UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Hendrik Eerikson

Privacy Preserving Fingerprint Identification

Bachelor's Thesis (9 ECTS)

Supervisors: Riivo Talviste, PhD
Kristjan Krips, MSc

Tartu 2020

Privacy Preserving Fingerprint Identification

Abstract:
Privacy preserving technologies are used to create applications for computing on sensitive data without compromising on the secrecy of said data.

In this thesis, secret sharing based multi-party computation is used to identify a fingerprint sample amongst a database of templates while preserving the secrecy of the sample and the templates. The FingerCode representation of fingerprints is used.

Privacy preserving fingerprint identification mitigates some of the privacy and security risks in biometric identification systems.

The secret sharing based fingerprint identification application developed in this thesis is more performant than a previous homomorphic encryption based one. Methodology for identifying fingerprints and programming privacy preserving applications using multi-party computation is given.

Fingerprint-based identification systems are vital tools for border control and law enforcement. Privacy preserving fingerprint identification could be used to prevent leakage and abuse of fingerprint data.

Keywords:
Fingerprint recognition, multi-party computation, biometrics

CERCS: P170

Privaatsust säilitav sõrmejälje tuvastus

Lühikokkuvõte:
Privaatsust säilitavad tehnoloogid võimaldavad arvutada sensitiivsete andmete peal selliselt, et andmete salastatus püsib.

Käesolevas töös kasutatakse ühissalastamisel põhinevat turvalist ühisarvutust, et tuvastada sõrmejälje proovi omanik võrreldes seda märgistatud sõrmejälgedega. Sõrmejälgede kujutamiseks on kasutatud sõrme tekstuuril põhinevat *FingerCode* representatsiooni.

Privaatsust säilitav sõrmejälje tuvastus leevendab osasid biomeetrilise tuvastuse süsteemide privaatsuse ja turvalisuse ohte.

Selle töö raames arendatud ühissalastusel põhinev sõrmejälje tuvastuse programm on kiirem kui varasem homomorfsel krüpteerimisel põhinev lahendus. Töös on esitatud metoodika sõrmejälgede tuvastamiseks ja turvalise ühisarvutusega privaatsust säilitavate programmide loomiseks.

Sõrmejälje tuvastuse süsteemid on laialdaselt kasututud piirikontrollis ja õiguskaitses. Privaatsust säilitav sõrmejälje tuvastus võib ennetada isikuandmete lekkimist ja kuritarvitamist.

Võtmesõnad:
Sõrmejälje tuvastus, turvaline ühisarvutus, biomeetria

CERCS: P170

Contents

Introduction

A security measure which is too cumbersome to use offers no protection. For this reason security engineers are constantly looking for more convenient methods for user verification. Biometrics-based verification has stood out as one such method. It does not require the user to remember a complicated secret or keep an access token with them. Instead, the identifying feature is some physiological or behavioural attribute of the person. Because of the ease-of-use, biometrics-based verification is used for border control, building access, smart device login, blood donor identification and many other purposes.

However, biometric identifiers have a critical shortcoming: unlike passwords or access tokens, they cannot be revoked. Therefore, a leaked biometric can be used to track a user or to gain unauthorized access. Storing unaltered biometric data in a database is a prime target for security breaches. In 2015, 5.6 million fingerprints were stolen from the U.S. government [Ale15]. A number of different attack vectors on biometrics-based verification have been documented [Rob07].

With these attacks in mind, an increased research effort has gone into developing biometrics-based systems that provide stronger security guarantees. One of the methods to achieve this is privacy preserving biometrics, which have had over a decade of development. Privacy preserving biometrics allow the owner of a biometric sample and the owner of biometric templates to find out whether they match while keeping their inputs secret.

In this thesis, a program for privacy preserving fingerprint identification is developed and an overview of the methodology is given. Using secret sharing based multi-party computation, a number of privacy concerns are mitigated: the developed program ensures that the user's biometric sample remains secret from the service hosts; the result of a query is only revealed to the provider of a fingerprint sample; the enrolled users' templates remain hidden from the service hosts. An arbitrary number of fingerprint template holders can upload their data to the service without having to participate in computation while also not revealing their data to any one party. The developed program improves on the fingerprint recognition speed of Barni *et al.* [Bar+10].

1 Preliminaries

1.1 Biometric Authentication and Identification

Authentication refers to asking the question "Is this person who they claim to be?". To authenticate themselves, a user states who they are using an identification number, login name, or the like and a verifier checks whether this statement is true.

Identification refers to asking the question "Who is this person?". An identification system conducts a one-to-many search to find a matching template to a person's identifier.

Biometric Authentication In recent years, biometrics-based authentication has become increasingly popular as a method of access control. Biometrics authentication systems (BAS) have been included in many smartphones, tablets and personal computers. The adoption of BAS has been encouraged by its speed, reliability and ease of use.

A typical BAS uses sensors for scanning the fingerprint, iris or face. Once a sensor captures the biometric data of an authorized user, access can be granted. A recognized person's biometric data has to be added to the BAS before they can be authenticated by it.

Roger Grimes [Gri19] emphasizes a number of security concerns regarding biometric authentication. A person's fingerprint or iris pattern cannot be changed. Thus, biometric data is not revocable. Once it has been leaked, it can be used for authentication attacks without any way of disarming the attacker. Therefore, biometric authentication should only be used in conjunction with additional security requirements; for example, requiring the possession of the device for which authentication provides access, such as a smartphone. Using BAS for remote authentication is generally not recommended.

Biometric Identification Biometric identification is widely used in law enforcement and border security. Fingerprint-based identification is prevalent owing to their high degree of uniqueness and the ease of scanning fingerprints. However, several privacy issues arise [PPJ03]. Since biometric data is not a secret to only its owner and it is not revocable, a query made into one biometric identification database could be used to query any other database. Because of the biological origin of biometric data, it can also be used to infer additional personal information.

International law enforcement organizations use large aggregate databases for biometrics [Int; Cim19]. Being able to identify biometric samples across national borders is crucial for fighting international crime [Lem10]. While international biometric databases

are useful for law enforcement, they come with escalated security and privacy concerns [CJ19].

It is desirable to preserve individuals' privacy and mitigate security risks while not compromising law enforcement. A biometric identification system should only reveal the information that is required and disallow any computation on the private data that has not been agreed upon earlier. The hosts of this system should not have privileged rights to the data.

Secure multi-party computation can be used to build a fingerprint identification system that eliminates some of these privacy concerns. One participant of the privacy-preserving fingerprint identification system cannot reveal the biometric data of the enrolled persons without colluding with other participants. The queries made to the system and their results will not be revealed to the participants. Therefore, a person cannot be tracked based on the queries made about them.

Using multi-party computation, there can be more than one party providing biometric templates to identify against. The hosts of the computation do not see the templates or the query samples. Therefore, external parties can add their biometric databases to the pool of templates without revealing anything about the enrolled persons. In this way, the heightened risk of a super database is reduced while still allowing for identification queries against the largest possible data set.

1.2 Related Work

As biometric authentication and identification systems have become more widespread, there has been an increased effort to deliver schemes which offer protection against unauthorized access and leakage of private data.

There are many different approaches to minimizing privacy risks and attack surfaces associated with biometric identification. These fall into two categories, protection in storage and protection during computation. Some techniques for protection during storage include fuzzy vaults [UPJ05] and cancelable biometrics [PRC15]. Biometric recognition schemes which offer protection during storage make concessions in terms of accuracy and are subject to attacks such as the false-accept attack [BCP13].

To protect the biometric data during processing, privacy preserving function evaluation is needed. For biometric recognition, the most common setting for privacy preserving function evaluation is 2-party multi-party computation with semi-honest adversary. This setting most closely mirrors a real world deployment with one party holding a biometric sample and the other having a template to match this sample against.

The earliest privacy preserving biometric recognition schemes used homomorphic encryption. A paper by Barni *et al.* proposes a protocol for secure fingerprint authentication using homomorphic encryption and filterbank based vectorised fingerprint representation [Bar+10].

An overview of different approaches for privacy preserving biometric authentication is given by Bringer *et al.*[BCP13].

1.3 Multi-Party Computation

Multi-party computation (MPC) is a method for securely evaluating a function. A number of parties use their inputs to jointly compute a function. In different settings, the benchmark for security is established differently, but in general it is required that no party can guess any other party's inputs with a better than chance accuracy. More lenient security settings allow for faster computation; stricter security settings usually come with a performance decrease.

MPC is useful when a number of parties don't want to reveal their confidential data but computing a useful statistic would require sharing said data with some other party. MPC is used to remove the need for a trusted third party that performs the computation. Recently MPC has developed from a purely theoretical research field into a multitude of practical deployments. An overview of numerous MPC applications is given by Archer *et al.* [Arc+18].

Sugar Beet Auction The classic example of a problem where MPC can be used is the Danish sugar beet auction [Bog+09]. In this scenario, a large number of sugar beet farmers and a single sugar producer want to jointly compute the market clearing price (CP), the price per unit at which total supply equals total demand. The computation is as follows: for every possible price point, each bidder enters the amount that they are willing to buy or sell. Then the CP is computed by finding the price point where the farmers are willing to sell just as much as the sugar producer is willing to buy. Afterwards, the sellers who entered a non-zero amount to sell at the CP, can sell that amount to the buyer.

Each participant wants to keep their bids secret as to not sell themselves short. Trusting that the buyer doesn't see any of the bids can also affect how the sellers bid. Therefore, the single buyer is not a suitable party to host this auction. In the referenced paper, MPC was used to eliminate the need for a trusted auctioneer.

Formalization Secure multi-party computation (MPC) comprises of multiple cryptographic protocols for a set of parties $\mathrm{P}_1, \ldots, \mathrm{P}_n$ to compute a function f on their private inputs $x_1, \ldots, x_n$ such that nothing besides the output $f(x_1, \ldots, x_n)$ is revealed. A corrupt participant should not be able to learn anything about the input of any other party except for what is leaked by the output.

MPC protocols differ greatly in the number of parties, the security model and the types of computation available. In this thesis, MPC based on secret sharing in the *three-party passively secure* model with an *honest majority* is used. Passive security means that all of the three parties follow the protocol exactly but will utilise any opportunity to learn about the other parties' input. This setting is often referred to as *honest-but-curious*. In the three party setting, an honest majority means that the input privacy is guaranteed as long as no two parties out of three collude with each other.

This security model was chosen as it provides a good balance of input security, speed of computation and availability of higher level protocols such as integer comparison. Protocols that are secure against a dishonest majority are computationally much more expensive. Actively secure protocols, where the adversary might not follow the protocol description, are also generally slower as they use preprocessing for correlated randomness [KPR18] or post-verification [Pan17].

Performance of MPC Compared to conventional computation, evaluating functions securely using MPC comes with a very large performance decrease. A single comparison of two integers takes approximately 1 ms on SHAREMIND MPC in the *three-party passive* model [Cyb19a]. In conventional computation, this operation takes a single CPU clock cycle. That is a slow down of around 10^6 times.

Generally, the performance bottleneck in multi-party computation is network latency and network bandwidth [Reb12]. MPC applications are optimized to reduce the amount of time spent waiting for a network message to be sent or received in two ways: on the protocol level and on the program level.

MPC protocols are optimized to minimize the number of times a party has to send a message to another party and wait for a reply, or in other words to minimize the number of communication rounds. The protocols are designed to compute things in advance and combine messages so as to send fewer larger messages instead of many smaller messages.

The programmer of MPC applications can reduce communication rounds by parallelizing their algorithms. Instead of sequentially computing on individual pieces of data,

the program should compute on vectors in a *single-instruction-multiple-data* (SIMD) style. This way the MPC protocols can be executed in parallel and the network messages of each execution can be combined. The number of operations computed per second using MPC increases drastically by computing in parallel.

The MPC protocols for computing different operations vary widely in their number of communication rounds. The relative slowdown of an MPC application depends on how vectorised the program is, the type of protocols used, the network configuration and the hardware used.

In the following section, examples of multi-party computation protocols are given.

1.4 Secret Sharing

For MPC to work, the inputs have to be somehow hidden such that computations can still be performed on them. One of the ways to achieve this is through secret sharing. Secret sharing refers to methods that divide an input value into shares and distribute them amongst participants such that the input can be reconstructed only with a sufficient number of shares and any smaller number of shares infer nothing about the input.

An algorithm $\mathbf{S}$ defines a *k-out-of-n* secret sharing scheme if it computes $\mathbf{S}(x) = [x_1, \ldots, x_n]$ (each x_i is called a *share*) and the following is fulfilled [Sha79]:

1. x is uniquely determined by any k shares from $x_1, \ldots, x_n$ and there exists an algorithm $\mathbf{S}'$ which efficiently computes x from these k shares.

2. Any $k - 1$ shares infer no information about x.

A brief overview of additive secret sharing is given along with descriptions of some basic operations on secret shared values. These protocols and bitwise secret sharing are used to construct more complicated protocols such as integer comparison. In this work, the SHAREMIND MPC [BLW08] general purpose MPC platform is used to implement the biometric identification. Therefore, the protocols described here are the ones implemented by SHAREMIND MPC. The following protocols are passively secure against a dishonest minority.

In this work we use a *3-out-of-3 additive secret sharing* scheme in the ring $\mathbb{Z}_{2^{64}}$. On an input x the secret sharing algorithm uniformly picks three shares x_1, x_2, x_3 in $\mathbb{Z}_{2^{64}}$ such that $x_1 + x_2 + x_3 = x \mod 2^{64}$.

Given n parties $\mathrm{P}_1, \ldots, \mathrm{P}_n$, we write P_{i+1} and P_{i-1} to denote the next and the previous party from the party P_i. We assume wrap-around of the indexes. In practice,

the parties are implemented as independent executions of the MPC program on different computers.

Sharing a value: A party P_i shares their input x by uniformly picking two values x_1 and x_2, setting $x_3 = x - x_1 - x_2 \mod 2^{64}$ and sending each x_j to P_j. We use $[x]$ to denote an additive secret sharing of x in the ring $\mathbb{Z}_{2^{64}}$.

Revealing a value: To reveal a value $[x]$, every party P_i sends their share x_i to parties P_{i+1} and P_{i-1}. Each party then sets $x = x_1 + x_2 + x_3 \mod 2^{64}$.

Addition: To compute $[z] = [x + y]$ each party P_i locally sets $z_i = x_i + y_i$.

Resharing: Resharing a value $[x]$ means picking new random shares $[y]$ such that $x = y$ and all shares in $[y]$ are independent and uniformly random. To reshare $[x]$, every party P_i picks a value r_i and sends it to P_{i+1}, then sets $y_i = x_i + r_i - r_{i-1}$.

Multiplication: To compute $[z] = [x \cdot y]$ the parties compute the following:

1. Reshare $[x]$ to get $[x']$.
2. Reshare $[y]$ to get $[y']$.
3. Each party P_i sends x'_i to P_{i+1}.
4. Each party P_i computes $z'_i = x'_i y'_i + x'_i y'_{i-1} + x'_{i-1} y'_i$
5. Reshare $[z']$ to get $[z]$

For formal definitions and proofs of correctness and security of these and other protocols, see Bogdanov's PhD thesis [Bog13].

Bitwise Secret Sharing For more complicated operations on secret shared values, a binary representation is needed. Instead of secret sharing the integer value the binary digits of the integer value are secret shared. The resulting shares are bit arrays that can be reconstructed into the original value by element-wise summing the shares. Each bit is shared separately using the same protocol as before but in the ring $\mathbb{Z}_2$. This type of secret sharing is often called XOR-sharing as the sum of binary digits is equivalent to the XOR operation. The protocol for converting an additively shared value to a bitwise shared value is called bit-extraction. The bit-extraction protocol is used for constructing the integer comparison protocol [Bog13].

1.5 Fingerprint Recognition

Fingerprints can be distinguished from one another by global and local features. Global features refer to regions where the fingerprint grooves form distinct shapes. These shapes are often classified as *whirls*, *loops* and *swirls*. Local features or minutiae refer to the ending or splitting of individual fingerprint ridges. [Mal05]

In describing the process of fingerprint recognition, the author of this thesis relies on a tutorial by Davide Maltoni [Mal05] and the rest of this section is based on his work.

Fingerprint recognition can generally be divided into three steps: fingerprint sensing, feature extraction and comparison. In the following paragraphs, a summary of the techniques described in the tutorial is given.

Fingerprint Sensing Historically, fingerprints were acquired by using black ink on the finger to create an imprint on paper and this imprint was digitized with a document scanner. Nowadays, digital sensors are used to capture the fingerprint without the use of ink or paper. Optical, solid-state and ultrasound sensors are used. A resolution of 250 to 300 dots per inch is considered the minimum required for locating the minutiae.

Image-Based Fingerprint Matching Two images of fingerprints are superimposed on one another and their similarity is calculated. Without sophisticated image preprocessing, this technique is not considered good as distortion of the fingerprint, uneven pressure on the sensor and changing skin condition can make images of the same fingerprint very different.

Minutiae-Based Fingerprint Matching Minutiae extraction algorithms vary widely but many require image binarisation. The binary image reveals the ridge and valley structure more clearly. The ridges can then be thinned and ridge ends and splits can be found. The coordinates and ridge angle of the minutiae are stored. Some algorithms extract the minutiae directly from a gray-scale image.

To compare two fingerprints base on their minutiae, the number of matching minutia pairs are found. Generally a prealignment of the fingerprints is necessary. That means translating/rotating or transforming the fingerprints such that the most minutiae match. Algorithms that match minutiae in a local area can avoid the difficult alignment process.

Minutiae based fingerprint matching is the most commonly used matching technique.

Ridge Feature-Based Fingerprint Matching Extracting minutiae from low quality images is very difficult and the extraction process in general is computationally expensive. Considering this, other ridge information such as texture information or the size and shape of the fingerprint silhouette may be used.

A method proposed by [Jai+99] tessellates the image around a recognized core and extracts local texture information in each tile. This method, called FINGERCODE, represents the fingerprint as a vector of real numbers such that the comparison of fingerprints involves comparing the euclidean distances between the vectors.

In this thesis, the FINGERCODE representation of fingerprints is used as the comparison of fingerprints in this representation is vectorisable and computationally cheap. The same representation was used by Brian *et al.* [Bar+10].

In the next chapter, the process of extracting a FINGERCODE from a fingerprint image and comparing FingerCodes using MPC is examined.

2 Privacy Preserving Fingerprint Identification

2.1 FingerCode

The performance cost of using MPC is very high for most operations. An MPC application should compute as many things as possible before the secure computation part. The users of the MPC application can do any kind of data preprocessing locally using conventional computation. The operations that have to be securely computed should be vectorisable and computationally cheap.

A filterbank-based fingerprint representation where instead of minutiae, texture information is used, fits these considerations. The filterbank, consisting of a number of Gabor filters, is used to extract local texture information for a number of different areas of the fingerprint. The feature vector is a fixed length vector of the texture information of the fingerprint sample. To compare two samples, the distance between these feature vectors is computed.

In this section, the FINGERCODE [Jai+99] algorithm is examined. FINGERCODE identifies a region of interest called the core in the fingerprint image. The area around the core is tessellated into a number of bands and sectors in which the texture information is extracted.

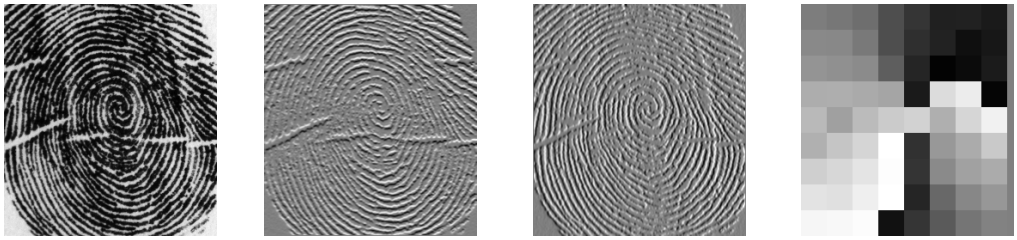
The FINGERCODE algorithm is translation, rotation and pressure invariant. This means that the algorithm accounts for different finger position and rotation in images and different pressure applied to the finger when recording the fingerprint. Translation invariance is achieved by finding a reference point or core in the fingerprint image.

Finding the Core The core is identified as the center of the fingerprint where the curvature of the ridges is the largest. Firstly, the ridge orientation field of the fingerprint must be computed:

1. Crop the fingerprint image so that only the ridges of the fingerprint are visible and divide the image into squares with sides of length w pixels.
2. Compute the vertical and horizontal gradients at each pixel using a gradient operation such as a *Sobel* filter [Fis+00].
3. For each $w \times w$ square, estimate the local orientation. See [Jai+99] for the equations used.

See Figure 1 for an illustration of the resulting images from each step.

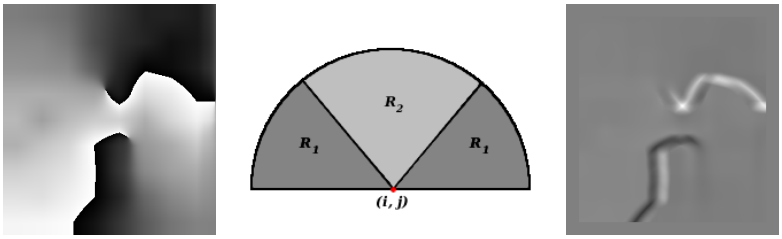
The reference point is identified from the orientation field:



(a) Fingerprint image (b) Horizontal gradient (c) Vertical gradient (d) Orientation field estimate

Figure 1. Computing the fingerprint ridge orientation field

1. Convert the orientation filed into a continuous vector field and smooth the $w \times w$ blocks with low-pass filtering.
2. Compute the image containing only the sine component of the smoothed orientation field.
3. Around each pixel, sum the pixel intensities in a sector R_1 above the pixel and a region R_2 combined from two sectors on the side of the pixel. Subtract the sum in R_2 from the sum in R_1 to find how much the orientation field curves around the pixel. See Figure 2b for an illustration of the regions.
4. The core is the pixel around which the orientation field curves the most.
5. To improve accuracy, repeat the steps around the core with a smaller w.



(a) Smoothed orientation field (b) Integration regions (c) Orientation field curvature

Figure 2. Finding the reference point

The core is used as the center from where to tessellate the fingerprint image into concentric bands that are divided into sectors. From each of those sectors local texture information is extracted. The number of bands and radius of the bands is chosen based

on the size of the sensor output and the quality of the images. In this thesis 5 bands with radius 14 were used. Each band is divided into 16 for a total of 80 sectors. An illustration of a tessellated fingerprint image is given in Figure 3.

Figure 3. Concentric bands and sectors

Normalizing the Image When recording the fingerprint, the pressure of the finger on the sensor is not always the same and is not even along the fingerprint. This leads to uneven contrast and brightness in the image. To account for this, the image need to be normalized so that the contrast and brightness are even over all sectors and among all fingerprints in the database. For each sector, the mean and variance of pixel intensities are normalised to a fixed value.

Gabor Filtering Gabor [PW08] filters that are tuned to the orientation and frequency of the ridges in the fingerprint remove noise and bring out the structure. Minutiae can be viewed as disruptions to the parallel pattern of the ridges. A Gabor filter with the right orientation can remove the ridge structure and keep the minutiae. In each image sector, the filtered image is used to extract local texture information.

Eight Gabor filters angled at different evenly spaced angles between 0° and 180° degrees form the filterbank. The filters are tuned to a frequency of $1/K$ where K is the average distance between fingerprint ridges. The specifications of the Gabor filters are explained in further detail in [JPH99].

In each sector of the tessellated image, the average absolute deviation (AAD) of the pixel values of the filtered images is computed. Since the region of interest is tessellated into 80 sectors and 8 filters are used, a total of 640 values are computed. A vector of these values is the FINGERCODE of a fingerprint image.

To find the closest match to a sample amongst some labeled templates, the euclidean distances between the sample's FINGERCODE and every template's FINGERCODE is computed and the label of the template with the smallest distance is considered the best

match. The distance of the closest match is compared against a threshold value to decide if the fingerprints match. The threshold is required because none of the templates might be a match.

2.2 Application Programming for Sharemind MPC

SHAREMIND MPC is a multi-party computation platform that provides several systems: servers that execute cryptographic protocols; programming languages to write MPC programs; client software to execute said programs; a web interface; privacy preserving databases; a data entry tool. SHAREMIND MPC has been continually developed for over a decade to make utilising multi-party computation easier.

A number of SHAREMIND MPC servers form a network that executes MPC protocols. This network can be thought of as a virtual machine with instructions for operating on data. Unlike a regular virtual machine, the Sharemind virtual machine computes on secret shared data.

There are multiple different ways of interfacing with the SHAREMIND MPC servers. These interfaces allow input parties to upload their data and result parties to run computations on said data. Although there are generally three parties running the servers, there can be an arbitrary number of input and result parties. See Figure 4 for an illustration of the parties involved in the MPC computation.

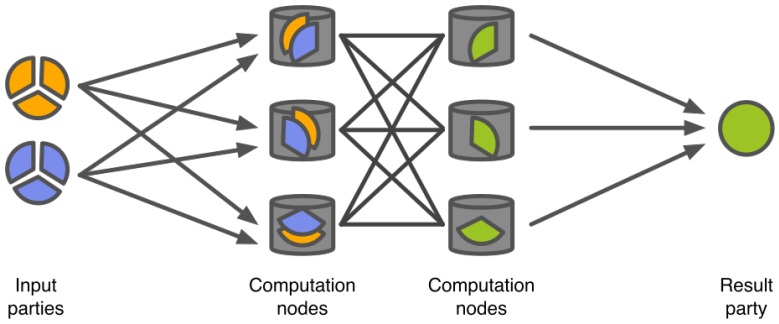


Figure 4. Input, output and computing parties [Cyb19b]

The input parties secret share their data locally and distribute the shares among the servers. The servers store their representative shares so they can be used in computations. In this thesis, the data was inputted using the SHAREMIND CSV IMPORTER tool.

The result party is the party with the fingerprint sample to be matched against the templates. The result party uses the SHAREMIND-RUNSCRIPT client application to input the FINGERCODE of their fingerprint sample and execute the query.

Shared3p A *protection domain* defines the security model for MPC computations. The most common protection domain used in SHAREMIND MPC is `shared3p` which provides security in the *three-party passively secure honest majority* setting. The protection domain contains data types such as integers and floating point numbers and functions for operating on these types. The `shared3p` security model is most suitable for a situation where all of the parties have a vested interest in achieving correct computation results and none of the parties trust any other party with their data. The following analysis is based on Bogdanov's PhD thesis [Bog13].

In `shared3p` only a single passively corrupted party is tolerated. If the corrupted party is an active adversary meaning that they might not follow the protocol honestly, the correctness of results is not guaranteed. An active adversary can stop computing or send incorrect messages to the other parties. Shared3p does not detect such a corruption. Therefore, the parties have to be chosen such that they are interested in the computation producing the correct results. This interest can be based on the business case of the MPC application. In the Danish sugar beet auction [Bog+09], the computing parties gained a business advantage from the MPC application producing correct results.

The `shared3p` *protection domain* assumes an honest majority. If an adversary has corrupted more than one party, these parties can share information amongst themselves to reveal the secrets of an honest party. However, one actively corrupted party can not reveal the secret data of other parties. In the Estonian student employment study[Tal16], the computing parties signed contracts obliging them to not collude with each other.

Considering these caveats, `shared3p` might seem like a very limiting security setting. That would be the case if the participants of the MPC application were untrustworthy or had no personal stake in keeping the data private. Every MPC privacy setting comes with some compromises, finding a balance between security, performance and usability is key to a successful application. In the case of international cooperation for fingerprint identification, the law enforcement agencies of different countries already have an interest in helping foreign agencies identify suspects across borders [Lem10]. As the agencies hold people's personal data, they have a interest in keeping it secret form foreign parties. The computing parties would have to be contractually obligated to not collude with each other.

In some cases, contractual obligation is not a convincing enough guarantee that some of the parties will not collude with each other. Security settings that tolerate a dishonest majority can prevent colluding parties from revealing the secret data of honest parties. The SHAREMIND MPC platform does not contain such a security setting.

SecreC Programming Language One of the ways of combining SHAREMIND MPC's cryptographic protocols into more complex privacy preserving applications is the SECREC programming language [Ran17]. SECREC is a domain specific language modeled after the C programming language. Its key feature is privacy preserving data types which abstract away the distributed nature of MPC and the secret shared representation of data. When programming in SECREC the user does not need to know about the multiple servers executing cryptographic protocols; The programmer only sees the protection domain in which the functions are executed.

In SECREC, all variables have a data type and an additional protection domain identifier. The protection domain can be *public*, which means that this data is not secret shared. If the protection domain is not public (for example `shared3p`), than only privacy preserving operations are allowed on that data. The SECREC type system and static analyser prevent the user from unintentionally leaking private data.

To make private data public, an explicit conversion is needed. The programmer has to verify that any data that is *declassified* (made public), does not reveal too much about the private inputs. It is possible to publish and reconstruct a value at a result party such that the computing parties do not see that value.

To prevent private data from leaking into public variables, SECREC does not allow branching based on private conditionals.

A SECREC program can be compiled into SHAREMIND MPC server instructions and executed by a result party using the SHAREMIND-RUNSCRIPT application. The result party can include private arguments when executing the program.

3 Results

A Python3 Jupyter[1] notebook was created to generate the fingercodes from fingerprint images. The privacy preserving program for comparing a fingercode against a database was written for the SHAREMIND MPC[2] platform using the SECREC language. The source code for the SECREC program can be found in Appendix A. The Jupyter notebook can be found in Appendix B.

3.1 Generating the FingerCode

The Jupyter notebook closely follows the FINGERCODE algorithm. A number of Python3 libraries were used for image processing. An Anaconda[3] environment was used to manage the required libraries. OpenCV[4] was used for image loading and Sobel derivatives, scikit-image[5] was used for Gabor filtering.

Using OpenCV, images are represented as NumPy ndarrays. The fingerprint images and all intermediate representations are grayscale images with floating point values. The sectors of the region of interest are implemented as binary ndarrays which are the same size as the fingerprint image.

A fingerprint image data set by Fingerprint Verification Competition [Bio01] was used as sample data. This data set contains images of 10 fingerprints with 8 samples of each one. The images in that data set vary in quality. For some images, the core is too far to the edge of the image to allow filtering. Therefore, only 39% of the images in the data set were used to generate FingerCodes. For performance measurements, the fingerprints can be repeated in the data set. Other papers in this field generally have used the NIST 9 fingerprint data set, but that is no longer publicly available online.

3.2 Comparing FingerCodes in SecreC

The input parties generate FINGERCODES from their fingerprint templates locally. For each FINGERCODE vector $x = (x_1, \ldots x_{640})$ the sum of the squares of the vector

[1] https://jupyter.org/
[2] https://sharemind.cyber.ee/sharemind-mpc/
[3] https://www.anaconda.com/products/individual
[4] https://opencv.org/
[5] https://scikit-image.org/

elements is locally precomputed.

$$s_x = \sum_{i=1}^{640} x_i^2 \tag{1}$$

The value s_x is later used to compute the distance. The SHAREMIND CSV IMPORTER tool is used to upload the FINGERCODES x along with the sum s_x and identifiers for each of the codes.

Fixed Point Numbers Operations on floating point numbers are generally slow in SHAREMIND MPC. If accuracy is not critical, a fixed point representation can be used to speed up the computation. In the fixed point representation, there is a fixed number of binary digits after the radix point. The value of a 64 bit fixed point number $b_{63}b_{62}\ldots b_2b_1$ with 32 binary digits after the radix point is computed as such:

$$2^{31} \cdot b_{63} + 2^{31} \cdot b_{62} + \ldots + 2^0 \cdot b_{32} + 2^{-1} \cdot b_{31} + \ldots + 2^{-32} \cdot b_1 \tag{2}$$

64 bit fixed point numbers can be stored as 64 bit integers. Multiplying a floating point number a by 2^{32} and rounding it to the nearest 64 bit integer results in the same binary digits as a fixed point representation of a. This is used as SHAREMIND CSV IMPORTER cannot read fixed point values from a CSV file.

Importing Data in SecreC The SECREC program loads the templates from SHAREMIND MPC's table database. The table has 642 columns: a column of unsigned integers for the labels of the templates, a column of fixed point numbers for s_x and 640 columns for the elements of the FINGERCODE vector x. The N FINGERCODES are loaded one column at a time into a $N \times 640$ matrix.

The query sample is provided as a command line argument to the SHAREMIND-RUNSCRIPT application.

Euclidean Distance The squares of the euclidean distances between the query sample and the templates are computed in an *single-instruction-multiple-data* (SIMD) fashion. The square of the euclidean distance between FingerCodes x and y is computed as such:

$$d^2 = \sum_{i=1}^{640} (x_i - y_i)^2 = \sum_{i=1}^{640} s_x - 2x_iy_i + s_y \tag{3}$$

The template FINGERCODES are stored in a $N \times 640$ 2-dimensional array. To perform multiplication between the templates and the sample, the sample has to be stored in an array with the same shape. The vector of the sample's FINGERCODE is extended vertically into a $N \times 640$ matrix by copying the vector into each row of the matrix.

Identification To find the closest match, we can compare the squares of the distances, computing the square root is not necessary. The template with the smallest distance to the sample's FingerCode is identified. The distance to it is compared against a threshold. A template is considered to match the sample if the distance is less than the threshold.

The threshold value affects the ratio between false positive matches and false negative matches. A larger threshold means that less similar fingerprint images are considered to be matching. If the threshold is smaller, images of the same fingerprint might be too different to be considered a match.

The fingerprint comparison algorithm scales linearly with the number of templates in the data set.

Identifying a fingerprint can be parallelized by dividing the data set into parts and running the identification algorithm separately on each subset of the data set. In order to saturate network and CPU resources, multiple instances of the SECREC program were executed simultaneously on separate subsets of the data set.

3.3 Performance

The performance of the secure computation phase of fingerprint identification was benchmarked on a dedicated cluster of servers. Each of the three servers has two Intel Xeon E5-2640 v3 8-core processors, 128 GB of RAM and direct 10 Gb/s network links to the other servers. The latency between two servers was 0.114 ms.

The total execution time of identifying a fingerprint against a data set of 320 templates took 4.88 seconds. Out of that, 2.67 seconds was spent reading the contents of the database into memory and finding the best match took 1.93 seconds.

Executing 4 processes with a data set of 80 templates each took 2.63 seconds total. Database access took 1.85 seconds and computing the best match took 0.47 seconds.

Because of the vast differences in hardware and network configuration, these results cannot be directly compared to [Bar+10] which recognized a sample among 320 templates in 16 seconds.

3.4 Future Work

One of the possible ways of increasing performance of the fingerprint identification MPC application is to improve database access speed. Currently SHAREMIND MPC does not allow storing vectors in table databases. Therefore, the FINGERCODE vector had to be divided into 640 table columns. Accessing each of these columns is very inefficient.

The database of FINGERCODES could be partitioned into a K-dimensional tree to speed up the process of finding a nearest neighbour to a sample FINGERCODE. This partitioning could be done locally by fingerprint template owners and the partitioned data could be combined into a single structure using MPC.

A single fingerprint comparison algorithm is not accurate enough for identification amongst a large data set. A combination of different algorithms can provide better accuracy [Mal05]. Therefore, it is beneficial to create privacy preserving variants of minutiae and image based recognition algorithms.

Conclusion

In this thesis, a privacy preserving application for identifying fingerprints was created and a methodology for comparing fingerprints and developing privacy preserving application using multi-party computation was given. The performance of the developed application was measured on dedicated hardware.

Fingerprint-based identification systems are vital tools for border control and law enforcement. Privacy preserving fingerprint identification could be used to prevent leakage and abuse of fingerprint data.

A secret sharing based approach to privacy preserving fingerprint identification is more performant than a homomorphic encryption based approach. The speed of execution is still not enough to reasonably identify a sample from a real-life international super database. Moreover, an identification scheme that is based on one algorithm is not accurate enough for use in a large data set. A more accurate and performant approach must be developed for a reasonable real-life deployment.

References

[Ale15] David Alexander. *5.6 million fingerprints stolen in U.S. personnel data hack: government*. Ed. by reuters.com. Sept. 2015. URL: https://www.reuters.com/article/us-usa-cybersecurity-fingerprints/5-6-million-fingerprints-stolen-in-u-s-personnel-data-hack-government-idUSKCN0RN1V820150923 (visited on 04/15/2020).

[Arc+18] David W. Archer, Dan Bogdanov, Liina Kamm, Y. Lindell, Kurt Nielsen, Jakob Illeborg Pagter, Nigel P. Smart, and Rebecca N. Wright. *From Keys to Databases – Real-World Applications of Secure Multi-Party Computation*. Cryptology ePrint Archive, Report 2018/450. https://eprint.iacr.org/2018/450. 2018.

[Bar+10] Mauro Barni, Tiziano Bianchi, Dario Catalano, Mario Di Raimondo, Ruggero Donida Labati, Pierluigi Failla, Dario Fiore, Riccardo Lazzeretti, Vincenzo Piuri, Fabio Scotti, and Alessandro Piva. "Privacy-Preserving Fingercode Authentication". In: *Proceedings of the 12th ACM Workshop on Multimedia and Security*. MM&Sec '10. Roma, Italy: Association for Computing Machinery, 2010, pp. 231–240. ISBN: 9781450302869. DOI: 10.1145/1854229.1854270. URL: https://doi.org/10.1145/1854229.1854270.

[BCP13] J. Bringer, H. Chabanne, and A. Patey. "Privacy-Preserving Biometric Identification Using Secure Multiparty Computation: An Overview and Recent Trends". In: *IEEE Signal Processing Magazine* 30.2 (2013), pp. 42–52.

[Bio01] BioLab. *FVC2000 Set "B" of Database 1*. Ed. by University of Bologna. 2001. URL: http://bias.csr.unibo.it/fvc2000/download.asp (visited on 08/05/2019).

[BLW08] Dan Bogdanov, Sven Laur, and Jan Willemson. "Sharemind: A Framework for Fast Privacy-Preserving Computations". In: *Proceedings of the 13th European Symposium on Research in Computer Security - ESORICS'08*. Ed. by Sushil Jajodia and Javier Lopez. Vol. 5283. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2008, pp. 192–206. ISBN: 978-3-540-88312-8.

[Bog+09] Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael Schwartzbach, and Tomas Toft.

"Secure Multiparty Computation Goes Live". In: *Financial Cryptography and Data Security*. Ed. by Roger Dingledine and Philippe Golle. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 325–343. ISBN: 978-3-642-03549-4.

[Bog13] Dan Bogdanov. "Sharemind: programmable secure computations with practical applications". PhD thesis. University of Tartu, 2013. URL: http://hdl.handle.net/10062/29041.

[Cim19] Catalin Cimpanu. *EU votes to create gigantic biometrics database*. Ed. by zdnet.com. Apr. 2019. URL: https://www.zdnet.com/article/eu-votes-to-create-gigantic-biometrics-database/ (visited on 04/22/2020).

[CJ19] Caitlin L. Chandler and Chris Jones. *EU pushes to link tracking databases*. Ed. by politico.eu. Apr. 2019. URL: https://www.politico.eu/article/eu-pushes-to-link-tracking-databases/ (visited on 04/22/2020).

[Cyb19a] Cybernetica. *Performance of shared3p protocols*. Ed. by Cyernetica. 2019. URL: https://docs.sharemind.cyber.ee/2019.03/development/secrec-reference#performance-of-shared3p-protocols (visited on 05/07/2020).

[Cyb19b] Cybernetica. *Secure multi-party computation*. Ed. by Cyernetica. 2019. URL: https://docs.sharemind.cyber.ee/2019.03/prologue (visited on 05/07/2020).

[Fis+00] Robert Fisher, Simon Perkins, Ashley Walker, and Erik Wolfart. *Feature Detectors - Sobel Edge Detector*. Ed. by Hypermedia Image Processing Reference. 2000. URL: http://homepages.inf.ed.ac.uk/rbf/HIPR2/sobel.htm (visited on 05/06/2020).

[Gri19] Roger A. Grimes. *6 reasons biometrics are bad authenticators (and 1 acceptable use)*. Ed. by csoonline.com. Jan. 2019. URL: https://www.csoonline.com/article/3330695/6-reasons-biometrics-are-bad-authenticators-and-1-acceptable-use.html (visited on 01/14/2020).

[Int] Interpol.int, ed. *Fingerprints*. URL: https://www.interpol.int/en/How-we-work/Forensics/Fingerprints (visited on 04/22/2020).

[Jai+99] A. K. Jain, S. Prabhakar, Lin Hong, and S. Pankanti. “FingerCode: a filterbank for fingerprint representation and matching”. In: *Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149)*. Vol. 2. 1999, 187–193 Vol. 2.

[JPH99] Anil K. Jain, Salil Prabhakar, and Lin Hong. “A Multichannel Approach to Fingerprint Classification”. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 21.4 (Apr. 1999), pp. 348–359. ISSN: 0162-8828. DOI: 10.1109/34.761265. URL: https://doi.org/10.1109/34.761265.

[KPR18] Marcel Keller, Valerio Pastro, and Dragos Rotaru. “Overdrive: Making SPDZ Great Again”. In: *Advances in Cryptology – EUROCRYPT 2018*. Ed. by Jesper Buus Nielsen and Vincent Rijmen. Cham: Springer International Publishing, 2018, pp. 158–189. ISBN: 978-3-319-78372-7.

[Lem10] Frederic Lemieux. “The nature and structure of international police cooperation: an introduction”. In: Jan. 2010, pp. 1–22.

[Mal05] Davide Maltoni. “A Tutorial on Fingerprint Recognition”. In: *Advanced Studies in Biometrics: Summer School on Biometrics, Alghero, Italy, June 2-6, 2003. Revised Selected Lectures and Papers*. Ed. by Massimo Tistarelli, Josef Bigun, and Enrico Grosso. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 43–68. ISBN: 978-3-540-28638-7. DOI: 10.1007/11493648_3. URL: https://doi.org/10.1007/11493648_3.

[Pan17] Alisa Pankova. “Efficient multiparty computation secure against covert and active adversaries”. PhD thesis. University of Tartu, 2017. URL: http://hdl.handle.net/10062/56359.

[PPJ03] Salil Prabhakar, S. Pankanti, and Anil Jain. “Biometric Recognition: Security and Privacy Concerns”. In: *Security & Privacy, IEEE* 1 (Apr. 2003), pp. 33–42. DOI: 10.1109/MSECP.2003.1193209.

[PRC15] V. M. Patel, N. K. Ratha, and R. Chellappa. “Cancelable Biometrics: A review”. In: *IEEE Signal Processing Magazine* 32.5 (2015), pp. 54–65.

[PW08] N. Petkov and M.B. Wieling. *Gabor Filter for Image Processing and Computer Vision*. Ed. by University of Groningen. 2008. URL: http://matlabserver.cs.rug.nl/edgedetectionweb/web/edgedetection_examples.html (visited on 05/06/2020).

[Ran17] Jaak Randmets. “Programming Languages for Secure Multi-party Computation Application Development”. PhD thesis. University of Tartu, 2017. URL: http://hdl.handle.net/10062/56298.

[Reb12] Reimo Rebane. “A Feasibility Analysis of Secure Multiparty Computation Deployments”. MA thesis. Institute of Computer Science, University of Tartu, 2012.

[Rob07] Chris Roberts. “Biometric attack vectors and defences”. In: *Computers & Security* 26.1 (2007), pp. 14–25. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2006.12.008. URL: http://www.sciencedirect.com/science/article/pii/S016740480600215X.

[Sha79] Adi Shamir. “How to Share a Secret”. In: *Commun. ACM* 22.11 (Nov. 1979), pp. 612–613. ISSN: 0001-0782. DOI: 10.1145/359168.359176. URL: https://doi.org/10.1145/359168.359176.

[Tal16] Riivo Talviste. “Applying Secure Multi-party Computation in Practice”. PhD thesis. University of Tartu, 2016. URL: http://dspace.ut.ee/handle/10062/50510.

[UPJ05] Umut Uludag, Sharath Pankanti, and Anil K. Jain. “Fuzzy Vault for Fingerprints”. In: *Proceedings of the 5th International Conference on Audio- and Video-Based Biometric Person Authentication*. AVBPA’05. Hilton Rye Town, NY: Springer-Verlag, 2005, pp. 310–319. ISBN: 3540278877. DOI: 10.1007/11527923_32. URL: https://doi.org/10.1007/11527923_32.

Appendix A

```
import shared3p;
import shared3p_table_database;
import shared3p_statistics_common;
import shared3p_matrix;
import stdlib;
import profiling;
import table_database;

domain pd_shared3p shared3p;

template<domain D : shared3p>
D fix64[[1]] intToFix(D int64[[1]] x) {
    D fix64[[1]] fix(size(x));
    D uint64[[1]] xuint = (uint64) x;
    __syscall("shared3p::assign_uint64_vec",
                __domainid(D), xuint, fix);
    return fix;
}



template<domain D, type T>
D T[[2]] extendHorizontal(D T[[1]] x, uint n) {
    uint m = size(x);

    D T[[2]] res(m, n);
    uint[[1]] indices(m * n);

    for (uint i = 0; i < m; ++i) {
        for (uint j = 0; j < n; ++j) {
            indices[i * n + j] = i;
        }
    }

    __syscall("shared3p::gather_$T\_vec",
                __domainid(D), x, res, __cref indices);

    return res;
}


template<domain D, type T>
D T[[2]] extendVertical(D T[[1]] x, uint n) {
    uint m = size(x);

    D T[[2]] res(n, m);
```

```
47      uint[[1]] indices(n * m);
48
49      for (uint i = 0; i < n; ++i) {
50          for (uint j = 0; j < m; ++j) {
51              indices[i * m + j] = j;
52          }
53      }
54
55      __syscall("shared3p::gather_$T\_vec",
56              __domainid(D), x, res, __cref indices);
57
58      return res;
59  }
60
61
62  void main () {
63      //profiling
64      uint32 profile_database = newSectionType(
65                                "database_access");
66      uint32 profile_comparison = newSectionType(
67                                "fingercode_comparison");
68
69
70      // Open database
71      string datasource = "DS1";
72      string table_name = "fingercodes";
73      tdbOpenConnection (datasource);
74
75
76      // Get client arguments
77      pd_shared3p int64[[1]] sample_fc_tmp = argument("samplefc");
78      pd_shared3p int64[[1]] sample_sq_tmp = argument("samplesq");
79      pd_shared3p fix64[[1]] sample_fc = intToFix(sample_fc_tmp);
80      pd_shared3p fix64 sample_sq = intToFix(sample_sq_tmp)[0];
81
82
83      // Get database data
84      uint N = tdbGetRowCount(datasource, table_name);
85      uint32 prof_sec = startSection(profile_database, N);
86      pd_shared3p uint32[[1]] id = tdbReadColumn(datasource,
87                                      table_name, 0::uint64);
88
89      pd_shared3p int64[[1]] sq_tmp = tdbReadColumn(datasource,
90                                  table_name, 2 * 1::uint64);
91      pd_shared3p fix64[[1]] sq = intToFix(sq_tmp);
92
93      pd_shared3p fix64[[2]] fc(N, 640);
94
95      print("Starting to read colums");
```

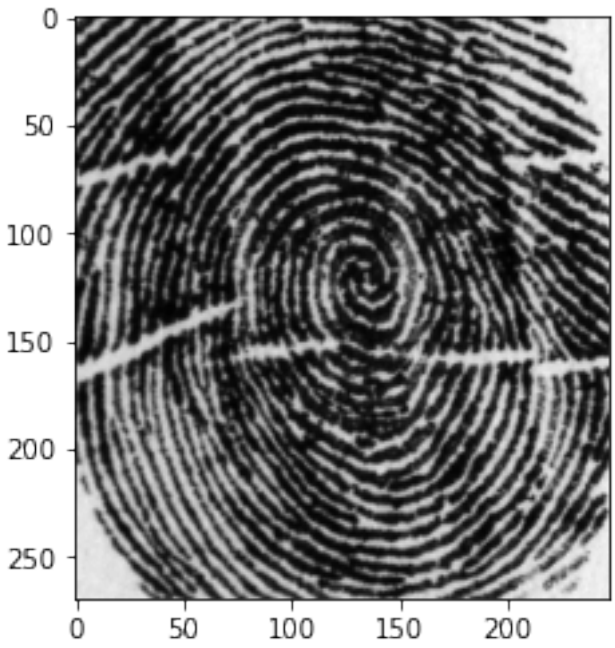
```
    for (uint i = 0; i<640; i++) {
        pd_shared3p int64[[1]] tmp = tdbReadColumn(datasource,
                                    table_name, 4 + 2 * i::uint64);
        pd_shared3p fix64[[1]] tmpfix = intToFix(tmp);
        fc[:,i] = tmpfix;
    }
    print("finished reading columns");

    pd_shared3p fix64[[2]] sample_repeat =
                              extendVertical(sample_fc, N::uint);
    print("finished extend");
    endSection(prof_sec);


    // Compute distances between sample and templates
    prof_sec = startSection(profile_comparison, N);

    pd_shared3p fix64 negtwo = -2;

    pd_shared3p fix64[[1]] dist = sample_sq + sq +
                              negtwo * rowSums(fc*sample_repeat);

    pd_shared3p fix64 match_dist = min(dist);
    pd_shared3p bool[[1]] mask = (dist == match_dist);
    pd_shared3p uint32[[1]] res = cut(id, mask);

    endSection(prof_sec);

    float64 threshold = 0.08;


    // Publish results
    publish("identified_in_list",  match_dist<threshold);
    publish("closest_match", res);
}
```

Appendix B

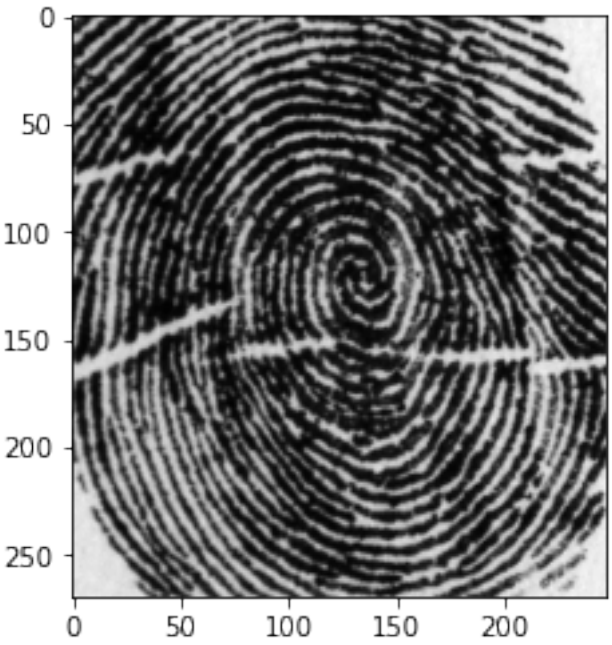
[1]:
```
import cv2

import numpy as np
from scipy import ndimage as ndi
from matplotlib import pyplot as plt

img = cv2.imread('databases/example.tif', 0)
print(img.shape)

img = img[15:-15,15:-15]
plt.imshow(img, cmap='gray')
```

```
(300, 278)
```

[1]:
```
<matplotlib.image.AxesImage at 0x7f9bbc7261d0>
```

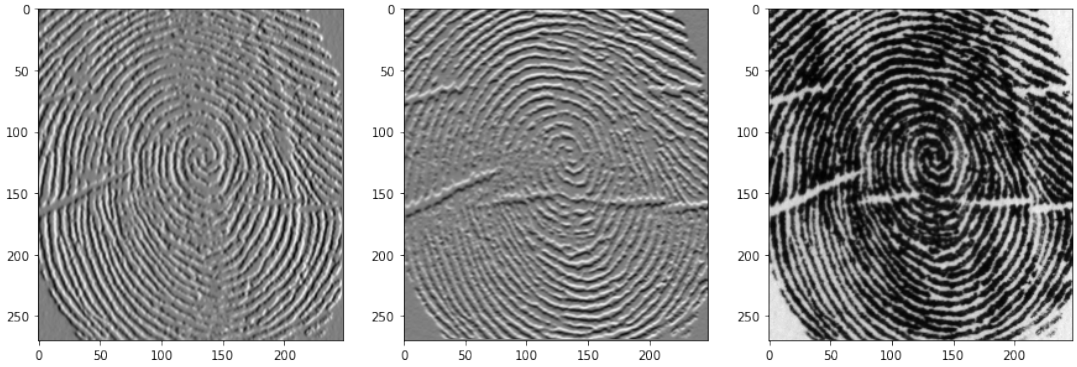


[2]:
```
# Normalize pixel values to (0, 1)

a = img.min()
b = img.max()
img_2 = (img - a) * (1/(b-a))
plt.imshow(img_2, cmap='gray', vmin=0, vmax=1)
img = img_2
```



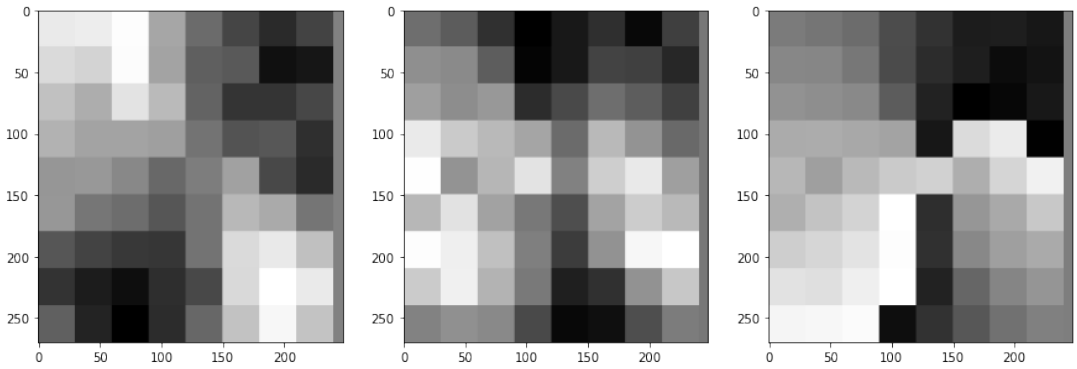
[3]:
```
# Finding the reference point

# Algorithm described in: Filterbank-Based Fingerprint Matching

ksize = 3  # kernel size for Sobel derivative
scale = 1
delta = 0
ddepth = -1  # destination will be the same type as input (f64)

# get gradient from partial derivatives in x and y axis using convolution
# https://docs.opencv.org/3.4/d2/d2c/tutorial_sobel_derivatives.html

grad_x = cv2.Sobel(img, ddepth, 1, 0, ksize=ksize, scale=scale,␣
 ↪borderType=cv2.BORDER_DEFAULT)
grad_y = cv2.Sobel(img, ddepth, 0, 1, ksize=ksize, scale=scale,␣
 ↪borderType=cv2.BORDER_DEFAULT)

grad_x = grad_x / np.abs(grad_x).max()  # normalize to [-1, 1]
grad_y = grad_y / np.abs(grad_y).max()

# Display
fig=plt.figure(figsize=(15, 5))

fig.add_subplot(1, 3, 1)
plt.imshow(grad_x, cmap='gray', vmin=-1, vmax=1)

fig.add_subplot(1, 3, 2)
plt.imshow(grad_y, cmap='gray', vmin=-1, vmax=1)
```

```
fig.add_subplot(1, 3, 3)
plt.imshow(img, cmap='gray', vmin=0, vmax=1)


plt.show()
```



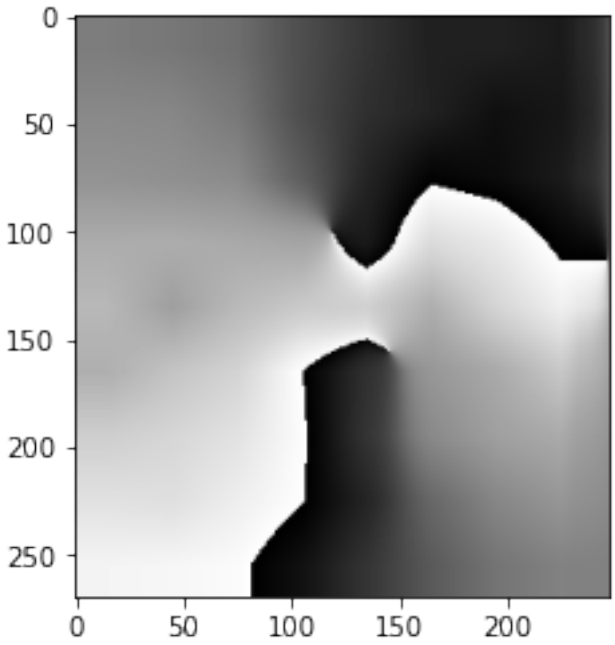
[4]:
```
# Using x and y gradient to estimate the direction of lines in a w*w block
# O is orentation field

w = 30  # block size for direction estimation
w2 = w // 2

size_x = img.shape[0] // w
size_y = img.shape[1] // w
Vx = np.zeros(img.shape, dtype=np.float32)
Vy = np.zeros(img.shape, dtype=np.float32)

Ux = 2 * grad_x * grad_y
Uy = (grad_x)**2 - (grad_y)**2

for i in range(0, size_x):
    for j in range(0, size_y):
        Vx[i*w:i*w+w, j*w:j*w+w] = np.sum(Ux[i*w:i*w+w, j*w:j*w+w])
        Vy[i*w:i*w+w, j*w:j*w+w] = np.sum(Uy[i*w:i*w+w, j*w:j*w+w])

O = 0.5 * np.arctan2(Vy, Vx)

# Display
fig=plt.figure(figsize=(15, 5))

fig.add_subplot(1, 3, 1)
plt.imshow(Vx, cmap='gray')

fig.add_subplot(1, 3, 2)
plt.imshow(Vy, cmap='gray')

fig.add_subplot(1, 3, 3)
```

```
plt.imshow(O, cmap='gray')

plt.show()
```

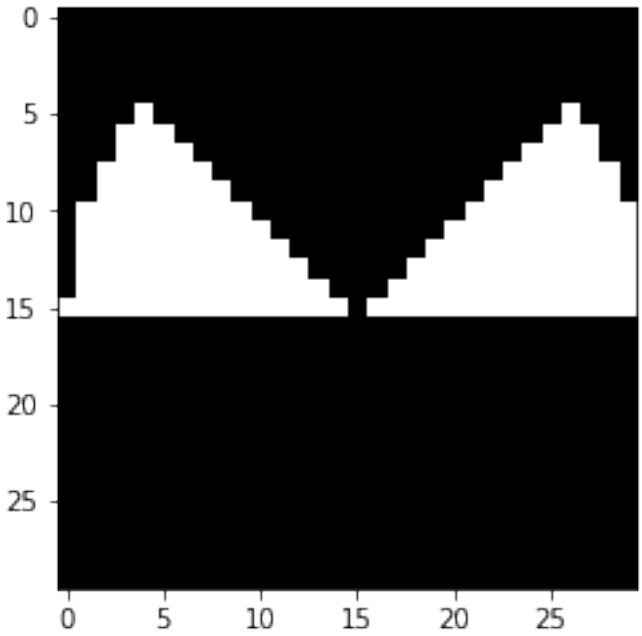


[5]:
```
# Smoothing the orientation field

phi_x = np.cos(2*O)
phi_y = np.sin(2*O)

Fx = np.zeros(img.shape)
Fy = np.zeros(img.shape)

sx = img.shape[1]
sy = img.shape[0]

for x in range(sx):
    for y in range(sy):
        Fx[y, x] = np.mean(phi_x[max(0, y-w2):min(y+w2, sy), max(0, x-w2):
 ↪min(x+w2, sx)])
        Fy[y, x] = np.mean(phi_y[max(0, y-w2):min(y+w2, sy), max(0, x-w2):
 ↪min(x+w2, sx)])

O_prime = 0.5 * np.arctan2(Fy, Fx)
plt.imshow(O_prime, cmap='gray')
```

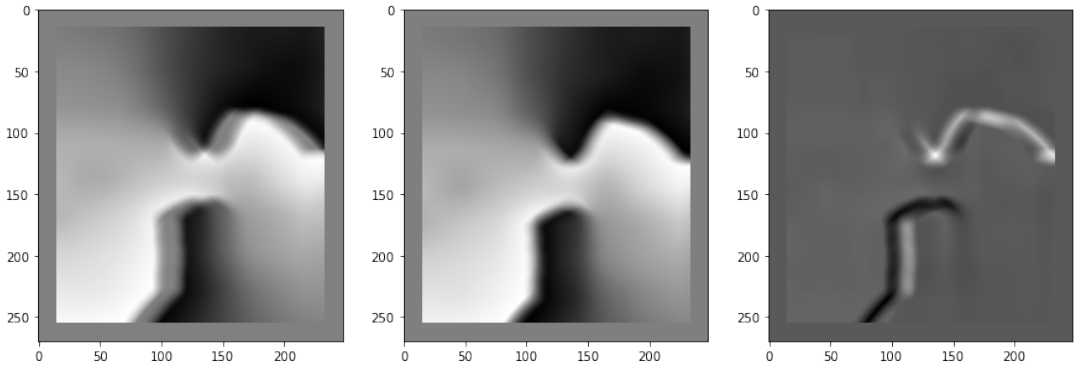
[5]: <matplotlib.image.AxesImage at 0x7f9bbab87e90>



[6]:
```
# compute the areas of integration

r = 15

Ri = [[1 if (i**2 + j**2) <= r**2 and i <= 0 and abs(i) < abs(j) else 0 for j␣
 ↪in range(-r, r)] for i in range(-r, r)]
Rii = [[1 if (i**2 + j**2) <= r**2 and i <= 0 and abs(i) > abs(j) else 0 for␣
 ↪j in range(-r, r)] for i in range(-r, r)]

Ri = np.asarray(Ri)
Rii = np.asarray(Rii)

plt.imshow(Ri, cmap='gray')
plt.show()
```

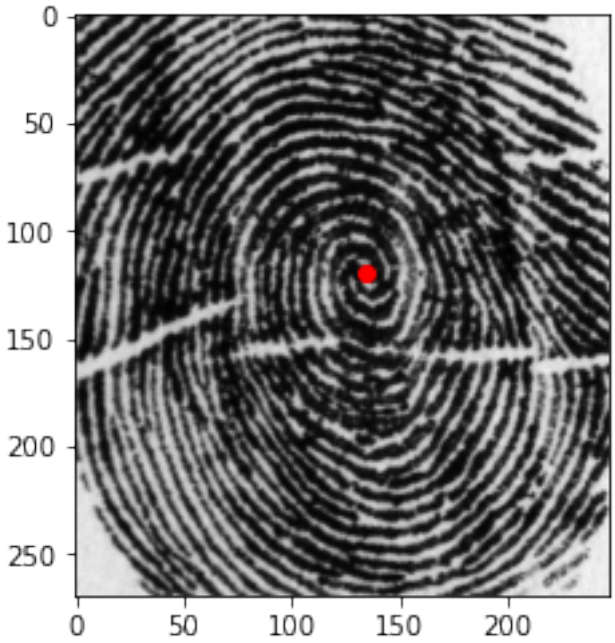


[7]:
```
# Integrate the curvature in the two areas Ri and Rii

eps_i = np.zeros(img.shape)
eps_ii = np.zeros(img.shape)

sx = img.shape[1]
sy = img.shape[0]

for x in range(r, sx-r):
    for y in range(r, sy-r):
        eps_i[y, x] = np.sum(Ri * O_prime[y-r:y+r, x-r:x+r])
        eps_ii[y, x] = np.sum(Rii * O_prime[y-r:y+r, x-r:x+r])

A = eps_i - eps_ii

# Display
fig=plt.figure(figsize=(15, 5))

fig.add_subplot(1, 3, 1)
plt.imshow(eps_i, cmap='gray')

fig.add_subplot(1, 3, 2)
plt.imshow(eps_ii, cmap='gray')

fig.add_subplot(1, 3, 3)
plt.imshow(A, cmap='gray')

plt.show()
```



[8]:
```
core = np.unravel_index(np.argmax(A), A.shape)
print(core)

plt.imshow(img, cmap='gray')
plt.plot(core[1], core[0], 'ro')
plt.show()
```

```
(119, 135)
```

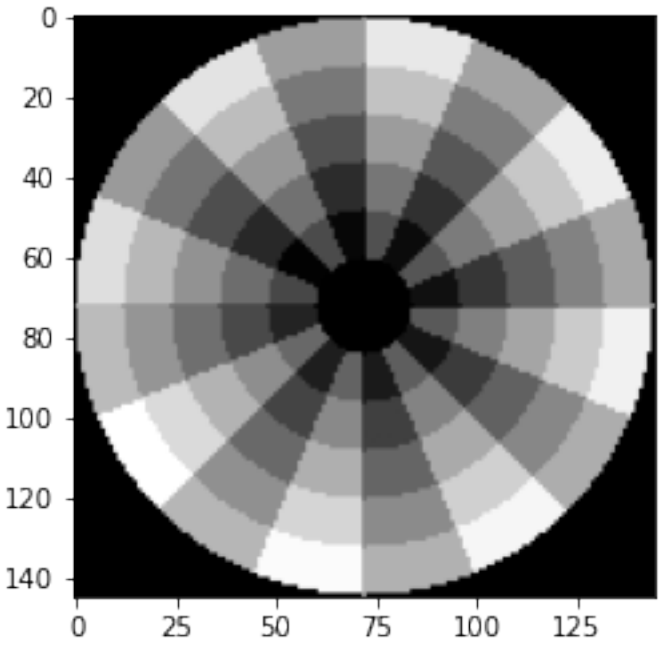


[9]:
```
# Create the image masks for each sector

import skimage.morphology
from math import pi

ns = 16  # number of sectors
nb = 5  # number of bands
```

```
bw = 12  # band width
r = (nb+1)*bw

print(r)

sectors = []

for j in range(1, nb+1):  # band
    a = skimage.morphology.disk(j*bw + bw)
    b = skimage.morphology.disk(j*bw)
    c = a - np.pad(b, bw)
    c = np.pad(c, bw * (nb - j))

    for i in range(0, ns):  # sector
        deg1 = ((2*pi/16) * i) - pi
        deg2 = ((2*pi/16) * (i+1)) - pi

        d = [[1 if deg1 < np.arctan2(y, x) <= deg2 else 0 for x in range(-r,␣
 ↪r+1)] for y in range(-r, r+1)]
        d = np.asarray(d)

        sectors.append(c * d)
```

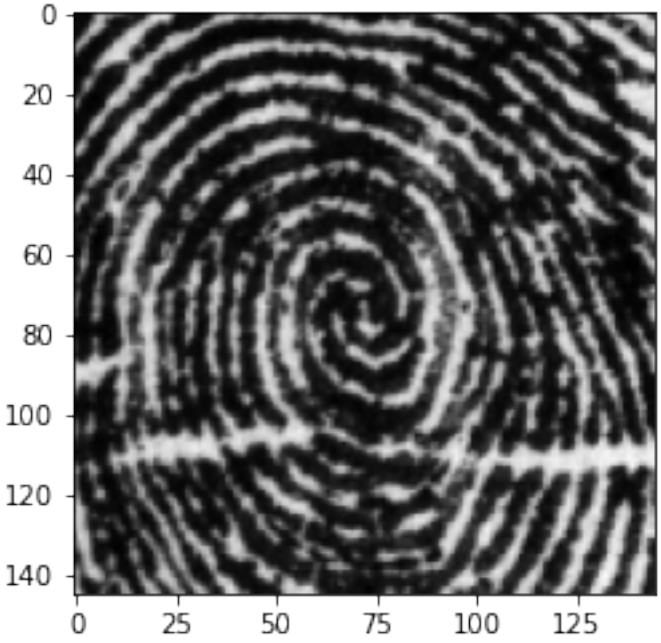
```
72
```

[10]:
```
a = np.zeros((145, 145))
for i in range(80):
    b = 30 if i%2==0 else 0
    a += (i+b)*sectors[i]

plt.imshow(a, cmap='gray')
plt.show()
```

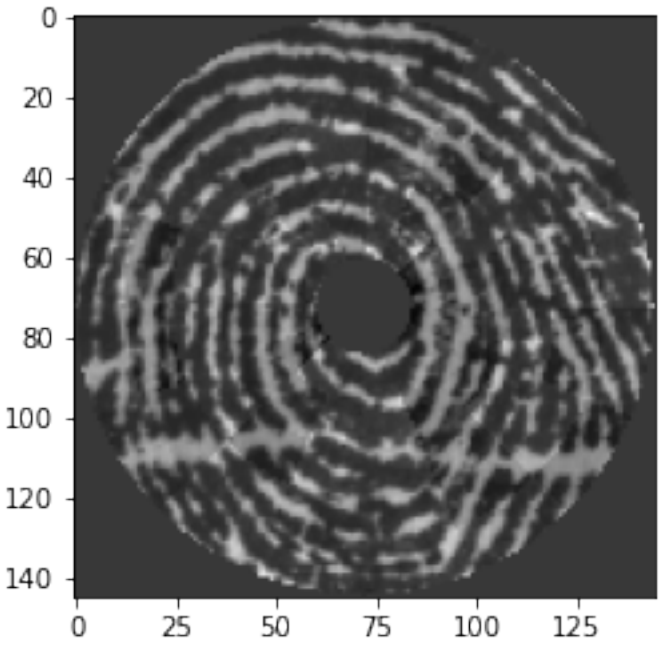


[11]:
```
crop = img[core[0]-r:core[0]+r+1, core[1]-r:core[1]+r+1]
print(crop.shape, r)

plt.imshow(crop, cmap='gray')
plt.show()
```

```
(145, 145) 72
```

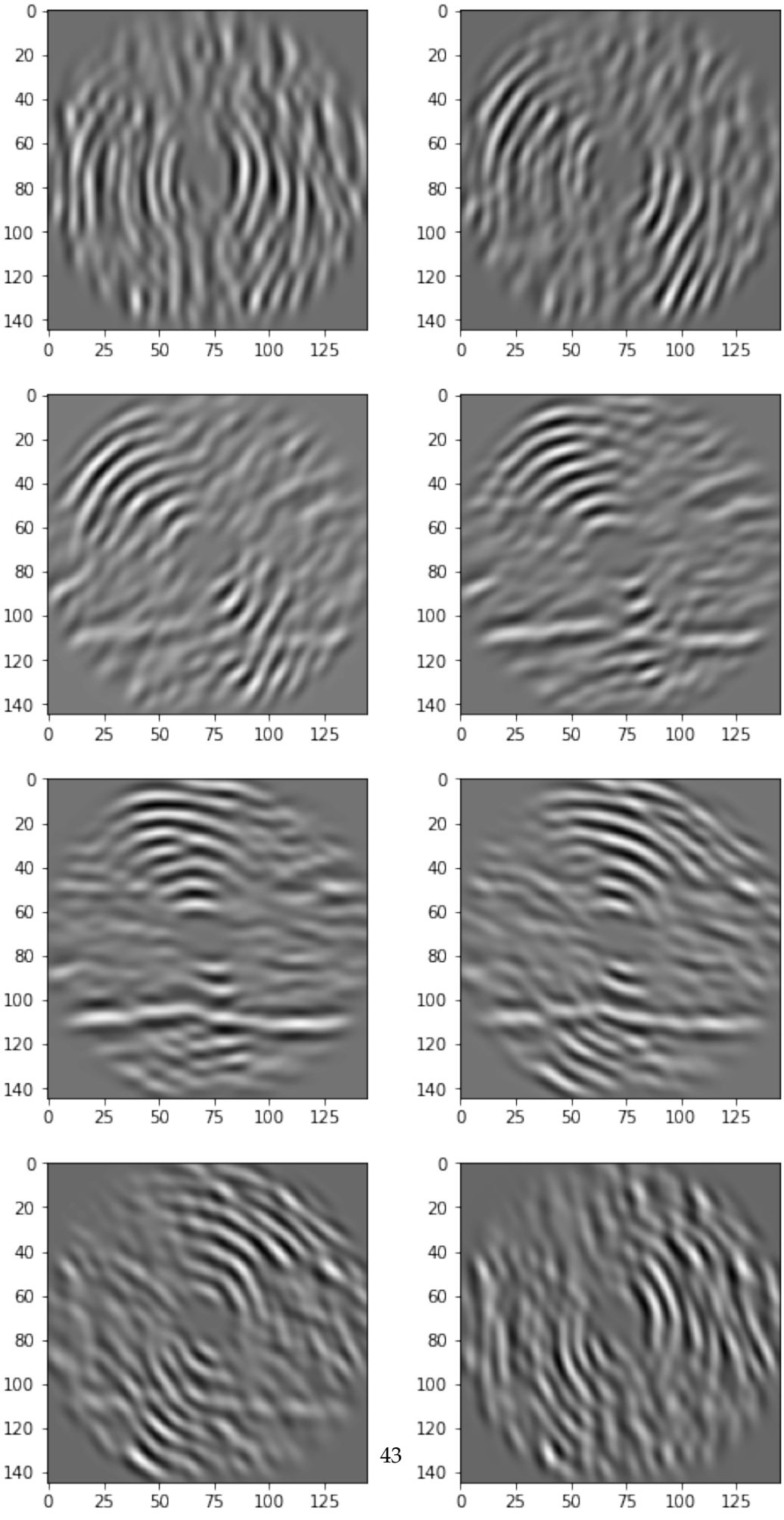


[12]:
```
# Normalize the pixel value mean and variance in each sector
norm = np.zeros(crop.shape)

m = 0.5
v = 0.5

for i in sectors:
    mean = ndi.mean(crop, i)
    var = ndi.variance(crop, i)

    mask = np.where(crop > mean, 1, -1)
    norm += i * (m + mask * np.sqrt( (v * (crop - mean)**2) / var ))


plt.imshow(norm, cmap='gray')
plt.show()
```



[13]:
```
# Create the gabor filters and the FingerCode feature vectors
from skimage.filters import gabor_kernel

f = 1 / 10
values = []

# Display
fig=plt.figure(figsize=(8, 16))


for i in range(8):
    theta = i * (pi/8)
```

```
    kernel = np.real(gabor_kernel(f, theta=theta, \
                            sigma_x=4.0, sigma_y=4.0))

    gabor = ndi.convolve(norm, kernel, mode='constant', cval=0.0)

    im = np.zeros(norm.shape)

    for sec in sectors:
        var = ndi.variance(gabor, sec)
        im += sec * var
        values.append(var)

    fig.add_subplot(4, 2, i+1)
    plt.imshow(gabor, cmap='gray')

plt.show()
```


0
20
40
60
80
100
120
140
0
25
50
75
100
125

Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Hendrik Eerikson**,
*(*author's name)

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

   reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   Privacy Preserving Fingerprint Identification,
   *(*title of thesis)

   supervised by Riivo Talviste and Kristjan Krips.
   *(*supervisor's name)

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Hendrik Eerikson
08/05/2020