3.1.2 Extended version of Gentry's fully homomorphic cryptosystem

The Goldreich-Goldwasser-Halevi (GGH) cryptosystem is an asymmetric cryptosystem based on lattices. A lattice is a discrete subgroup of $\mathbb{R}^n$, represented as the set of all integer linear combinations of some basis $B = (\vec{b_1}, \cdots, \vec{b_n}) \in \mathbb{R}^n$ of linearly independent vectors. The version of Gentry's bootstrappable fully homomorphic cryptosystem in [8] is GGH-type based on ideal lattices. In this, the secret key B_{sk}

is a good basis (large correction radius and solves certain instances of closest vector problem) whereas the public key B is a bad basis (small correction radius and solving closest vector problem is algorithmically hard) and is usually chosen as the Hermite Normal Form (HNF) of the lattice. The encryption and decryption are analogous to channel noise addition and error correction. The encryption of a message m consists of a correctable error vector that encodes m to a lattice point. The decryption depends on the error correcting capability and can be done by recovering the error vector. The homomorphic scheme in [8] uses a principal-ideal lattice and proposes to squash the decryption circuit to reach a full homomorphism. The extension described here is mentioned in [27] and has improved the cardinality of the plaintext. It utilizes the ability to execute low to medium-degree polynomials before the cipher gets corrupted so as to lose data.

Notations

Vectors are represented by bold letters. $\mathbf{a} = \langle a_0, \cdots, a_{n-1} \rangle$ and $a(x) = \sum_{i=0}^{n-1} a_i \cdot x^i$. For $\mathbf{c} \in \mathbb{R}^n$, $\mathbf{c}$ mod $\mathbf{B}$ is computed as $[\mathbf{c} \times \mathbf{B}^{-1}]_f \times \mathbf{B}$ where $[.]_f$ denotes the fractional part.

Key Generation

Consider a principal-ideal lattice J generated by a polynomial $v(x)$ in the ring of polynomials modulo $f_n(x) \doteq x^n + 1$. The HNF of lattice J is of the form:

$$HNF(J) = \begin{bmatrix} d & 0 & 0 & 0 \\ -r & 1 & 0 & 0 \\ -(r^2) \bmod d & 0 & 1 & 0 \\ & \ddots & & \\ -(r^{n-1}) \bmod d & 0 & 0 & 1 \end{bmatrix}$$

where d represents the resultant of polynomials $v(x)$ and $f_n(x)$ and r is a root of $f_n(x) \bmod d$. $\mathbf{B}$ is the public key encryption matrix and is determined by the integer pair (d, r). The secret key is the pair $(v(x), w(x))$ where $v(x) \times w(x) = d \mod f_n(x)$ but a single odd coefficient of $\mathbf{w}$ is sufficient.

Encryption

1. Let m be the message where $m \in \mathbb{Z}_{2^k}$ and $\mathbf{u}$ is a random noise vector with $\mathbf{u} \in \{0, \pm 1\}^n$.
2. $\mathbf{a} = 2^k\ \mathbf{u} + m \cdot \mathbf{e}_1$, where $\mathbf{e}_1$ is the first vector of canonical/standard basis.
3. Compute vector $\mathbf{c} = \mathbf{a} \mod \mathbf{B}$ where $\mathbf{c}$ has only one non-zero component which represents the encryption.

Decryption

For decryption, only one of the odd coefficients of $\mathbf{w} \mod d$ denoted by w_i is required. Compute plain text message as $m = ((c \cdot w_i) \mod d) w_i^{-1} \mod 2^k$.

[.] is used to represent data encrypted using extended Gentry's homomorphic system.

Background

In Gentry's system [8], the plaintext b is in $\mathbb{Z}_2$. $\mathbf{w} = \langle w_0, \cdots, w_{n-1} \rangle$.

$(\mathbf{c} \cdot \mathbf{w}) \mod d = (\mathbf{a} \cdot \mathbf{w}) \mod d = \mathbf{a} \cdot \mathbf{w}$ as all entries of $\mathbf{a} \cdot \mathbf{w}$ are smaller than $d/2$ in absolute value. We know $\mathbf{c} = \langle c, 0, \cdots, 0 \rangle$. So,

$$(\mathbf{c} \cdot \mathbf{w}) \mod d = (c \cdot \langle w_0, w_1, \cdots, w_{n-1} \rangle) \mod d$$
$$= \langle (c \cdot w_0) \mod d, (c \cdot w_1) \mod d, \cdots, (c \cdot w_{n-1}) \mod d \rangle.$$

Also, $(\mathbf{c} \cdot \mathbf{w}) \mod d = \mathbf{a} \cdot \mathbf{w} = 2\mathbf{u} \cdot \mathbf{w} + b\mathbf{e}_1 \cdot \mathbf{w} = 2\mathbf{u} \cdot \mathbf{w} + b \cdot \langle w_0, w_1, \cdots, w_{n-1} \rangle$.

We get that any decryptable ciphertext c must satisfy,

$\langle (c \cdot w_0) \mod d, (c \cdot w_1) \mod d, \cdots, (c \cdot w_{n-1}) \mod d \rangle = b \cdot \langle w_0, w_1, \cdots, w_{n-1} \rangle (\bmod 2)$

i.e. $(c \cdot w_i) \mod d = b \cdot w_i (\mod 2)$ and only one of the w_i's is sufficient to recover b as $(c \cdot w_i) \mod 2$.

3.2 Image processing primitives

Image processing refers to applying mathematical operations using signal processing methods for processing of images. The input is an image and the output after processing would be an image or some set of characteristics related to the image. Some techniques used in our algorithm, described in Chapter 4, are discussed here.

3.2.1 Thresholding

It is the simplest image segmentation method used to separate out regions of an image based on the pixel intensity. Each pixel's intensity is compared to a threshold value (T); if pixel value is greater than the threshold, it is assigned one value, else it is assigned another value. The thresholding operation can be expressed as:

$$\text{Value at } (x,y) \text{ after thresholding} = \begin{cases} value1, & \text{if value at } (x,y) > T \\ value2, & \text{otherwise} \end{cases}$$

There are different methods for thresholding, which include :

1. Global thresholding

   It is used to get a bi-level image and a global value is used as threshold value. This can also be used for removing noise from an image which includes filtering out very small or very large values.

2. Adaptive thresholding

   This method is used for images with varying illumination. The threshold value is calculated for small regions of the image so that different regions of the same image have different threshold values and it gives better results for images with different lighting conditions.

3. Otsu's thresholding

   This is used for bimodal images, whose image histogram has two peaks (foreground pixels and background pixels). It also uses a single threshold value for the image which is automatically calculated from the image histogram for the bimodal image.

3.2.2 Vertical projection histogram

A color histogram is a graphical representation of the distribution of the composition of colors in the image. It is produced by discretization of the colors in an image into a number of bins and then counting the number of pixels in each bin. By looking at the image histogram, the entire tonal distribution of the image can be determined. In a vertical projection histogram, the columns form the bins instead of the colour intensity. The columns are examined for determining the pixel distribution and this is used for distinguishing different regions of the image.

3.2.3 Feature extraction

The input image can be transformed to a reduced set of features that contain the relevant information from the input. The desired processing operations can be performed by using this reduced representation instead of the initial image. It can be related to dimensionality reduction. Feature extraction proves to be very useful in the area of optical character recognition. Image features like edges, corners, pixel variations, ridges and blobs are commonly used in feature extraction techniques.

Chapter 4

License Plate Matching

The secure license plate matching system comprises of two main phases - the extraction of characters from plate and the exact matching of the characters with a database of license plate characters.

4.1 Extraction of characters from plate

Thresholding is applied to the license plate image and the characters are segmented. Feature extraction is performed on the segmented characters by converting it to a one-dimensional array. The feature array of the character is compared to the feature arrays of the standard character set (0-9 and A-Z) and the character with the minimum difference to the standard character features is determined.

Fig. 4.1 depicts the processes for obtaining characters from the license plate. The initial step in extraction of characters is thresholding followed by removal of the borders. Vertical projection of the resulting image is taken to find the range in which consecutive black pixels (across width) occur. The black pixel count in each column

is used for segmentation of the plate image to cropped images of characters which are then transformed to a one-dimensional array.

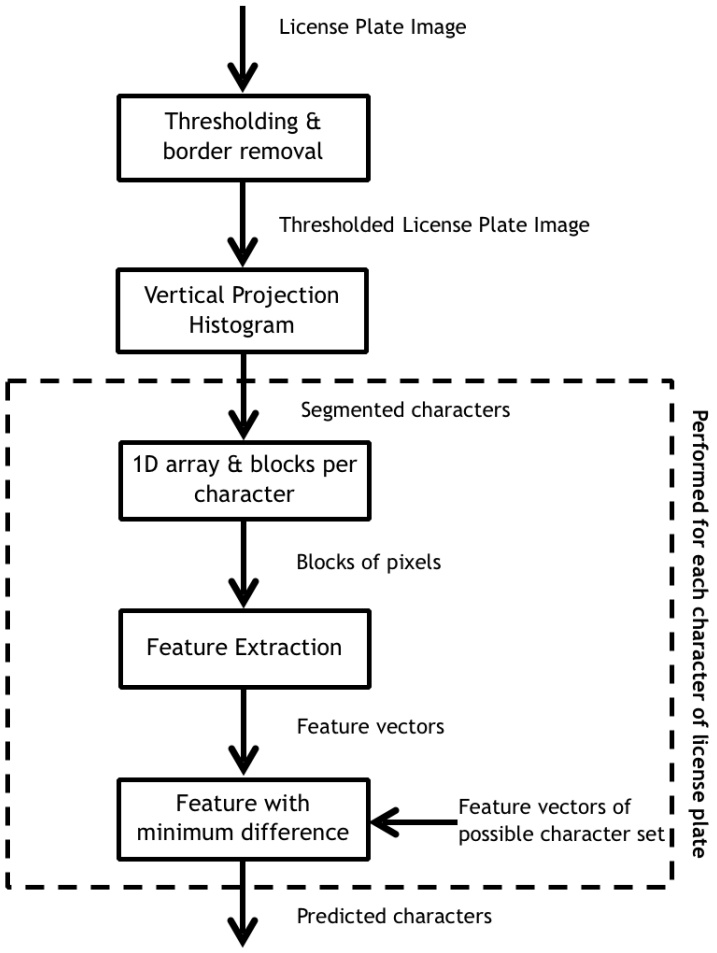


Figure 4.1: Processes for converting license plate image to character string

The one-dimensional array is divided into different blocks of equal size and the pixel concentration of each block is recorded. The resulting array is then transformed to an array of $+1$'s and -1's depending on :

$$\text{Value in the feature array at i} = \begin{cases} +1, & \text{if } value[i+1] < value[i] \\ -1, & \text{otherwise} \end{cases}$$

This feature array is compared to the feature arrays, obtained using the same method, for each character in the standard character set. The character yielding

the minimum difference with feature arrays is predicted as the character. To make the algorithm robust, predictions are made for different block sizes and the most common or maximum predicted character is considered as the desired character.

Let the feature of license plate characters be represented by $P = p_1, p_2, \ldots p_b$ and the feature for the standard array be represented by $Q = q_1, q_2, \ldots q_b$ where b represents the number of blocks into which the one-dimensional array is divided.

$$\text{Difference} = \sum_{i=1}^{b} (p_i - q_i)^2 \tag{4.1}$$

where the p_i's and q_i's are either $+1$ or -1.

The above mentioned steps of the algorithm are applied for each of the segmented characters yielding a list of characters corresponding to the characters in license plate. The character list thus obtained is first converted to integers by mapping A-Z to 10-35. Then the 6 integers representing the license plate characters are concatenated to form an integer. Each integer value is between 0 and 35 and hence require 6 bits for representation.

Let $a_5, a_4, a_3, a_2, a_1, a_0$ be the integers, then a is the concatenated integer.

$$a = \sum_{i=0}^{6} a_i \cdot 2^{6 \cdot i} \tag{4.2}$$

We present a new way to represent the license plate characters by using a single integer for all the characters. This could help in processing the plates for various other applications as the processing can be done on integers. The storage of license plates can also be done as integers and this would reduce the storage costs.

Here the character recognition is performed in plain domain. Character recognition from an image in encrypted domain would involve expensive operations and there is

no added advantage of performing the matching in encrypted domain on the images as to doing it on the characters because the entity already has the license plate data. Since the requirement is to know if there is a match for the license plate or not, it is sufficient that the processing be performed on the recognized characters of the license plate which are encrypted. However, the character recognition algorithm presented in this work could be easily translated to encrypted domain as it is primarily composed of linear operations. But the operations are expensive in the encrypted domain as it requires operations like integer comparisons for the feature array formation and this model does not have the advantage in performing those operations in encrypted domain.

However, the license plate information could be sensitive and hence the revelation of this information to the other entity has to be minimized during the process of matching. The license plate value that has to be searched in database as well as the database of license plates have to be protected. So the matching process is carried out in encrypted domain.

4.2 Exact number matching

This is performed in encrypted domain and involves operations on integers. The license plate for which the match has to be found is referred as test plate and it is matched against a database of license plates. The license plate characters are represented as integers and for matching, the difference between the integer corresponding to test plate and each entry in database (which is also an integer obtained using method mentioned in Section 4.1) is calculated. If there is a zero among the differences, it implies that the test plate value and one entry in database

are the same which means then there is a match. This matching is done in encrypted domain so that the information associated with the license plates are protected.

Notations

A : entity with license plate whose match has to be found (test plate).

x : integer representing test license plate characters.

B : entity with a database of license plates.

y_i : integers which represents the license plate characters for i = 0 ... s.

s : number of plates in database.

k : number of bits representing the license plate characters.

T : trusted third party.

pk : public key of cryptosystem.

sk : private key of cryptosystem.

t : bit used to indicate if there was a match or not.

$\sum$: sum of the elements.

$\prod$: product of the elements.

$[[var]]$: var encrypted using Paillier encryption.

$[var]$: var encrypted using extended Gentry's fully homomorphic system.

Three scenarios are considered for the protocol designs:

- **Scenario 1**

  Two entities **A** and **B** are involved. **A** has the public and private keys and sends the public key to **B** along with x in encrypted domain. The difference calculation is performed at **B** for each y_i in encrypted domain and the result is send in encrypted form to **A** who decrypts it with the private key to find if there was a match or not. The result, whether there is a match or not, is

known by **A** only. Here the database at **B** is stored in plaintext format as the encryption keys depend on **A** and it varies.

A can be a police agency whereas **B** the security agency for an organization who keeps track of all the vehicles entering and leaving the office premises.

- **Scenario 2**

  Two entities **A** and **B** are involved. **B** has the public and private keys and sends the public key and the y_i's in encrypted form to **A**. The difference calculation is carried out in encrypted domain at **A** and the encrypted results are send back to **B**. **B** decrypts the differences and finds if there is a match or not and send this to **A**. Here the database at **B** can be stored in encrypted format but the result is known to both entities. This is useful if the database has to be stored in encrypted form, preventing information leak from the database.

  Here also **A** can be a police agency and **B** the security agency for an organization who has data regarding all the vehicles in the office premises.

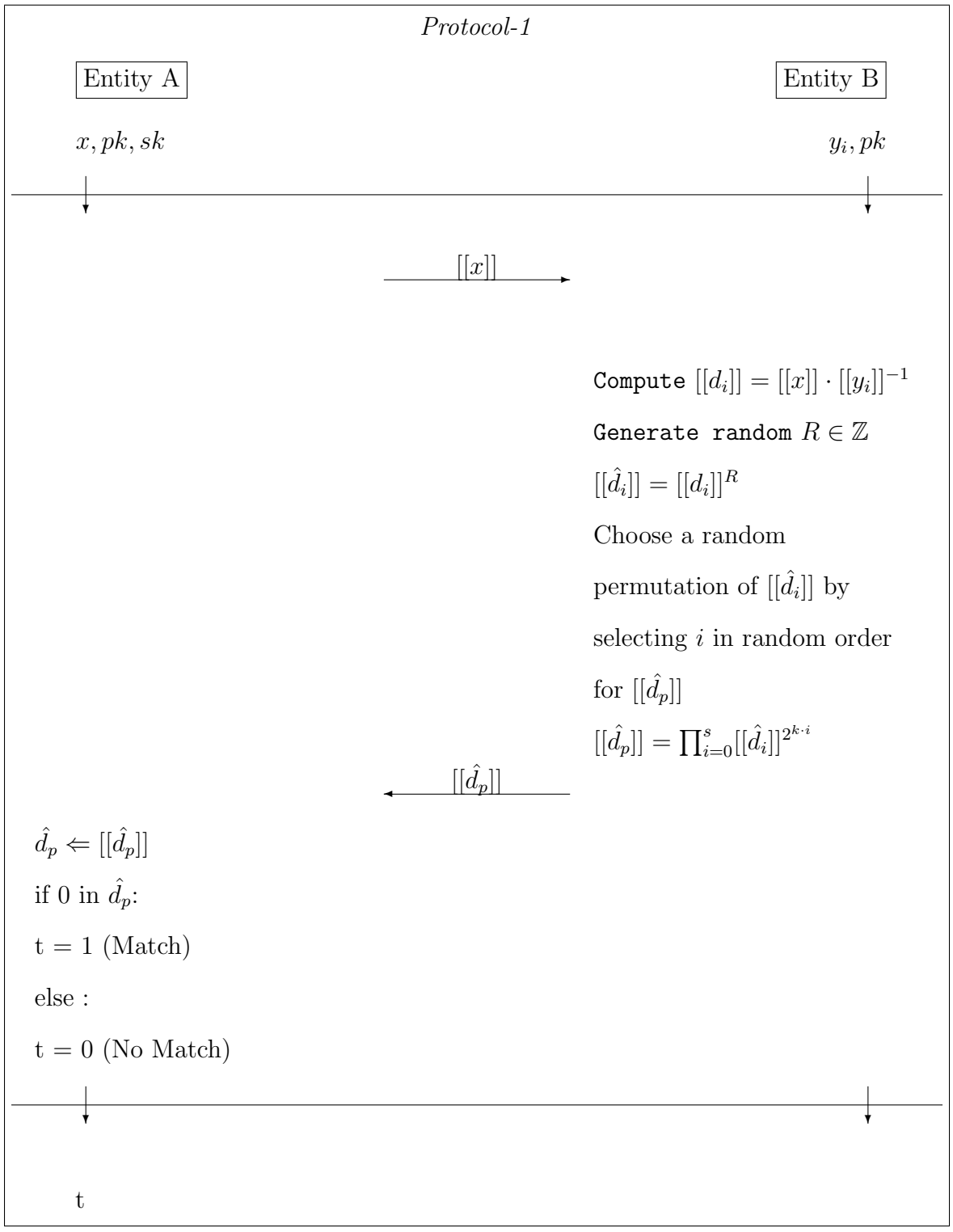
- **Scenario 3**

  Three entities **A**, **B** and **T** are involved. The public and private keys are at **T** and the public key is send to **A** and **B**. **A** sends x in encrypted form to **B**. The difference calculation is performed at **B** for each y_i and the results are send to **T** who has the private keys to decrypt the result. **T** sends to **A** if a match is found or not. Here the database is stored in encrypted format and the result is known to **A** and **T**.

  A can be the police agency whereas **B** the security agency to whom an organization has outsourced its security related works. **T** could be the organization who is trusted by both the security agency and the police.

4.2.1 Protocols using Paillier Encryption

Protocol 1 for scenario 1



In scenario 1, **A** sends $[[x]]$ encrypted using public key pk and pk to **B**. The difference of the test plate integer $[[x]]$ with each element $[[y_i]]$ at **B** is computed at **B** as

$$[[d_i]] = [[x - y_i]] = [[x]] \cdot [[y_i]]^{-1} \tag{4.3}$$

At **B**, the differences are blinded so that when the results are send to **A**, **A** does not learn any information about the data at **B** other than if there is a match or not.

$$[[\hat{d}_i]] = [[d_i \cdot R]] = [[d_i]]^R \tag{4.4}$$

where R is a random integer. Here multiplication with R is used for blinding of values. If there is a match the result is zero, which remains unaltered even when multiplied by any value whereas for other results, it yields a random value from which no information about the database entries can be inferred by **A**. For reducing communication costs, the $[[\hat{d}_i]]$ values are packed as follows before being sent to **A**

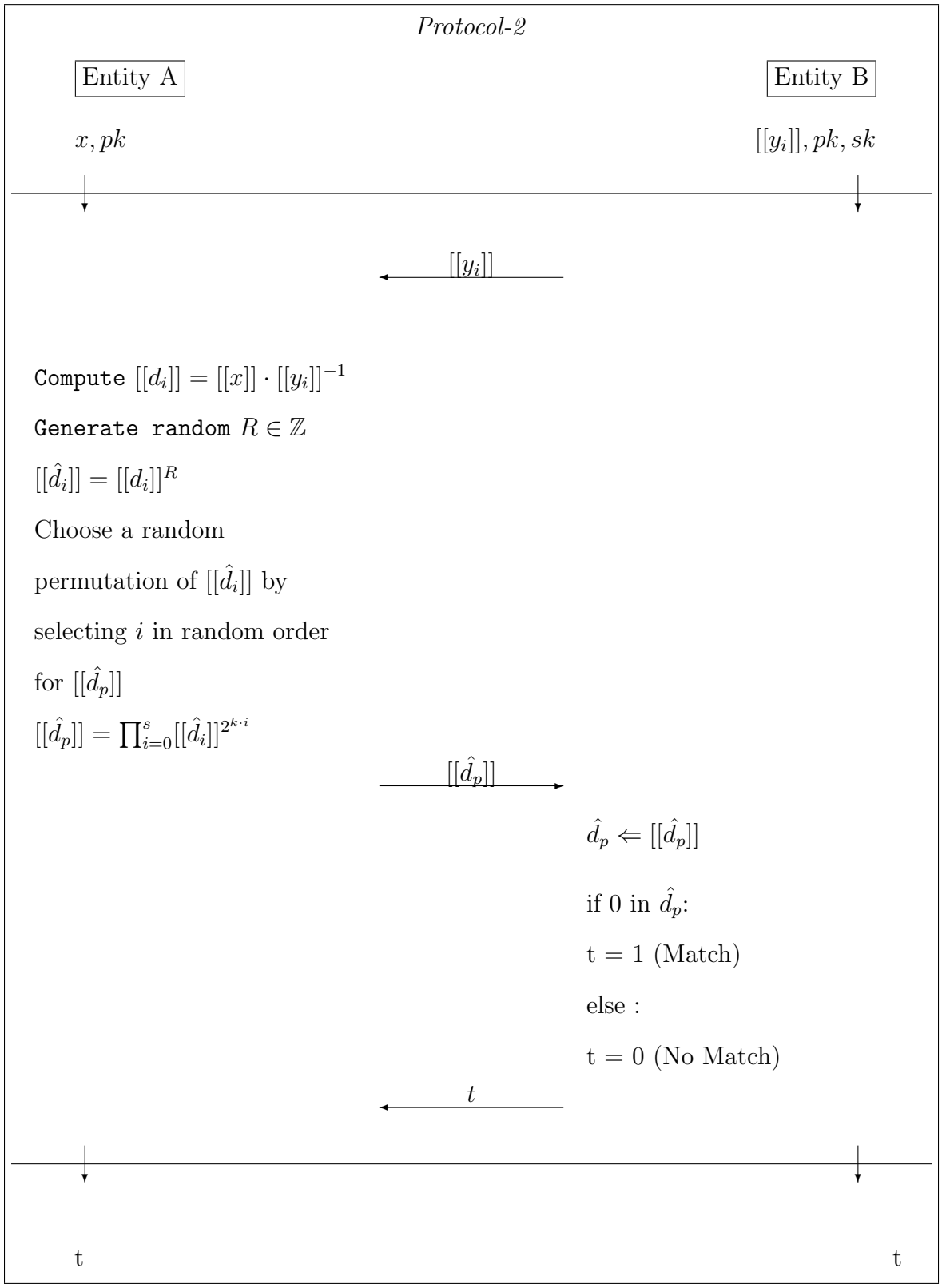
$$[[\hat{d}_p]] = [[\sum_{i=0}^{s} \hat{d}_i \cdot 2^{k \cdot i}]] = \prod_{i=0}^{s} [[\hat{d}_i]]^{2^{k \cdot i}} \tag{4.5}$$

where k is the number of bits representing the license plate characters.

The differences are randomized (before packing) so that if there is a match, one cannot predict which entry of the database was the match which would otherwise leak some information about the position of the test plate entry in the database. The randomization is performed by taking database entries in scattered order while packing (i.e. order of i in Equation 4.5).

The results are decrypted at **A** using the private keys sk and if there is a zero among the decrypted values, then **A** can conclude that there was a match for the test plate in the database. A bit t is used to indicate if there is a match or not; t is 1 when there is a match and 0 when not.

Protocol 2 for scenario 2



In scenario 2, **B** sends $[[y_i]]$ and pk to **A** and the difference between test plate integer value $[[x]]$ and database entries $[[y_i]]$ are computed and then blinded and packed at **A** as mentioned in Equations 4.3, 4.4 and 4.5.

The communication cost is high if the database entries are to be sent to **A** individually, so data packing can be done for the $[[y_i]]$ values before being sent to **A**. The $[[y_i]]$ values can be packed as :

$$[[y_p]] = [[\sum_{i=0}^{s} y_i \cdot 2^{k \cdot i}]] = \prod_{i=0}^{s} [[y_i]]^{2^{k \cdot i}} \tag{4.6}$$

where k is the number of bits representing the license plate characters.

For computing the difference using the packed $[[y_i]]$ values, $[[x]]$ is also packed similar to Equation 4.6 as :

$$[[x_p]] = \prod_{i=0}^{s} [[x]]^{2^{k \cdot i}} \tag{4.7}$$

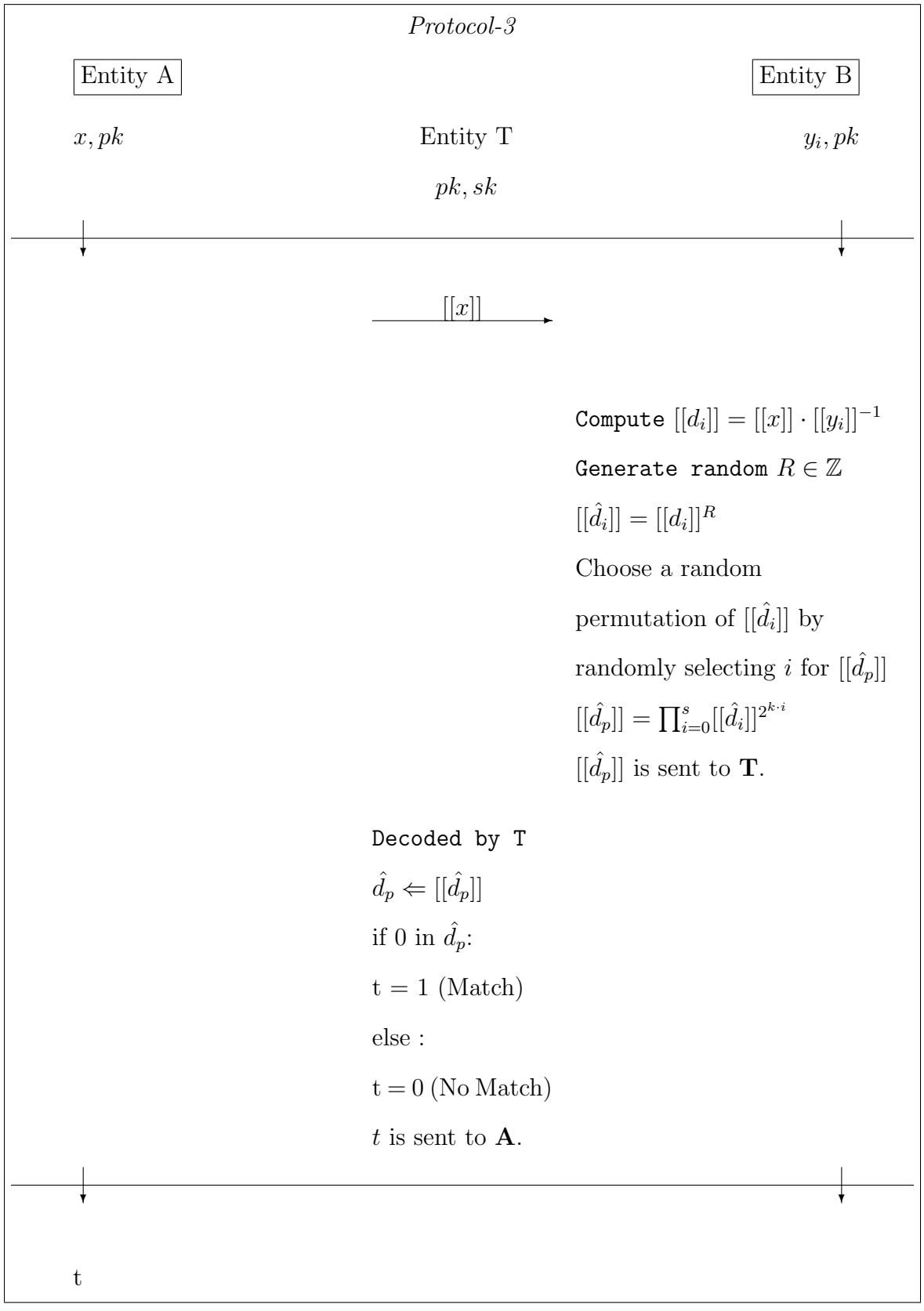
The difference between the packed $[[x]]$ and $[[y_i]]$ values can be calculated, with blinding using a random integer R, as :

$$\begin{aligned} [[\hat{d_p}]] &= [[\sum_{i=0}^{s} R \cdot (x_p - y_p) \cdot 2^{k \cdot i}]] \\ &= \prod_{i=0}^{s} ([[x_p]] \cdot [[y_p]]^{-1})^{R \cdot 2^{k \cdot i}} \end{aligned} \tag{4.8}$$

The differences are randomized to avoid information leak from the database, by mapping the position of the entry when there is a match, and this can be done by taking database entries in scattered order while packing (i.e. order of i in Equation 4.6).

In this, **A** and **B** knows if there is a match or not but the database at **B** is stored in encrypted format. It is also reasonable to allow the computation to be performed at **A** as it is **A** who wants to find if there is a match or not.

Protocol 3 for scenario 3



There are three entities involved in scenario 3 and it is an integration of scenarios 1 and 2. There is a third party **T** who has the public and private keys. The public key pk is provided to **B** who has the database stored in encrypted form.

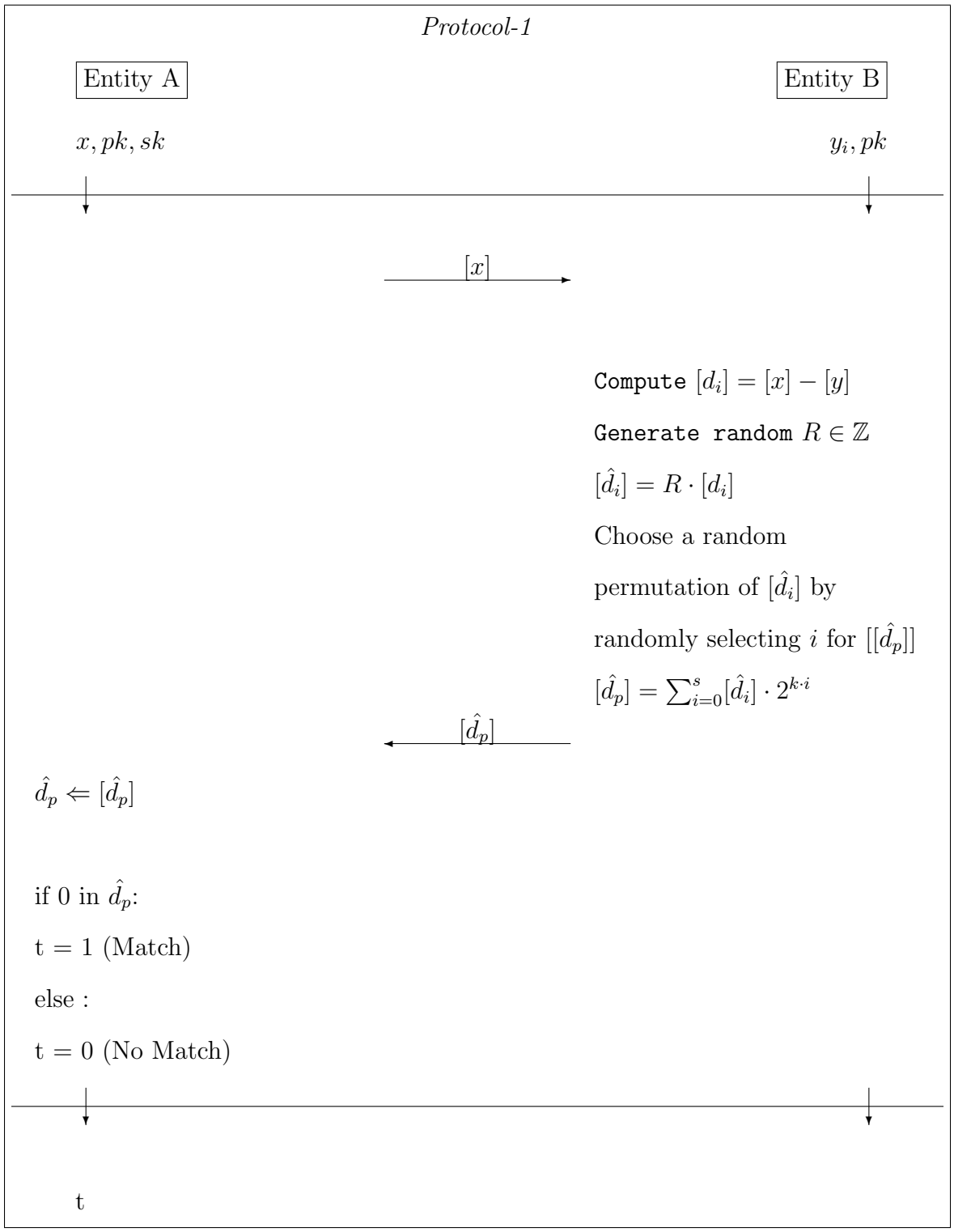
A has a test plate x, which is encrypted using pk and $[[x]]$ is send to **B**. At **B** the differences between $[[x]]$ and database entries $[[y_i]]$ are computed as in Equation 4.3 and then blinding and packing are performed as described by Equations 4.4 and 4.5. The values are randomized before packing so that even if there is a match, no information about the position of the entry in database can be known. Thus leakage of any information about the database is prevented.

The encrypted differences are sent to **T** where the decryption is done using the private keys. **T** determines whether there is a match or not depending on decrypted values. The presence of a zero in decrypted values indicates that a match was found. A bit t is sent to **A** to notify **A** about the results; if t is one there was a match, else no match.

Here both **T** and **A** are aware of the results i.e. if there is a match or not. In this scenario the communication cost is lesser than the one in scenario 2. The database is also stored in encrypted format but the computations are performed by **B** as in scenario 1.

This protocol can be deployed in a situation involving a police agency, acting as entity **A**, who wants to find if a vehicle was in the premises of an organization or not. The organization has outsourced its security to an agency, represented by **B**, who monitors the security details associated with the office including the vehicles in the parking lot. The organization is the third party **T** who is trusted by both police and the security agency.

4.2.2 Protocols using extended Gentry's fully homomorphic cryptosystem



Protocol 1 is for a scenario like scenario 1 of 4.2.1. **A** sends encrypted test plate value $[x]$ and public key pk to **B**. At **B**, the difference of the test plate value $[x]$ and the database entries $[y_i]$ is calculated in encrypted domain as

$$[d_i] = [x] - [y_i] \tag{4.9}$$

The differences are then blinded so that when the results are send to **A**, no information about the data at **B** other than if there is a match or not can be known.

$$[\hat{d}_i] = R \cdot [d_i] \tag{4.10}$$

where R is a random integer. For better communication, the $[\hat{d}_i]$ values are packed as follows before being sent to **A**

$$[\hat{d}_p] = \sum_{i=0}^{s} [\hat{d}_i] \cdot 2^{k \cdot i} \tag{4.11}$$

where k is the number of bits representing the license plate characters. Blinding does not affect zero as the operation is multiplication by R and thus if there is a match then one of the values would still be zero. The differences can be randomized by scattering the order while packing the difference values (i.e. order of i in Equation 4.11), which helps in masking the position of the entry in database, if a match is present.

The results are send to **A** where it is decrypted using the private keys sk. A bit t is used to indicate if there is a match or not; t is 1 when there is a match and 0 when not.

Chapter 5

Implementation

The implementation consists of two parts; a character recognition algorithm and a data matching algorithm. The character recognition algorithm was developed, in spite of numerous techniques already being used for this purpose, so that it is easily realizable in the encrypted domain. The data matching algorithm securely performs integer matching.

5.1 Character recognition

The character recognition algorithm was implemented in Python using OpenCV library. OpenCV is a free library aimed at real-time computer vision applications. It has C++, C, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. OpenCV-Python is the Python API of OpenCV, which is a Python wrapper around original C++ implementation, and the support of Numpy, a library for numerical operations, makes it easier to work with. OpenCV version 2.4.8 was used in Ubuntu 14.04.1 LTS platform for this implementation.

5.1.1 Constraints

Dutch license plates are considered for this implementation. They consist of a sequence of six characters composed of alphabets and digits. The current system uses black letters on a yellow background. The license plate characters are assumed to be in a single line. The input image should be a license plate image as this method is for recognizing the characters and does not extract license plate from a bigger image. The font of the license plate is assumed to be Gill Sans, one of the commonly used license plate fonts.

5.1.2 Process description

The input for character recognition is a license plate image of Fig. 5.1 format. The license plate is subject to thresholding and border removal, yielding an image as shown in Fig. 5.2. The global threshold value is set to 127.

Figure 5.1: Example of a Dutch vehicle license plate

Figure 5.2: License plate image after thresholding

The characters are segmented from the thresholded image by using vertical projection histogram method. Using this, the areas containing continuous black pixels are detected and those areas are cropped out. The Fig. 5.3 shows the range of continuous black pixels in Fig. 5.2. Eight range of values are detected, six license plate characters

and two hyphens, and two of them are eliminated depending on their heights as the hyphens occupy only a certain height unlike the characters.

```
OrderedDict([(32, 81), (100, 151), (164, 187), (202, 257),
(276, 331), (354, 407), (422, 445), (455, 510)])
```

Figure 5.3: Range of continuous black pixels in thresholded image

Each of the characters are cropped out using the ranges in Fig. 5.3 and they are converted to one dimensional arrays, both height-wise (y direction) and width-wise (x direction). The character obtained from range (32,81) is '3' as in Fig. 5.4.

Figure 5.4: Segmented character from range (32,81)

The one dimensional array of a character is converted to an array of blocks where each block holds the sum of pixel intensities of the pixels in that block. The array of blocks are obtained in both x direction and y direction. Fig. 5.5 shows the array of blocks for a block-size 100 for the character '3'.

```
Array of blocks for blocksize  100  in x direction are
[15000, 5662, 4436, 13283, 19103, 16844, 18363, 13549, 14591,
 14125, 16786, 16633, 19301, 15947, 6340, 2214, 4891, 15513]
Array of blocks for blocksize  100  in y direction are
[23396, 16126, 17371, 17481, 14999, 17673, 15800, 13377, 1288
6, 12577, 12116, 9865, 5271, 3439, 4102, 7094, 10750, 18258]
```

Figure 5.5: Blocks for character '3' for block-size 100 in x and y direction

The array of blocks are then converted to a feature array, as in Fig. 5.6, which consists of $+1$'s and -1's. If the value at position $i+1$ in array of blocks is less than the value at position i, then the value at i in feature array would be $+1$, otherwise -1.

```
def convert_block_to_feature(val1, val2):
    if val1 < val2:
        return +1
    return -1
```

Listing 5.1: Conversion to feature array

```
Feature array for blocksize  100  in x direction are
[1, 1, -1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, 1, 1, -1, -1]
Feature array for blocksize  100  in y direction are
[1, -1, -1, 1, -1, 1, 1, 1, 1, 1, 1, 1, 1, -1, -1, -1, -1]
```

Figure 5.6: Feature array for character '3' for block-size 100 in x and y direction

The feature array for the character '3' is compared with the feature arrays (obtained using same methods) of the set of possible or standard character set. The character in the standard character set which is most similar to the character which is being checked ('3' in this case), is predicted as the correct character. The most similar character is obtained by calculating the difference between the feature arrays of the character '3' and each character of the standard character set. The sum of the square of difference between each of the values in feature arrays for character '3' and each of the character in standard set are taken and the minimum value of the sum is considered for the minimum difference between feature arrays.

The Dutch license plate font utilizes Gill Sans as its basis, although with numerous modifications. For the standard character set, we have chosen the font Gill Sans assuming that all our license plates use this. The characters which form our standard character set are in Fig. 5.7.

This character prediction is done for each character using different block-sizes so that we get a more reliable prediction of characters. For each character we take the

Figure 5.7: The standard character set for our implementation

character which was predicted the maximum times (one predicted character for each block size) as the correct prediction. Twenty six characters are predicted for each character, as there are 13 block sizes for x direction and y direction each.

BLOCK SIZES = [5, 10, 15, 20, 25, 30, 40, 45, 50, 60, 75, 90, 100]

The above mentioned steps are carried out for each character from the license plate. Fig. 5.8 shows the predicted characters for the license plate in Fig. 5.1 and the list of recognized characters.

```
Counter({'three.jpg': 17, 'eight.jpg': 4, 'B.jpg': 2, 'G.jpg'
: 1, 'two.jpg': 1, 'A.jpg': 1})
Counter({'five.jpg': 19, 'S.jpg': 3, 'six.jpg': 2, 'G.jpg': 1
, 'X.jpg': 1})
Counter({'G.jpg': 25, 'zero.jpg': 1})
Counter({'N.jpg': 21, 'M.jpg': 4, 'G.jpg': 1})
Counter({'H.jpg': 7, 'P.jpg': 4, 'N.jpg': 3, 'T.jpg': 2, 'W.j
pg': 2, 'J.jpg': 2, 'zero.jpg': 1, 'four.jpg': 1, 'I.jpg': 1,
 'B.jpg': 1, 'five.jpg': 1, 'Y.jpg': 1})
Counter({'four.jpg': 23, 'six.jpg': 1, 'M.jpg': 1, 'A.jpg': 1
})

Identified characters of license plate
['three', 'five', 'G', 'N', 'H', 'four']
```

Figure 5.8: Character predictions for Fig 5.1

The identified alphabets are then converted to integers between 10 and 35, yielding to a list of integers which are concatenated as per Equation 4.2 to produce an integer

representing the license plate characters. Fig. 5.9 shows the integer, which is the concatenated value used to represent the license plate in Fig. 5.1.

```
# Convert alphabets to integers between 10 and 35
char_to_int_map = {key:value for (key,value) in
                zip(string.ascii_uppercase, xrange(10,36))}

#if value is an alphabet get the corresponding integer value
#else get the integer as such from the character string
char_int = map(lambda x : char_to_int_map[x]
               if x in string.ascii_uppercase
               else value[x], char_string)
# Concatenate integers for characters to an integer (requires 6 bits each)
concatenated_value = sum([2**(i*6) * each for each,i in
                zip(char_int, xrange(len(char_int)-1,-1,-1))])
```

Listing 5.2: Constructing the concatenated integer

The concatenated integer, as shown in Fig. 5.9, is obtained for each license plate entry in database.

```
Identified characters of license plate
['three', 'five', 'G', 'N', 'H', 'four']
Integers representing characters
[3, 5, 16, 23, 17, 4]
Concatenated integer
3309401156
```

Figure 5.9: Integers representing the alphabets and the concatenated integer representing the characters of the license plate in Fig 5.1

The whole process of character recognition is in Python and takes about 21 seconds for converting one image to a concatenated integer value. The sample database for 88 entries is in Appendix A.

5.2 Secure matching

The matching of the license plate characters to a database of license plate characters are done in encrypted domain using homomorphic encryption. We have implemented using two algorithms - Paillier encryption and extended Gentry's method.

5.2.1 Paillier encryption

We implemented the Paillier encryption for integer matching in Python and C++. In C++, the Secure Computation Library(SeComLib) is used for realizing cryptographic schemes in encrypted domain. It supports Windows and Linux platforms. It is written in C++ and uses the GMP and MPIR libraries for big integer arithmetic.

The input to the secure number matching system using Paillier encryption is the concatenated integer representing the license plate characters, which is the output of processes described in Section 5.1.2. The keys are generated with a key size of 1024 bits and the concatenated integer is encrypted using the public keys.

```
Paillier::Ciphertext cipher_test =
          publicCryptoProvider.EncryptInteger(test_value);
```

Listing 5.3: Paillier encryption in C++

```
  cipher_test = paillier.encrypt(pub, test_value)
```

Listing 5.4: Paillier encryption in Python

The functions 'EncryptInteger' and 'encrypt', in listings 5.3 and 5.4, are defined to encrypt the value *test_value* as per Paillier algorithm using the public key.

The encrypted test plate value, which is the encrypted concatenated integer, is compared to the database of license plate values to find if there is a match or not.

This is done in encrypted domain i.e. the equality check of data (as described in Section 4.2) is performed on the ciphertexts representing the test value and each of the database entries. The encrypted differences are then multiplied with a random value so as to blind them. The encrypted values are then decrypted and checked for the presence of a zero, which in turn indicates the presence of the test plate value in the database.

In Fig. 5.10, the test plate value is 24887451920 and is searched in database with 500 and 5000 entries. The results show that the test plate value was found in the first database with 500 entries but was not found in the second one with 5000 entries.

```
> ./test_paillier 24887451920 db_500.txt
Entering Encrypted domain : Paillier Encryption
__Checking in Database__

Match Found !
Time taken for key generation 0.427479 secs
Time taken for one encryption 0.000004 secs
Time taken for finding match 0.676882 secs
>
> ./test_paillier 24887451920 db_5000.txt
Entering Encrypted domain : Paillier Encryption
__Checking in Database__

Match Not Found !
Time taken for key generation 0.459667 secs
Time taken for one encryption 0.000005 secs
Time taken for finding match 6.886805 secs
>
```

Figure 5.10: Execution time of two test plate values for Paillier encryption

The time taken for Paillier key generation in C++ is about 0.4 seconds and for one encryption it takes about 5 microseconds. The difference computations for finding match is performed in encrypted domain. The time required for matching depends on the size of the database; we test with database of sizes 88, 250, 500, 750, 1000, 1250, 1500, 1750, 2000, 2250, 2500, 2750, 3000, 3250, 3500, 3750, 4000, 4250, 4500, 4750 and 5000 and time varies between 0.1 to 7 seconds.

5.2.2 Extended Gentry's cryptosystem

Extended version of Gentry's cryptosystem as described in [27] was implemented in C++, for data matching purposes. It uses the latest versions of GMP and NTL libraries for arithmetic operations.

The input for the secure integer matching using extended Gentry's system is the concatenated integer obtained from character recognition phase described in Section 5.1.2. The key size is 1024 and the concatenated integers are encrypted for computations using the public key.

```
crypto->Encrypt(encTest, test_value);
```

Listing 5.5: Encryption in C++ using extended Gentry's system

The *test_value* is encrypted and stored in a variable *encTest* as in Listing 5.5. The encrypted test plate value is compared to each encrypted entry in database i.e. their difference is calculated. The encrypted differences are then blinded by multiplying with a random number. When decrypted, if any of the differences yields a zero, it indicates that there was a match in the database for the test plate which resulted in the zero.

Fig. 5.11 shows the time required for finding a match, in encrypted domain, using extended Gentry's method. The test plate value is 24887451920 and is searched in databases with 500 and 5000 entries. A match is successfully found in the database with 500 entries whereas no match was found in the second database with 5000 entries. The time required for key generation in this system is about 1 second and it takes about 1 millisecond for encryption of one integer. The time required for finding a match depends on the number of entries in the database; we use databases with 88,

250, 500, 750, 1000, 1250, 1500, 1750, 2000, 2250, 2500, 2750, 3000, 3250, 3500, 3750, 4000, 4250, 4500, 4750 and 5000 entries and the time varies between 1.5 to 90 seconds.

```
> ./test_gentry 24887451920 db_500.txt                ~
Entering Encrypted domain : extended Gentry's system
__Checking in Database__

Match Found !
Time taken for key generation 0.977817 secs
Time taken for one encryption 0.000557 secs
Time taken for finding match 8.980774 secs
>                                                     ~
> ./test_gentry 24887451920 db_5000.txt               ~
Entering Encrypted domain : extended Gentry's system
__Checking in Database__

Match Not Found !
Time taken for key generation 0.983162 secs
Time taken for one encryption 0.001299 secs
Time taken for finding match 90.365850 secs
```

Figure 5.11: Execution time of two test plate values for extended Gentry's system

The execution time for extended Gentry's system is more when compared to the Paillier system, this is because the Paillier system is additively homomorphic whereas the extended Gentry's system is designed to have fully homomorphic capabilities.

Chapter 6

Results and Discussions

In this chapter, we analyze the results obtained by implementing the character recognition and secure matching algorithms. We also look into the limitations and possible extensions for this work.

6.1 Character recognition

The character recognition scheme described in Section 4.1, was implemented in Python and it works with an overall accuracy of 98.5%. Out of the 528 characters tested 520 were identified correctly. The characters B and H were the most mistaken characters as per the analysis of the results. But even if one of the characters of the license plate is wrongly identified, the recognition of the license plate, as a whole, is considered wrong. From the 88 license plates tested, 81 were identified correctly yielding an accuracy of 92% for the recognition of characters of the plates. For the construction of feature array from array of blocks of pixels, we consider block sizes 5, 10, 15, 20, 25, 30, 40, 45, 50, 60, 75, 90 and 100. The absence of block size 5 in the array of blocks reduces the success rate of license plate character recognition to 85%.

The time taken for the generation of integers representing the character string of a license plate from a license plate image is around 21 seconds.

Table 6.1 shows the success rate of our algorithm and some other character recognition algorithms. The success rates show that our algorithm has rates comparable to many of the state-of-the-art techniques used for character recognition. In spite of techniques existing for character identification, this was developed so as to avoid the use of any classifiers and non-linear methods.

Reference	**Method**	**Success rate (%)**
[15]	Statistical feature extraction	85%
[12]	Template matching	90.3%
[28]	SVM Integration with feature extraction	93.7%
[26]	Hierarchical Neural Network(HNN)	95.2%
[12]	Genetic algorithm	96.8%
[31]	ABBYY OCR software	98.7%
Proposed	Feature extraction and minimum difference	98.5%

Table 6.1: Character recognition success rates of different algorithms

The method described in [31] has success rates closest to our implementation but the first one uses an open source optical character recognition (OCR) engine, ABBYY, whereas our algorithm uses mainly arithmetic operations and integer comparisons to recognize characters. So it is easy to translate the algorithm to encrypted domain, if needed, even though our implementation is in plain domain. The proposed method does not require support from training data set, classifiers or character recognition engines for recognizing the characters. This makes it possible to be implemented in encrypted domain using homomorphic encryption.

6.2 Secure integer matching

We implemented The algorithms for integer matching in encrypted domain using Paillier encryption and extended Gentry's system in C++ and the comparison of execution time for secure matching are represented as a graph of the execution time vs database sizes in Fig. 6.1 and as a table in Table 6.2. The matching is performed with databases of size 88, 250, 500, 750, 1000, 1250, 1500, 1750, 2000, 2250, 2500, 2750, 3000, 3250, 3500, 3750, 4000, 4250, 4500, 4750 and 5000. In the graph in Fig. 6.1 representing execution time for matching of integers in encrypted domain, the x-axis is the size of the database which is the number of entries in database and the y-axis is the time taken (in seconds) for matching of data in encrypted domain and finding the result. The graph shows a linear trend and the time required for matching increases as the database size increases.

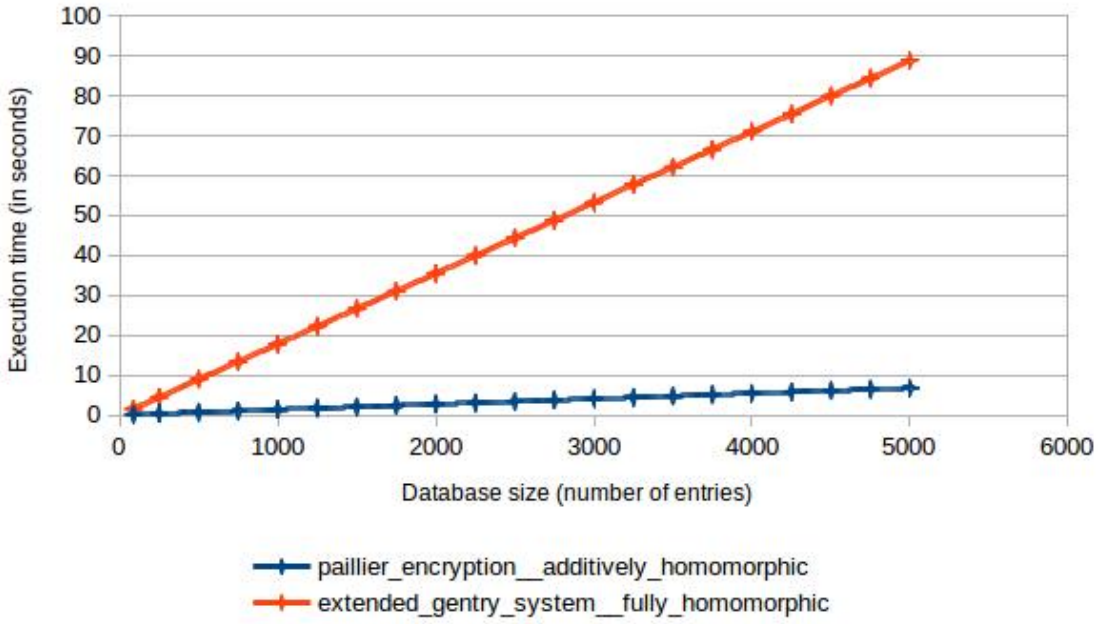


Figure 6.1: Graph for the execution time taken for secure matching whose values are given in Table 6.2

Number of database entries	Time in seconds for data matching	
	Paillier encryption	Extended Gentry system
88	0.1188844286	1.5669965238
250	0.3435913333	4.4371881905
500	0.6800360952	8.9294647619
750	1.0152990952	13.3617146667
1000	1.3579642857	17.809368381
1250	1.7114976191	22.2216005238
1500	2.034182	26.6457921429
1750	2.3653290476	31.0814909048
2000	2.719012	35.511724619
2250	3.0455483333	39.9713572381
2500	3.3895809524	44.3662747619
2750	3.7145151429	48.8084137143
3000	4.069070381	53.2607063333
3250	4.3893354762	57.7946869524
3500	4.7279144286	62.1409449048
3750	5.0778222381	66.6039687143
4000	5.4306632857	71.0456799524
4250	5.7507589048	75.4335374286
4500	6.0678429048	80.0278768571
4750	6.4301996667	84.4231748571
5000	6.7568149048	88.9000360476

Table 6.2: Execution time for matching data in encrypted domain

The time for each method described in Table 6.2 is an average of times for 20 executions taken as

$$\texttt{Average time} = \frac{execution\_time_1 + execution\_time_2 + \cdots + execution\_time_{20}}{20}$$

Twenty values are searched in each database size for a match and the time taken for matching is recorded and the average of the twenty values are calculated. These times are excluding the time required for key generation and encryption of data.

The execution time depends on the size of the database; for Paillier encryption it is between 0.12 to 6.75 seconds whereas for extended Gentry's system it is between 1.57 to 88.90 seconds. The database sizes are taken from 250 to 5000 in intervals of 250 and there is a database with 88 entries, which was obtained from testing the character recognition of license plates. The time taken by extended Gentry's system is more as it is a fully homomorphic cryptosystem whereas Paillier is an additively homomorphic cryptosystem. The fully homomorphic system can handle multiple operations (addition and multiplication) and computations without decryption error but this implementation of integer matching does not make complete use of the fully homomorphic properties. Even though Paillier performs faster for this implementation, extended Gentry's method could be used for extending the encrypted domain applications to the image level.

The algorithm using Paillier encryption for integer matching was implemented in Python also, but C++ performs much faster than its Python counterpart. It takes 0.5 seconds for key generation in C++ version whereas for Python it takes about 2 seconds. Hence the performance comparisons of secure matching of data for the two algorithms, Paillier encryption and extended Gentry's fully homomorphic cryptosystem, for various database sizes is done for C++ implementations.

The success rate of matching in encrypted domain using Paillier encryption and extended Gentry's fully homomorphic cryptosystem is 100% and hence the percentage of the whole system depends on successfully recognizing the license plate characters.

6.3 Limitations and future work

The limitations of this implementation in the initial character recognition phase include the input image constraints. The input should be a Dutch license plate and the image should not contain any other background. The font for character recognition is assumed to be Gill Sans which is a commonly used font for license plates. The accuracy can be increased further by devising methods for distinguishing characters with similar structure, for example B and 8. One way is to get the pixel count of the border line of each image, as B would have more black pixels on its left border when compared to 8. Also the plate is assumed to have the characters in a single line rather than two lines. The character recognition related processes are performed in plain domain and hence the implementation was in Python so as to speed up the coding time, using different libraries like OpenCV, Numpy and collections. The processing in encrypted domain is faster in C++ and hence the secure matching is implemented in C++ using SeComLib library. The output from Python section is passed as input to the C++ section of the system.

A future direction for this work is to implement the database in encrypted form and try to reduce the communication and storage costs. One of the ways is using data packing. The protocols described in Section 4.2 have mentioned data packing but the implementation does not cover this aspect. Data packing would help to reduce computational as well as communication cost between different parties involved in the system. It would be interesting to investigate the scenario with a trusted third party also. Algorithms using other homomorphic systems for secure matching and different equality check methods could be designed and compared with the proposed methods.

Another promising idea is to perform the character recognition also in encrypted domain. The algorithm in Section 4.1 is designed such that it can be easily realized

in the encrypted domain as it involves arithmetic operations and integer comparisons only. The fully homomorphic properties can be used to increase security and realize character extraction also in encrypted domain. The image can be encrypted and then the operations be performed on the encrypted data. This was not employed here as it would not add significant security to the scenario under consideration and would increase the computations and time also.

The input to the proposed character recognition system is a license plate image, this could be extended so that the input is an image from where the license plate region is detected for character identification. The license plates are assumed to have characters lying in a single row, it could be extended to include two row license plates also. This can be obtained by using the projection method to find the two rows.

Chapter 7

Conclusions

With the increasing number of privacy and security threats, it is important to ensure that there is minimal information leak. There are advanced techniques in use for license plate recognition and lots of research is being done on secure processing but not much combining both. In [17] both the aspects are combined but the matching is not done in encrypted domain. This work aims to integrate both the fields and to the best of my knowledge, is a first of its kind. We consider a scenario where there are two entities who wants to keep their data private and perform the license plate matching. Entity **A** has the test license plate and entity **B** has a database of license plates. **A** does not want **B** to know any details of the license plate in its possession but wants to find if the test license plate is a part of the database or not. **B** does not want **A** to know about any other database entries other than if the test plate value is in it or not. This is realized by implementing the license plate matching using homomorphic encryption.

A practical application of this scenario is when police wants to track a car and have the license plate, which is the test plate. They want to determine if it has entered an organization and the security agency of that organization has all the license plates of

vehicles that entered their premises. Both the entities want their data to be protected, yet perform the computations so as to track the vehicle.

Our implementation is a novel approach which includes a character recognition system for Dutch license plates and a secure matching algorithm. We obtain 92% accuracy for plate recognition and 98.5% accuracy for character recognition and the algorithm can be easily translated to the encrypted domain. The existing techniques for character recognition, with high success rates, are not easily realizable in encrypted domain. We use a new representation format for license plate characters, where one integer represents a license plate from which the characters can be extracted. The secure matching algorithms using Paillier encryption and extended Gentry's algorithm are error-free. The time taken for integer matching using Paillier encryption varies from 0.12 to 6.75 seconds for database size 88 to 5000 whereas the execution time using extended Gentry's scheme is between 1.57 and 88.90 seconds for database sizes between 88 and 5000.

Our work is one step towards preserving the privacy of license plates and is a promising area for research. License plate matching systems play a crucial role in road pricing, traffic management and law enforcement applications. The results help in choosing an efficient protocol for matching depending on the requirements of the system and application.

Appendix A

Sample database of license plate values

Shown below is the database containing 88 integers representing license plate characters. This was used for the data matching algorithms and the values were obtained from the character recognition of license plate images described in Chapter 4.

8646887252	6467966672	3377821151	5391087317
4333065413	6464255197	27960071505	6497043667
4319430854	9820771585	8745506523	3327460548
8748135428	2240149077	7659652373	40720328
2218111841	5510117060	1099265485	8647951447
8682526025	109892559	7621186897	4451088582
6597981761	8660567635	7538357829	7673304072
1232197603	2223880390	4300276545	7604606914
9668228935	6602164177	4333114630	1116018521
2307720481	2272086111	38647041	7558993733

5407586503	6480507923	2238552009	7555023560
8747062467	5509092367	9718322000	2236680003
8748636124	6497081416	2236437987	9718314568
22923207	33764000260	32773846093	9787228368
6449616975	2203390675	27401138840	2271540680
1232409877	3309401156	28155417867	8614417735
28058141634	21017281	19394748570	5411505859
1179502789	6567343391	5426205765	8646903563
7587825737	9720706754	3362912323	4450243867
4434212295	6583338461	7641282132	7624263112
1195492041	5393176795	4367627081	7573698754
6564427721	5476817089	8679945680	7656552261

Bibliography

[1] `http://www.abbyy.com/`.

[2] M. Barni, T. Bianchi, D. Catalano, M. Di Raimondo, R. D. Labati, and P. Failla. Privacy-preserving fingercode authentication. In *Proceedings of the 12th ACM workshop on Multimedia and security*, pages 231–240. ACM, 2010.

[3] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in cryptology*, pages 10–18. Springer, 1985.

[4] Z. Erkin, M. Beye, T. Veugen, and R. L. Lagendijk. Privacy-preserving content-based recommender system. In *Proceedings of the on Multimedia and security*, pages 77–84. ACM, 2012.

[5] Z. Erkin, T. Veugen, T. Toft, and R. L. Lagendijk. Generating private recommendations efficiently using homomorphic encryption and data packing. *Information Forensics and Security, IEEE Transactions on*, 7(3):1053–1066, 2012.

[6] Z. Erkin, T. Veugen, T. Toft, and R. L. Lagendijk. Privacy-preserving distributed clustering. *EURASIP Journal on Information Security*, 2013(1):1–15, 2013.

[7] C. Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. `crypto.stanford.edu/craig`.

[8] C. Gentry and S. Halevi. Implementing Gentry's fully-homomorphic encryption scheme. In *Advances in Cryptology–EUROCRYPT 2011*, pages 129–148. Springer, 2011.

[9] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences, 28*, pages 270–297, 1984.

[10] D. Hongyao and S. Xiuli. License plate characters segmentation using projection and template matching. In *Information Technology and Computer Science (ITCS), International Conference on. Vol. 1. IEEE*, pages 534–537, 2009.

[11] V. Karthikeyan, R. Sindhu, K. Anusha, and D. S. Vijith. Vehicle License Plate Character Segmentation-A STUDY. *International Journal of Computer and Electronics Research*, 2(1), 2013.

[12] S. Karungaru, K. Terada, and M. Fukumi. Character Recognition from Virtual Scenes and Vehicle License Plates Using Genetic Algorithms and Neural Networks. In *INTECH Open Access Publisher*, 2012.

[13] S. H. M. Kasaei and S. M. M. Kasaei. Extraction and recognition of the vehicle license plate for passing under outside environment. In *Intelligence and Security Informatics Conference (EISIC), 2011 European, IEEE*, pages 234–237, 2011.

[14] H. S. Kolour and A. Shahbahrami. An Evaluation of License Plate Recognition Algorithms. In *International Journal of Digital Information and Wireless Communications (IJDIWC)*, pages 247–253, 2011.

[15] P. Kulkarni, A. Khatri, and P. Banga. Automatic Number Plate Recognition (ANPR). In *RADIOELEKTRONIKA 19th International Conference*, pages 243–250, 2009.

[16] R. L. Lagendijk, Z. Erkin, and M. Barni. Encrypted signal processing for privacy protection: Conveying the utility of homomorphic encryption and multiparty computation. *Signal Processing Magazine, IEEE*, 30(1):82–105, 2013.

[17] S. S. M. Noor and N. M. Tahir. Fusion of License Plate and Face Recognition for Secure Parking. *Jurnal Teknologi*, 61(1), 2013.

[18] F. M. Oliveira-Neto, Lee D Han, and M. K. Jeong. An Online Self-Learning Algorithm for License Plate Matching. *Intelligent Transportation Systems, IEEE Transactions on*, 14(4):1806–1816, 2013.

[19] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in cryptology-EUROCRYPT'99*, pages 223–238. Springer, 1999.

[20] S. Qiao, Y. Zhu, X. Li, T. Liu, and B. Zhang. Research of improving the accuracy of license plate character segmentation. In *Frontier of Computer Science and Technology (FCST), 2010 Fifth International Conference IEEE*, pages 489–493, 2010.

[21] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM, 21(2)*, pages 120–126, 1978.

[22] D. J. Roberts and M. Casanova. Automated License Plate Recognition (ALPR) Use by Law Enforcement: Policy and Operational Guide, Summary. Technical report, Document 239605, IACP Technology Center, International Association of Chiefs of Police, September 2012.

[23] M. S. Sarfraz, A. Shahzad, M. A. Elahi, M. Fraz, I. Zafar, and E. A. Edirisinghe. Real-time automatic license plate recognition for CCTV forensic applications. In *Journal of real-time image processing*, pages 285–295, 2013.

[24] R. E. Schapire. The boosting approach to machine learning: An overview. In *Nonlinear estimation and classification*, pages 149–171. Springer, 2003.

[25] M. Tebaa, S. El Hajji, and A. El Ghazi. Homomorphic encryption applied to the cloud computing security. In *Proceedings of the World Congress on Engineering*, volume 1, pages 4–6, 2012.

[26] N. Thome, A. Vacavant, L. Robinault, and S. Miguet. A cognitive and video-based approach for multinational License Plate Recognition. In *Machine Vision and Applications, Springer-Verlag*, pages 389–407, 2011.

[27] J.R. Troncoso-Pastoriza, D. Gonzalez-Jimenez, and F. Perez-Gonzalez. Fully Private Noninteractive Face Verification. *Information Forensics and Security, IEEE Transactions on*, 8(7):1101–1114, 2013.

[28] Y. Wen, Y. Lu, J. Yan, Z. Zhou, K. M. Von Deneen, and P. Shi. An Algorithm for License Plate recognition Applied to Intelligent Transportation System. In *Intelligent Transportation Systems, IEEE Transactions on 12*, pages 830–845, 2011.

[29] Y. Yoon, H. Yoon K.D. Ban, and J. Kim. Blob extraction based character segmentation method for automatic license plate recognition system. In *Systems, Man, and Cybernetics (SMC), 2011 IEEE International Conference on IEEE*, pages 2192–2196, 2011.

[30] L. Zheng and X. He. Character segmentation for license plate recognition by K-means algorithm. In *Image Analysis and Processing-ICIAP 2011*, pages 444–453. Springer, 2011.

[31] L. Zheng, X. He, B. Samali, and L. T. Yang. Accuracy enhancement for license plate recognition. In *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on*, pages 511–516. IEEE, 2010.