8

Parallel Secure Computation Scheme for Biometric Security and Privacy in Standard-Based BioAPI Framework

Arun P. Kumara Krishan, Bon K. Sy and Adam Ramirez
Queens College & Graduate Center/CUNY
Computer Science Department, SB A202
Flushing NY 11367

1. Introduction

A biometric system may be viewed as a composition of various segments; i.e., unit(s) capturing biometric samples, unit(s) preprocessing captured samples into formats from which biometric data can be extracted, or unit(s) matching biometric samples with enrolled signature(s). Various vendors may provide these components. Traditionally, once a unit is chosen from a vendor, the user is restricted in their choice of interfacing units. This restriction stems from the burden of building a custom interface and the pool of available vendors who offer units compatible with the chosen unit. The BioAPI consortium, an organization of various industry leaders, has recently developed the ISO/IEC 19784-1:2005 (also known as BioAPI 2.0) standard to address the interoperability issue. The BioAPI 2.0 framework provides a common interface for biometric systems to subscribe services of standard compliant units made available by various vendors. This allows one to pick and choose between vendors or swap units from different vendors with minimal overhead. Thus, if someone develops a more efficient/accurate matching unit, the existing matching unit can be essentially "hot swapped" with minimal interruption in service; no longer is one limited to continue using a less desirable system because of expensive migration costs. BioAPI provides an excellent framework for interoperability, but lacks provisions for the security or privacy of biometric data. To illustrate this point, consider a typical scenario:

A biometric based authentication or credentialing system requires an individual to enroll their biometric data (e.g., fingerprint) so that the biometric data can be stored for future comparison purposes. For authentication, user A presents their biometric data to be compared against enrolled biometric template(s). User A must present their biometric data to a remote matching engine for it to compare with the enrolled biometrics. In this case, user A is forced to reveal their biometric data to another party.

Various approaches exist to ensure privacy and security in biometric systems. These approaches fall into two general categories: lossy (Newton 2005) and lossless (Yao 82, Sy 2009). The problem with a lossy approach is that the anonymization of biometric data is data dependent and may not be extendable from one application to another. On the other hand, lossless approaches such as secure computation (Yao 82) are practical only on a limited set of functions characterized by linear arithmetic operations. At the time of this writing, secure

computation based on full privacy homomorphism (Gentry 2009) for computing privately any arbitrary function is not practical due to its prohibitively expensive cost in terms of computation time. Therefore, there remains a need to protect the privacy of biometric data at the same level as a lossless approach yet at the same time achieving real time results.

We have developed a unique secure computation privacy protection mechanism - referred to as SIPPA (Secure Information Processing with Privacy Assurance) - for privacy preserving biometric data retrieval and processing. SIPPA (Sy 2010) is essentially a hybrid approach where it is a lossless approach when the biometric data being compared is exactly the same, and it is a lossy approach when the biometric data being compared are not exactly the same. The basic idea behind SIPPA is to model data (D1/D2) as a linearized vector, and to transform the vector into a symmetric matrix. To compare the two data sets, we take advantage of the approximately consistent relationship between the norm deviation of the data and the norm deviation of the corresponding eigenvectors weighted by the eigenvalues. For privacy protection, neither party will share data, the corresponding matrix and the eigenvalues/eigenvectors with the other party. Instead, P1 and P2 will engage in a two-party secure computation to uncover the similarity between the weighted eigenvectors. Through the approximately consistent relationship between the data and eigenvectors, the similarity between the data can be derived, and used subsequently as a basis for the reconstruction of the source data D2. As we will discuss in the later section, SIPPA offers two very attractive properties:

1. The private data D2 can be reconstructed by P1 only if P2 sends helper data to P1 after judging D1 and D2 to be sufficiently similar. In other words, P2 has control over the sharing of his/her private data D2 – a desirable property often mentioned by privacy advocates (Solove et al 2006) as facilitating privacy by providing control over information.
2. The accuracy of the reconstruction depends on the level of closeness/similarity between D1 and D2. In other words, the ability of P1 to reconstruct accurately D2 decreases when P1 does not already know some D1 that bears similarity to D2 – typical scenario of biometrics where the biometric samples from the same individual are "sufficiently similar" but are seldom identical.

For SIPPA to be practically useful and to deliver real time computational results, a parallel processing architecture is necessary. We illustrate the design and implementation of a parallel processing architecture for SIPPA (PSIPPA, MPSIPPA), and the realization of PSIPPA (clients and servers) as service components in the BioAPI 2.0 framework. Our attempt to realize PSIPPA as a service component has two specific objectives. First, PSIPPA is exposed as a biometric service in the BioAPI framework. Second, PSIPPA presents itself as a case study to explore the possibility of extending the BioAPI framework to incorporate an innovative slice-based provisioning mechanism of biometric services.

2. Related work

The privacy of personal biometrics is generally protected by means of two approaches; namely, lossy data processing and lossless data processing. In security surveillance, protecting privacy through the lossy approach is not uncommon. Preserving privacy through the lossy approach is achieved by protecting the private content typically by perturbation, randomization or masking of the original data to the extent that it could still be useful for security purposes.

Many face de-identification techniques (Gross et al 2005) for privacy protection are based on lossy approach. The basic idea is to conduct lossy anonymization to mask away granular biometric face information not essential to its end goal on security identification or surveillance. For example, Newton et al. (2005) proposed a k-Same model based on the k-anonymity framework. The k-Same model takes the average of k face images in a face set and replaces these images with the average image; therefore, each face image presented to a face recognition/comparison system cannot be distinguished from at least k-1 faces in the gallery. In general, information leakage could be a significant risk when k is small and/or known unique aspect of an individual is not sufficiently anonymized. In other words, the degree of privacy protection based on lossy anonymization is data dependent and may not be extendable from one application to another that have different privacy requirements.

An approach towards lossless privacy protection is Secure Multi-party Computation (SMC). SMC protects computational privacy while preserving content (Du & Atallah 2001). In other words, the original content on personal biometrics can be retrieved and used in some secure computation scheme for deriving event information of interest, but the scope of derivation is limited to what is allowed by the private computational mechanism of the process.

Generally speaking, SMC deals with the problem in which two or more parties with private inputs would like to compute jointly some function of their inputs, but no party wishes to reveal its private input to other participants. For example, a physician (party P1) wants to compare a medical image of a patient containing personal biometrics with the medical image of another patient stored in a hospital database. The data custodian (e.g., party P2 who is the database administrator) in the hospital and the physician (party P1) may participate in a SMC protocol to jointly compute the output of a matching function that compares the personal biometrics in the medical images. In another example, a user (party P1) and the authentication server (party P2) may jointly compute the distance function based on the user voice sample and the enrolled voice template withheld by the authentication server for biometric verification. The Secure Multi-party Computation (SMC) problem was first introduced by Yao (1982) and extended by Goldreich et al (1987)., and by many others (Goldwasser 1997).

Goldreich pointed out that solutions to specific problems should be developed and customized for efficiency reasons. Du and Atallah (2001a) presented a series of specific solutions to specific problems; e.g., privacy-preserving cooperative scientific computations, privacy-preserving database query, and privacy-preserving geometric computation. In their Privacy-Preserving Cooperative Scientific Computations (PPCSC) (Du & Atallah 2001b), they proposed a protocol between two parties to solve the problem $(A1+A2)x = b1 + b2$, where matrix A1 and vector b1 belong to party P1, matrix A2 and vector b2 belong to party P2. At the end of the protocol, both parties know the solution x while nobody knows the other party's private inputs.

In SIPPA, the private data exchange and information processing involves PPCSC. Specifically, biometric data is represented in the form of a matrix Ai (i=1, 2), and the (most significant) eigenvector weighted by the eigenvalue is represented by the vector bi (i=1, 2). We will show in the later section that the solution vector x satisfying $(A1+A2)x = b1 + b2$ for PPCSC offers the boundary information for each party to estimate the distance between the eigenvectors of the data matrices of the two parties. This distance estimate then forms the basis for comparing the biometric data of both parties as well as the generation of helper data for reconstructing the source data. Further details about this will be discussed in a later section.

Our approach to tackle the problem of PPCSC is to employ homomorphic encryption and singular value decomposition (SVD) on the matrices of P1 and P2 to achieve privacy protection. This approach was discussed in our recent BTAS paper (Sy 2009a). For completeness, we will include the high level summary of the approach in a later section.

3. Secure Information Processing with Privacy Assurance (SIPPA)

In this research, the privacy model for biometrics can be formulated as below: Party P1 has some biometric data expressed in terms of a linearized vector D1. Party P2 has some linearized vector template D2 about a subject P3. The objective is for P1 to determine whether D1 and D2 are similar under the following conditions:

1. P1 and P2 do not reveal their private data to each other.
2. P1 and P2 both need to determine whether D1 and D2 are sufficiently similar.
3. If D1 and D2 are sufficiently similar, P2 will provide some helper data HD with a negligible overhead for P1 to reconstruct D2 using only D1 and HD.

Eigen-based approach (Turk & Pentland 1991) has been developed in the early 90s for people identification based on face biometrics. An important property of eigenface is to represent a linearized image as a covariance matrix with its corresponding eigenvectors as a set of linearly independent basis for capturing the variation of the image.

Covariance matrix of an image representation is symmetric. In fact, any linearized data vector D will yield a symmetric matrix out of $D \cdot D^T$. Let A1 and A2 be the proper transformation of some linearized data D1 and D2 through $D1 \cdot D1^T$ and $D2 \cdot D2^T$ respectively. Let $\{(\lambda^1_i, V^1_i) \mid i = 1,2\ldots\}$ and $\{(\lambda^2_i, V^2_i) \mid i = 1,2\ldots\}$ be the corresponding sets of eigenvalues and unity normalized eigenvectors for A1 and A2 respectively. For the sake of discussion and without the loss of generality, we will focus on only the largest Eigen components (λ^1_1, V^1_1) and (λ^2_1, V^2_1). Let x be a vector such that $(A1+A2)x = \lambda^1_1 \cdot V^1_1 + \lambda^2_1 \cdot V^2_1$. By definition, $A1V^1_1 = \lambda^1_1 \cdot V^1_1$ and $A2V^2_1 = \lambda^2_1 \cdot V^2_1$. These relationships manifest an endomorphism mapping with the following three observations:

Observation 1: $\lambda^1_1 \cdot V^1_1$ and $\lambda^2_1 \cdot V^2_1$ are the transformation of the eigenvectors V^1_1 and V^2_1 through A1 and A2 respectively.

Observation 2: The resultant sum of the vectors $\lambda^1_1 \cdot V^1_1$ and $\lambda^2_1 \cdot V^2_1$ are the transformation of the vector x through (A1+A2).

Observation 3: Vector x can be decomposed into components with V^i_1 (i=1,2) as basis; i.e., $x = V^1_1 + \varepsilon_1$ and $x = V^2_1 - \varepsilon_2$; whereas ε_i (i=1,2) can be considered as an error/offset term accounting for the deviation of x from V^i_1 (i=1,2).

The graphical interpretation of the observations just mentioned above is illustrated in figure 1 below:

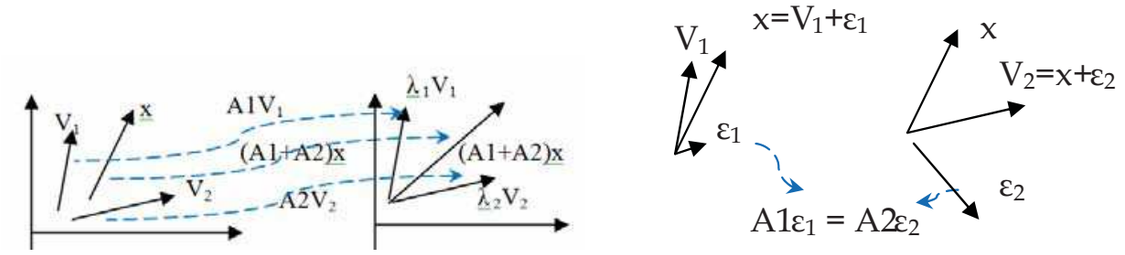


Fig. 1. Eigen-based Approach for SIPPA

As one could notice, x converges to the eigenvector as (λ^1_1, V^1_1) and (λ^2_1, V^2_1) converge to each other. This, together with observation 3, leads to the following trivial, yet important property:

Property 1: As (λ^1_1, V^1_1) and (λ^2_1, V^2_1) converge to each other, x converges to the eigenvector V^i_1 (i=1,2) and ε_is (i=1,2) converge to zero.

The key mathematical structure of SIPPA is the algebraic system of linear equations defined by $(A1+A2)x = \lambda^1_1 \cdot V^1_1 + \lambda^2_1 \cdot V^2_1$. These linear equations define the constraint relationship between a separator boundary and the eigenvectors of the two parties; whereas the solution to the algebraic system reveals the information needed to derive the lower bound distance of the norm deviation of the eigenvectors. If the lower bound is deemed acceptable, the eigenvalue and some scalar value would be sent by the server (P2) to the client (P1) as the helper data. Based on the helper data and D1, P1 can derive the norm deviation between the eigenvectors of P1 and P2 under the assumption of equidistance. Subsequently, P1 can derive a sufficiently good approximation of D2.

As one could note in property 1, it essentially states that one could infer the closeness between V^1_1 and V^2_1 through V^i_1 and x without knowing V^j_1 – the basis of SIPPA; where (i j) = (1 2) or (2 1). Furthermore, it can easily be shown that $A1\varepsilon_1 = A2\varepsilon_2$, or $D1 \cdot D1^T \cdot \varepsilon_1 = D2 \cdot D2^T \cdot \varepsilon_2$, which provides a convenient way in the SIPPA scheme to derive Dj when the scalar $Dj^T \cdot \varepsilon_j$ is known; where $x = V^1_1 + \varepsilon_1$ and $V^2_1 = x + \varepsilon_2$. In other words, SIPPA can easily facilitate "separation of duty" in the sense that data exchange/processing is only possible when both parties collaborate. Similarly, SIPPA separates the step for similarity comparison and that for retrieval. Therefore, the security principles "need-to-know" and "least privilege" can be implemented in the SIPPA environment.

3.1 SIPPA algorithm details

There are three major aspects of SIPPA:

1. Derivation of the eigenvalues and the corresponding unity normalized eigenvectors of the symmetric matrix representation of the data.
2. Derivation of a boundary vector separating the eigenvectors of the two parties, which is formulated as a two-party PPCSC secure computation.
3. Reconstruction of the source data based on the helper data composed of the eigenvalue and a scalar derived from the vector product between the transpose of the linearized source data vector and the boundary vector.

We will first outline the key steps of SIPPA, and then the secure computation protocol for PPCSC.

Let Dv and De be the sample and source linearized data respectively. The key steps of SIPPA are summarized below:

Step 1. Derive symmetric matrix representation of the data in the form of $Dv \bullet Dv^T$ (= A1) and $De \bullet De^T$ (= A2).

Step 2. Derive the eigenvalues and the corresponding unity normalized eigenvectors $\{(\lambda^1_i, V^1_i) \mid i= 1,2\ldots\}$ of $Dv \bullet Dv^T$ and $\{(\lambda^2_i, V^2_i) \mid i= 1,2\ldots\}$ of $De \bullet De^T$.

Step 3. Compute x such that $(A1+A2)x=\lambda^1_1 \cdot V^1_1 + \lambda^2_1 \cdot V^2_1$ based on our PPCSC (Sy 2009b) summarized in Fig. 2 below

Step 4. Derive the closeness between De and Dv via the minimum distance between V^1_1 and V^2_1. The minimum distance between V^1_1 and V^2_1 is estimated via $|V^2_1 - x|$.

Step 5. If De and Dv are sufficiently close as measured by $|V^2_1 - x| <$ some pre-defined threshold, proceed to send the following helper data: λ^2_1 and $De^T \cdot x$.

Step 6. Derive $\underline{V}^2_1 = V^1_1 + k(x - V^1_1)$, and then the estimated source data $De' = \lambda^2_1 \cdot \underline{V}^2_1 / De^T \cdot x$; where k =2 under the assumption of equidistance.

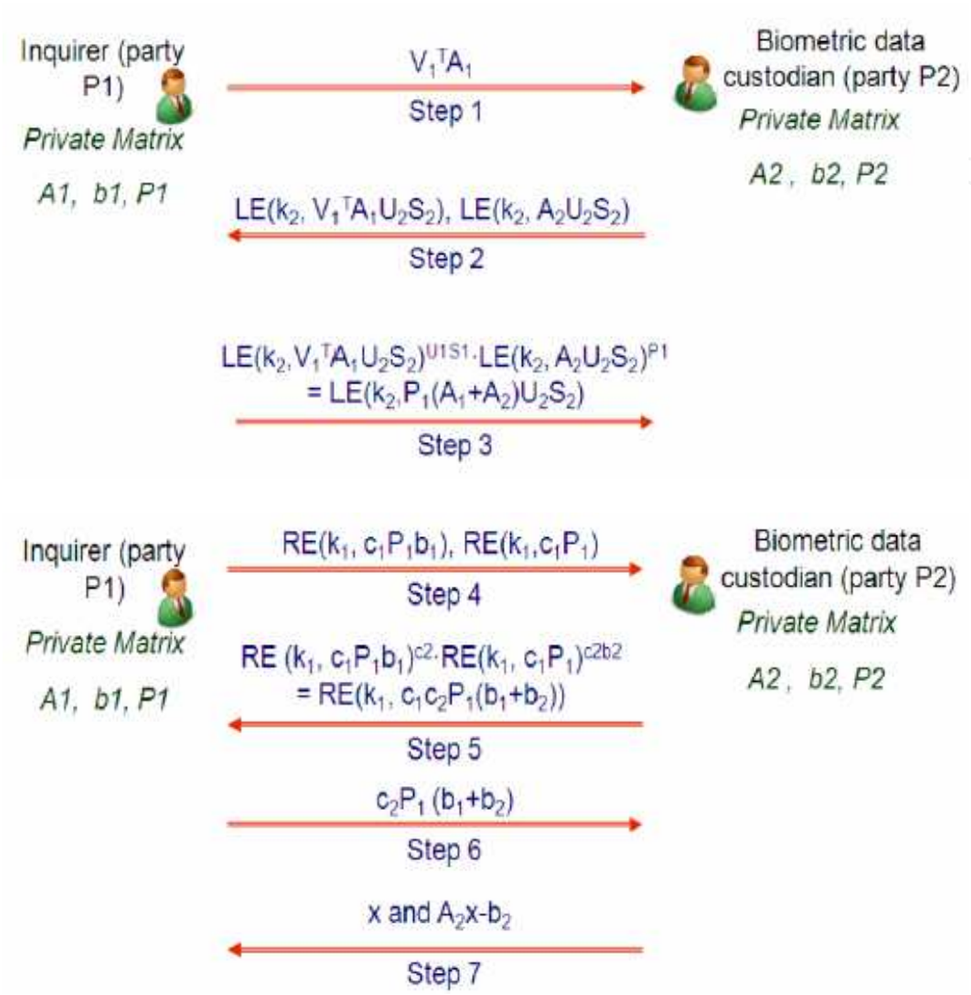


Fig. 2. PPCSC Illustration

The boundary vector x in step 3 will be derived under PPCSC. The basic idea behind the secure computation protocol for PPCSC required in step 3 for solving $(A_1+A_2)x = b_1 + b_2$ is to solve $P_1(A_1+A_2)P_2 y = P_1(b_1+b_2)$; where ($P_1$, A_1, b_1) are private to P1, and (P_2, A_2, b_2) are private to P2. Note that even if P2 knows $P_1(A_1+A_2)P_2$ and $P_1(b_1+b_2)$, P2 can only derive y but could not know A_1, b_1, and P_1; thus the privacy of A_1, b_1, and P_1 for P1 is preserved. Once y is solved, P2 can derive $x = P_2 y$ and sends it to P1. Note that any adversary intercepting or sniffing from the network the value of x cannot derive De or Dv unless the adversary also has either (A_1, b_1, V^2_1) or (A_2, b_2, V^1_1).

During the process of secure computation, A_1, b_1, A_2, b_2 are never exposed individually except $P_1(A_1+A_2)P_2$ and $P_1(b_1+b_2)$ for P2. Therefore, it is information-theoretic secure; i.e., the privacy of (A_1, b_1) for P1, and the privacy of (A_2, b_2) for P2, is guaranteed. Readers interested in the step-by-step details on the secure computation mechanism for solving x in $(A_1+A_2)x = b_1+b_2$ are referred to our articles elsewhere (Sy 2009b) (Sy 2009c).

4. Experimental studies on SIPPA parameters & Usability for biometric security

For proof-of-concept, we conducted experimental studies to better understand the potential of SIPPA for biometric applications. This experimental study consists of three parts. The first part is a simulation study targeting the specific parameters to investigate their inter-relationship. This experimental study aimed at tackling the following two questions:

1. How is the quality of the estimate on the closeness between De and Dv affected by the dimension of x?
2. How is the closeness between De and De' (estimated De) affected by (a) $|V^2_1-x|$, and (b) dimension of x?

The second part is an application of SIPPA to the private reconstruction of digital images, and the third part is the private reconstruction of the voiceprint.

4.1 Experimental study part 1: SIPPA parameters

In the simulation study, we generated 5 test data sets categorized by different dimensions. The vector dimensions in these five data sets are 5, 10, 20, 40, and 60 respectively. In each data set, 10 pairs of client (Dv)/server (De) vectors are generated, thus totaling 50 pairs for all dimensions. Figure 3 shows the normalized data difference |De-Dv|/Dim in y-axis and the normalized eigenvector difference $|V^2_1 - V^1_1|$/Dim in x-axis. Figure 4 shows graphically the relationship between the normalized deviation measure $|V^2_1 - V^1_1|$/Dim and $|V^2_1 - x|$/Dim.

Figure 3 shows an approximately linear relationship between the deviation of the client (Dv)/server (De) and that of the corresponding eigenvectors. The degree of closeness between two sources of data is reflected by the closeness between the corresponding eigenvectors. In other words, the client with Dv and the corresponding eigenvector V^1_1 will be able to deduce the degree of closeness between Dv and De if the similarity distance as measured by 2-norm $|V^2_1 - V^1_1|$ is known. And if the degree of closeness between Dv and De is known, it provides a basis for reconstructing De.

Figure 4 also shows an approximately linear relationship in relatively lower vector dimension (≤ 20) between the degree of closeness of the eigenvectors (of the client and server), and that of the server (thus the client in an inversely proportional sense) and the boundary vector x. Consequently, if the dimension of the data vector is relatively low (i.e., ≤ 20), then the degree of closeness as measured by the 2-norm $|V^1_1-x|$ or $|V^2_1-x|$ can act as a predictor for $|V^2_1 - V^1_1|$.

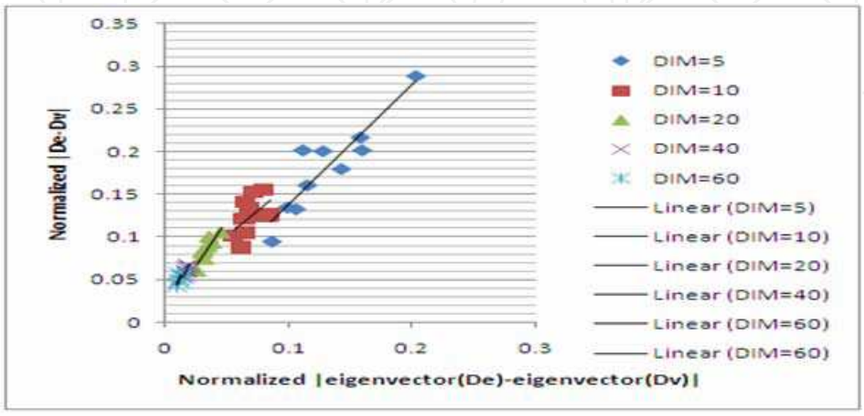


Fig. 3. Data deviation vs. eigenvector deviation

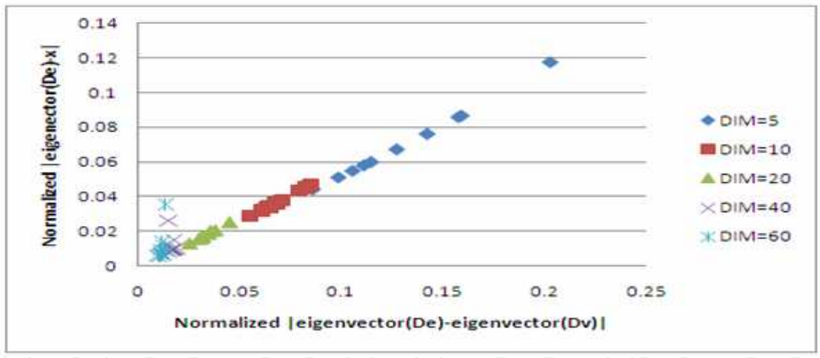


Fig. 4. Goodness of fit of the difference estimator

Based on the result of the simulation study, we chose the data dimension to be 10 in the second part of our experimental study.

4.2 Experimental study part 2: Biometric face image

In the SIPPA experimentation with digital images, three doll faces and one real person image containing biometric face information were used. All images are stored in PGM (Portable Gray Map) format. In the experimentation using doll faces, Figure 5 shows the result of the experimentation. The first row shows two black-and-white original images of two Barbies. The first column shows the reference sample images consisting of two other Barbies, and a generic blonde hair doll. The resolution of each image is 64x85 with 255 gray-level.

Altogether six doll faces are reconstructed from every combination pair of a reference image and a sample image. In the magnified version of Figure 5, the best reconstruction is from Barbie-to-Barbie, while the worst reconstructions are the two on the last rows.

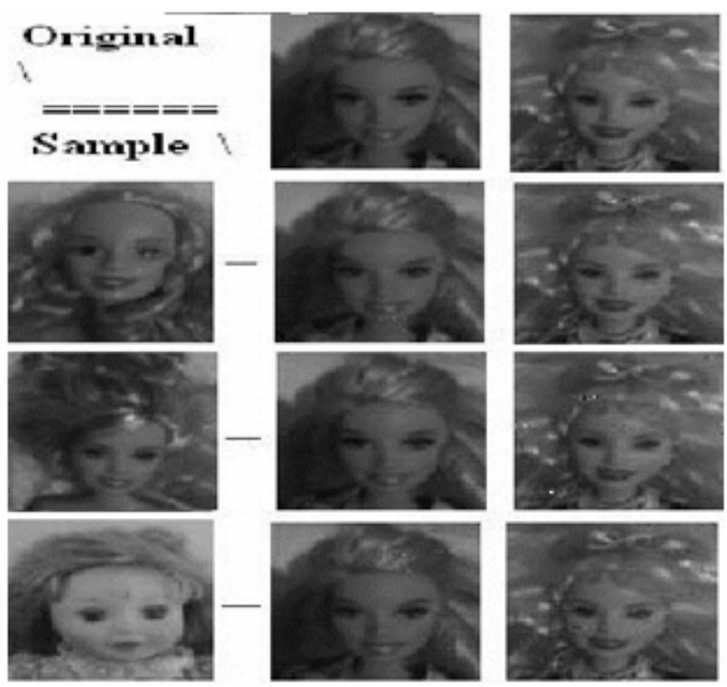


Fig. 5. SIPPA application to doll's face

In the experimentation using real human images, the server side utilizes a black-and-white image with 255 gray levels (Figure 6). The resolution of the image is 256x192. The linearization of the image results in a 49152 (=256x192) x 1 vector. This is about 9 times larger in comparing to the linearization of the doll images, which have a vector size of 5440 (=64x85) x 1. Obviously the dimensions of the linearized vector in both cases are beyond the optimal performance range of SIPPA. As such, the 49152x1 (or 5440x1) linearized image vector is split into multiple 10x1 vectors. SIPPA is then applied iteratively to reconstruct the original image.

During the reconstruction, two parties − referred to as client and server − will participate in a SIPPA session. The client has an image, referred to as a sample image. The server has an image, referred to as a source image. In the SIPPA session, each party takes a portion of the linearized image sequentially in the form of a 10x1 vector to construct a symmetric matrix, and derives the eigenvalue/eigenvector of the matrix. Then both parties participate in a secure computation protocol for PPCSC as described in step 3 of SIPPA in the previous section.

The outcome of PPCSC is the boundary vector x. The client party then uses the boundary vector x and the client side 10x1 linearized vector of the sample image to reconstruct the server side 10x1 linearized vector of the source image. The only information that the server provides is the helper data composed of the eigenvalue of the server side 10x1 linearized source image and a scalar. The original 10x1 linearized source image is never shared with the client party. Once the reconstruction of the 10x1 server side linearized source image is completed, the SIPPA process repeats for the next block of the 10x1 linearized vector until all 10x1 image blocks are processed. The entire source image is then reconstructed by the client party using the estimated 10x1 source image blocks.

Fig. 6. Source image

Fig. 7. Similar sample image

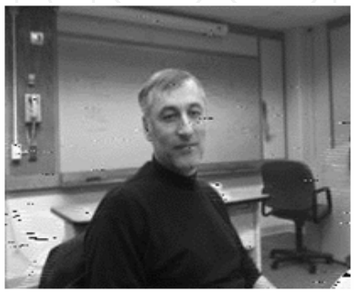


Fig. 8. Reconstruction using similar image

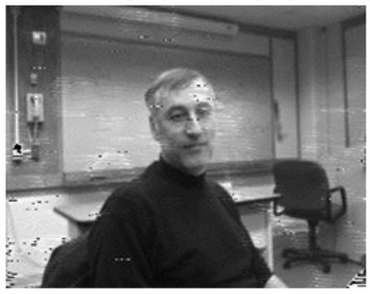


Fig. 9. Reconstruction using dissimilar image

The SIPPA reconstruction of the entire human source image is conducted twice in this experiment. In the first trial, the client side provides a sample image shown in Figure 7 that is sufficiently similar to the server side source image as shown in Figure 6. All images have a gray scale of 255 and have the same resolution 256x192.
The outcome of the reconstruction by applying SIPPA based on the server side source image (Figure 6) and the client side sample image (Figure 7) is shown in Figure 8. During the reconstruction, the source image is never shared with any party except the corresponding information used in the SIPPA processing.
During the second trial, a sample image of a different human subject (not Figure 6 or 7) is used on the client side for SIPPA. The image of the other human subject is not shown due to the restriction on the human subject clearance. The outcome of the reconstruction is shown in Figure 9, which shows a poorer quality in comparison to Figure 8.
By visual inspection and using Figure 6 as a reference, the face biometrics in Figure 8 is preserved better than that in Figure 9 – when the client side sample image is closer to that of the server side source image (Figure 6).

4.3 Experimental study part 3: Utilizing SIPPA in voice biometrics

In this part, a study of SIPPA utilizing voiceprints is conducted using the speaker verification system reported in our paper elsewhere (Sy 2009b). The objective of the experimental study is to study SIPPA in terms of its ability to reconstruct voiceprints that can cause the speaker verification system to behave in the same way as if the original voiceprints of its users are applied to the system. Twenty four speakers of different native languages participated in the experimental study. Altogether 118 different viable True User attempts and 101 different viable Impostor attempts were used to evaluate the system as described in the following experiments.
After filtering the instances of FTE (Failure to Enroll) and FTA (Failure to Acquire) due to noise introduced by phone devices and background environment, each voiceprint for verification and the enrolled voice template are used by SIPPA to reconstruct the voiceprint of a speaker for the speaker verification system; i.e., the speaker does not present his/her voiceprint to the speaker verification system. The distance between the enrolled voice template and the voiceprint reconstructed by SIPPA - abbreviated by KL-dist(enroll, sippa(VectorDim)), is computed. VectorDim specifies the SIPPA vector dimension. For the control experiment, the distance between the enrolled voice template and the verification voiceprint – abbreviated by KL-dist(enroll,verify) is also computed. Kullback–Leibler divergence is the distance function used to calculate a similarity score in this case, as described in our paper elsewhere (Sy 2009b).
The system behavior characterized by ROC using original speaker voiceprints and SIPPA reconstructed voiceprints are shown in Figure 10 for a comparison purpose. In this experimentation we deliberately set the *pre-defined threshold* as stated in step 5 of the SIPPA algorithm in the previous section to be infinity. In doing so, the "usability" of SIPPA is the highest, while the performance is expected to be the lowest when compared with the cases where the pre-defined threshold is used to filter the cases where the source and sample data are significantly different. In other words, all the voiceprints reconstructed by SIPPA were used in the derivation of the ROC irrespective to the closeness of a reconstructed voiceprint and its reference voiceprint.

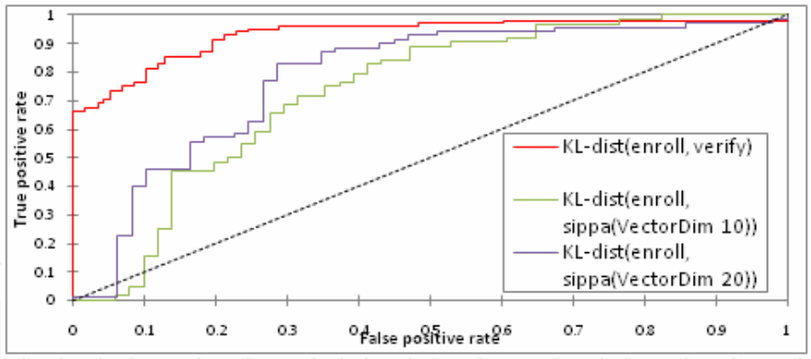


Fig. 10. ROC with threshold as infinity

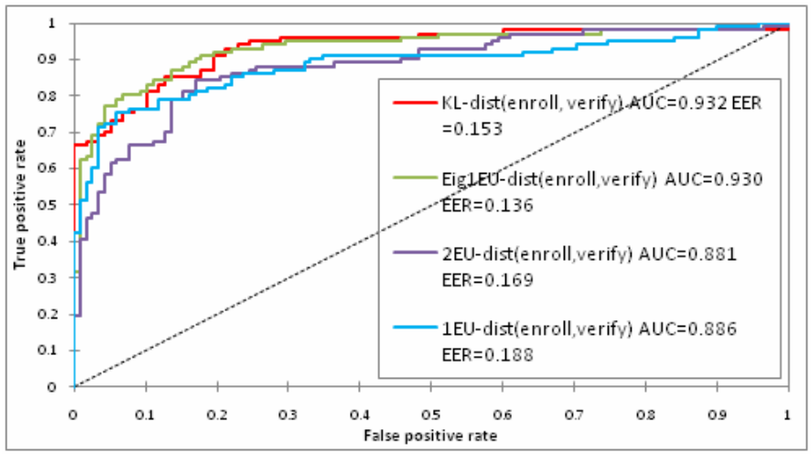


Fig. 11. ROC using original enroll voiceprint

4.3.1 Other Distance Functions more suitable for SIPPA reconstructions

Since SIPPA is Eigen-Based and utilizes Euclidean Distance of Eigenvectors to estimate the similarity between datasets, we tried to determine if an Eigen, Euclidean based distance function may be used to yield similar performance to the KL-Distance mentioned above. If such a distance function can be found, SIPPA reconstructions may be used more successfully in voice biometrics. The essential problem here is obtaining a distance function that performs as-good-as/better than the KL-Distance function used in our voice system described elsewhere (Sy 2009b), while also not being adversely affected by "noise" that may be generated by SIPPA reconstructions of the Verification voiceprint.

As described elsewhere (Sy 2009b), the voiceprints used by our system are created by extracting Mel-Spectrum features from a wav file (using 8KHz sampling rate) to derive a 20X1 mean vector and a 20X20 covariance matrix of the corresponding multivariate Gaussian model. The 20X1 mean vector and 20X20 Covarience matrix are the basis used in all the varied Eigen, Euclidean based distance functions that we evaluated. Their performance relative to the KL-Distance described above is characterized by ROC curves shown in Figure 11. Their performance with SIPPA reconstructions, where SIPPA threshold is set to infinity is plotted in Figure 12. Table-2 describes these distance functions. All ROC curves were generated by utilizing the same dataset.

As observed in Figure 11, the voice system using Eig1EU-dist distance function performs as good as that using the KL-dist function on the same dataset. The Eig1EU-dist function leads to an EER of 0.136 with an AUC (Area Under Curve) of 0.930, while the KL-dist function leads to an EER of 0.153 with an AUC of 0.932. What is interesting about the Eig1EU-dist function is the fact that even when utilizing SIPPA reconstructions (SIPPA threshold at infinity) of Verification Voiceprints, the performance of the voice biometric system is not affected in too adverse a manner as seen in Figure 12. The system configured with Eig1EU-dist distance function performs as good as one configured with the KL-Distance function without SIPPA i.e. KL-distance comparing Original Enroll and Sample Voiceprints.

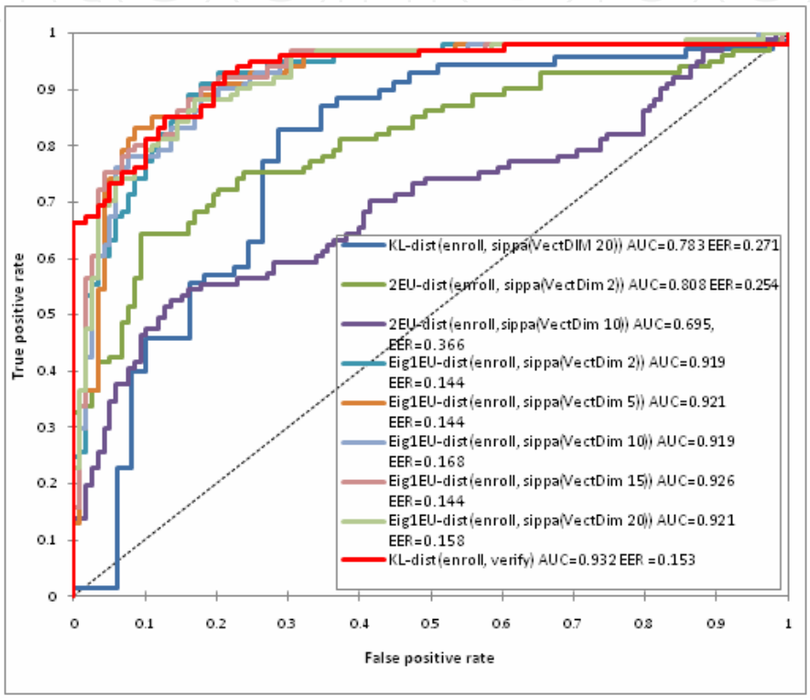


Fig. 12. ROC using original enroll and SIPPA

4.3.2 SIPPA privacy protection & thresholds

When a person interacts with a voice verification system, the privacy of the person and his/her voiceprint can be protected through SIPPA. Specifically, the user may engage in a SIPPA session as a SIPPA server, and chooses to send a close approximation of a sample verification voiceprint only if the verification system has a reference voiceprint of the person. The person can choose to set her/his SIPPA threshold to infinity. However, this might not provide effective privacy protection across different biometric template standards and modalities. Setting a SIPPA threshold provides for finer control. The basic idea is to utilize the one piece of information that the SIPPA server knows about the SIPPA client - Vector X. If one can devise a distance function that provides an ROC similar to the ROC using a KL-distance function for comparing the *Original Enroll* and *Original Verify* voiceprints, the SIPPA server can set an effective threshold which can be customized intelligently for varied levels of privacy protection. The essential question here is, by knowing the vector X, how effectively (based on some distance function) can the SIPPA

Distance Function	Description
KL-dist	Distance function described elsewhere (Sy 2009b)
Eig1EU-dist	Similarity score calculated by: Obtaining a symmetric matrix by multiplying the 20X1 multivariate mean vector (stored in each of the voiceprints being compared, as detailed earlier) with its transpose. Symmetric Matrix (**Me)** is obtained from the enrolled voiceprint and Symmetric Matrix (**Mv**) is obtained from the verify voiceprint. Obtaining 20 symmetric matrices by multiplying each column of the 20X20 covariance matrix (stored in each of the voiceprints being compared, as detailed earlier) with its transpose. 20 Symmetric Matrices (**Ce1 ... Ce20)** are obtained from the enrolled voiceprint and a further 20 Symmetric Matrices (**Cv1 ... Cv20)** are obtained from the verify voiceprint. Obtaining the largest Eigenvalue and its corresponding unity-normal Eigenvector for the Symmetric Matrices **Me,Mv,Ce1...Ce20,Cv1...Cv20 —** producing Eigenvectors **MEe**,**MEv**,**CEe1...CEe20**,**CEv1...CEv20**. Obtaining 2-Norm Euclidean distances for **|MEe-MEv|, |CEe1-CEv1|...|CEe1-CEv1|** The Similarity score (i.e. the output of this distance function) is the Geometric Mean of the 21 2-Norm Euclidean distance values calculated in the previous step.
2EU-dist	Similarity score calculated by: Obtaining the 2-Norm Euclidean distance**(2EM)** between the 20X1 Enroll-Voiceprint mean vector and the 20X1 Verify-Voiceprint mean vector. Obtaining two 400X1 vectors by converting the 20X20 covariance matrices (in each of the Voiceprints being compared) into a vector by essentially copying values from the first column of the matrix into the first 20 values of the vector and then obtaining values from the second column of the covariance matrix into the next 20 values of the vector and so on. One 400X1 vector (**Ve**) is obtained from the Enroll Voiceprint and one 400X1 vector (Vv) is obtained from the Verification Voiceprint. Obtaining the 2-Norm Euclidean distance**(2EC) for |Ve – Vv|**. The Similarity score i.e. the output of this distance function is the Geometric Mean of **2EM** and **2EC**
1EU-dist	Similarity score calculated by: Obtaining the 1-Norm Euclidean distance**(1EM)** between the 20X1 Enroll-Voiceprint mean vector and the 20X1 Verify-Voiceprint mean vector. Obtaining two 400X1 vectors by converting the 20X20 covariance matrices (in each of the Voiceprints being compared) into a vector by essentially copying values from the first column of the matrix into the first 20 values of the vector and then obtaining values from the second column of the covariance matrix into the next 20 values of the vector and so on. One 400X1 vector (**Ve**) is obtained from the Enroll Voiceprint and one 400X1 vector (Vv) is obtained from the Verification Voiceprint. Obtaining the 1-Norm Euclidean distance **(1EC) for |Ve – Vv|**. The Similarity score (i.e. the output of this distance function) is the Geometric Mean of **1EM** and **1EC**

Table 2. Tested distance functions described.

server/client determine if both voiceprints being compared are from the same person, or if they are from two different persons.

To illustrate, let's consider that a user would like to verify his voiceprint with a verification system. The user assumes the role of a SIPPA server while the verification system assumes the role of a SIPPA client. Using Mel spectrum as discussed (Sy 2009b), each voiceprint is a 420X1 vector. 21 SIPPA sessions are required with each session assuming the responsibility for reconstructing one 20X1 vector. Each of the SIPPA server sessions obtains the vector X, and computes the 2-norm Euclidean distance between their own eigenvector and vector X. The user at this point can utilize these 21 2-norm Euclidean distance scores to determine if her/his voiceprint is sufficiently similar to the reference voiceprint of the verification system (SIPPA client). If they are sufficiently similar, indicating the voiceprints may be from the same person, the user may opt to send helper data to the SIPPA client; i.e. the verification system.

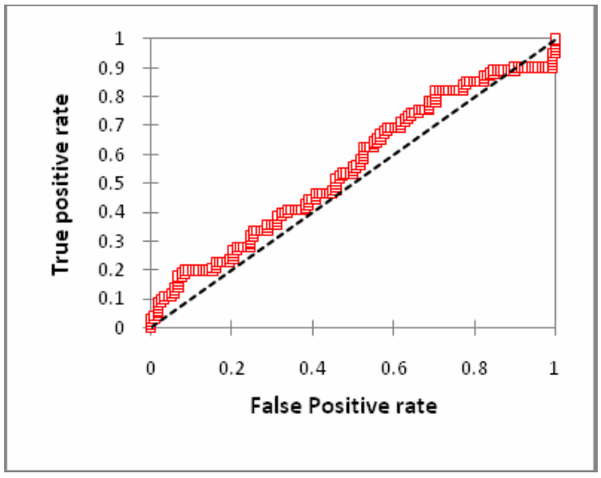


Fig. 13. Max Function for Calculating Distance

An ROC curve can be used to examine the quality of a distance function which produces a Similarity Score based on these 21 2-norm Euclidean distance scores, or in general the 21 SIPPA Server Eigenvectors and the 21 X vectors. We can generate the ROC by varying the SIPPA threshold. Specifically, the SIPPA threshold, in conjunction with Similarity Score produced by a distance function, can be used to decide whether the two voiceprints may be from the same person, or two different persons; thus an instance of true/false positive. If SIPPA server relies on a distance function that merely uses the maximum of the set of 21 2-norm-Euclidean distance scores as a distance value, and, say, the distance function results in a ROC as in Fig. 13, then the voice system will have behaved no better than flipping a coin to determine whether the voiceprints are from the same person.

The performance of various distance functions producing a similarity score for the two voice prints being compared utilizing only the SIPPA server Eigenvectors and X vectors are shown in Figure 14. Table 3 describes these distance functions. To reiterate, all ROC curves in this chapter were generated by utilizing the same dataset.

4.3.3 Dual SIPPA, verification-engine threshold

The ROC curves at Figure 15 show the behavior of the Voice-Biometric system described elsewhere (Sy 2009b) when two different distance functions were used; namely, a two-

threshold SIPPA based distance function and KL-distance function. In this case, the Voice-Biometric System is the SIPPA client. The ROC of the system employing the two- threshold SIPPA distance function is calculated in the following manner: Let's consider that **n** denotes the set of possible SIPPA thresholds we would like to evaluate the system on and **m** denotes the set of Voice-Biometric system thresholds we would like to evaluate the system on. The total possible combinations of thresholds are **m*n**. Hence the ROC will contain **m*n** data points. For each possible combination of thresholds, either a True-Negative, or a False-Negative, will be counted if the SIPPA threshold leads to not sending helper data. If SIPPA server (i.e., the user) does send helper data according to the (SIPPA) threshold, then the user voiceprint will be reconstructed by the SIPPA server (i.e., the system) and compared against the template. The comparison will then yield a count of either True-Negative, True-Positive, False-Positive or False-Negative count.

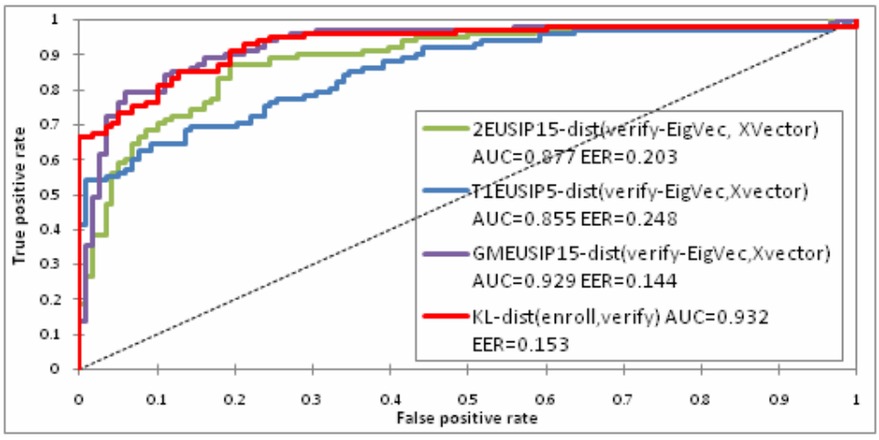


Fig. 14. ROC of different distance functions.

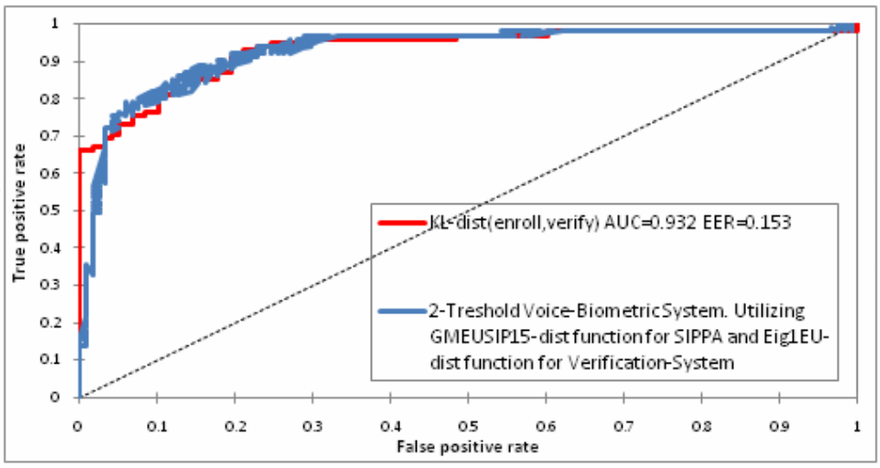


Fig. 15. ROC of 2- threshold with SIPPA

Deducing from the plots in Figure 15, adopting the two-threshold SIPPA based distance function allows the system to perform as good as one based on KL-distance. But the two-threshold SIPPA based distance function allows privacy preserving authentication without

Distance Function	Description
KL-dist	Distance function described elsewhere (Sy 2009b)
2EUSIP15-dist	Similarity score calculated by: 1. SIPPA vector dimension is 15X1; therefore 28 SIPPA Sessions are required to process the two Voiceprints. Generating 28 15X1 SIPPA Server Eigenvectors(**SE1...SE28**) and 28 15x1 X vectors (**X1...X28**) 2. 2-NormEuclideanDistance is obtained for |**SE1-X1**|...|**SE28-X28**| producing 28 Euclidean distance values. 3. Similarity score is obtained by calculating the mean of the 28 Euclidean distance values calculated above.
T1EUSIP5-dist	Similarity score calculated by: 1. SIPPA vector dimension is 5X1; therefore 84 SIPPA Sessions are required to process the two Voiceprints. Generating 84 5X1 SIPPA Server Eigenvectors(**SE1...SE84**) and 84 5x1 X vectors (**X1...X84**) 2. 2-NormEuclideanDistance is obtained for |**SE1-X1**|...|**SE84-X84**| producing 84 Euclidean distance values. 3. Similarity score is obtained by calculating the t-value utilizing the one sample t-test with a population mean of 0.
GMEUSIP15-dist	Similarity score calculated by: 1. SIPPA vector dimension is 15X1; therefore 28 SIPPA Sessions are required to process the two Voiceprints. Generating 28 15X1 SIPPA Server Eigenvectors(**SE1...SE28**) and 28 15x1 X vectors (**X1...X28**) 2. 2-NormEuclideanDistance is obtained for |**SE1-X1**|...|**SE28-X28**| producing 28 Euclidean distance values. 3. Similarity score is obtained by calculating the geometric-mean of the 28 Euclidean distance values calculated above.

Table 3. Details of SIPPA distance functions plotted in Figure 14

compromising accuracy! Given the promise of this result, it will be worthwhile to conduct a large scale study to quantify the characteristics of various distance functions described in this section, and to observe how SIPPA behaves with respect to each of them.

5. Standard-based parallel SIPPA design and implementation

The experimental study in the previous section is focused on the accuracy performance. Currently SIPPA would need parallel processing, referred to as Parallel SIPPA, in order to deliver real time computation performance on image processing. The essential idea behind Parallel SIPPA is the creation of multiple SIPPA-Client and SIPPA-Server instances. The parallel SIPPA architecture is summarized in Figure 16. SIPPA realized under parallel computing can produce faster results. In parallel SIPPA, a Server Application (SA) initializes multiple instances of SIPPA-Server in different physical/virtual machines. As SIPPA-Server instances are created, their contact information is added to a queue maintained by SA. In dealing with digital images in PGM/PPM file format, SA linearizes the file into multiple

10X1 vectors where each vector is used to produce a symmetric matrix described in step 1 of the SIPPA algorithm reported in the previous section.
All SIPPA-Server instances have access to any of these symmetric matrices based on a matrix's unique id. When the Client Application (CA) starts, it takes possession of a sample client PGM/PPM image and linearizes the file into multiple 10X1 vectors; where each vector is used to produce a symmetric matrix in a similar fashion as the server side.
After CA contacts SA, SA de-queues an element and sends the contact information of SIPPA-Server instance to CA. CA continuously creates a new SIPPA-Client instance on available SIPPA-Server instances until all 10X1 vectors have been assigned resulting in parallel processing. CA provides every SIPPA-Client instance with an array of symmetric matrices and a chosen SIPPA-Server instance's contact information. This SIPPA-Client instance and its assigned SIPPA-server instance now engage in a sequential SIPPA session, processing each symmetric matrix at a time. As the SIPPA-Server/Client pairs finish processing their allotted set of matrices, the SIPPA-Server instances are added back to SA's queue and the SIPPA-Client instance is destroyed.
The reconstructed 10X1 vector produced by each SIPPA-client/SIPPA-Server pair is sent back to CA. When the entire set of 10X1 vectors is completed, all these vectors are combined to reconstruct an approximation of the server side image by CA.

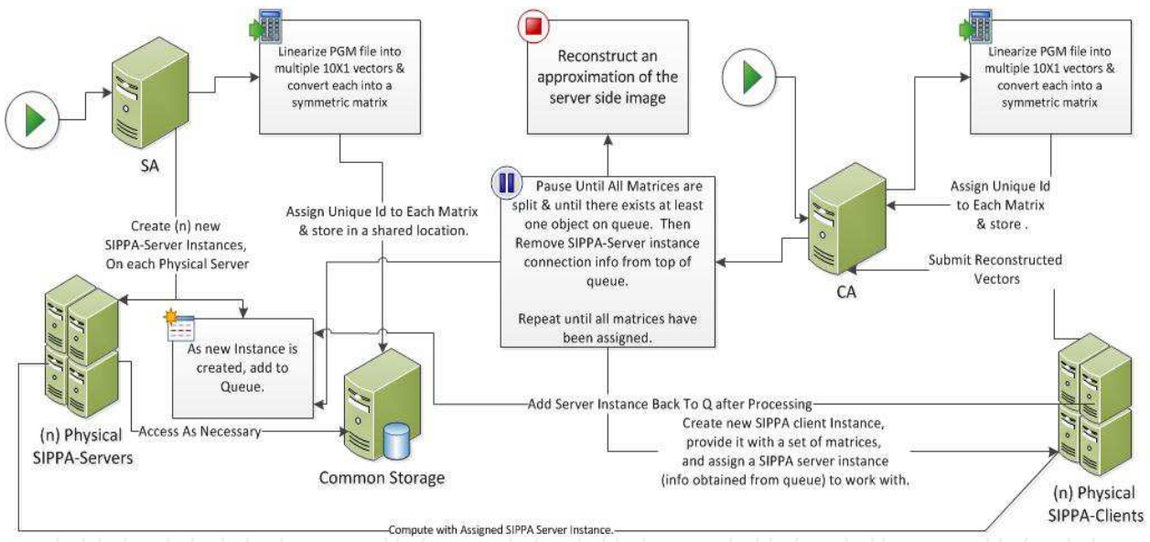


Fig. 16. Parallel SIPPA system delineation

5.1 Parallel SIPPA experimental results

In this experimental study we try to determine the optimal configuration for Parallel SIPPA i.e. the number of physical/virtual machines that need to be assigned for SIPPA-Servers and SIPPA-Clients respectively, and the number of SIPPA-Server instances that need to be created initially by SA (Server Application).
Performance of the SIPPA system is highly dependent on the degree of parallelism that can be obtained with the available hardware under which SIPPA operates. In our tests, we experiment with various platforms, both sequential and parallel; to pinpoint increases in performance as well as where our system is bottle-necked. By sequential we mean one instance of SIPPA-Server and SIPPA-Client runs at any given time in the entire system, and

the 10X1 vectors are processed one at a time, whereas by parallel we mean multiple instances of SIPPA-Server and SIPPA-Client run at any given time processing multiple 10X1 vectors at any given time. Our testing configurations included: Single Core virtual machines (VM), Dual Core physical machines (DC), and a Quad Core machine (QC).

To obtain 90 - 100% CPU utilization on all physical/virtual machines (both server and client), a proper client to server ratio must be applied. This is because SIPPA-client instances have an additional workload when compared with SIPPA-Server instances; i.e. they obtain their matrices dynamically from CA and then store them locally for processing. This additional network I/O coupled with the fact that they also have to reconstruct the server side matrix makes the Client Machines consume 6 times more resources than a Server machine. With a naïve parallel approach of using one physical/virtual machine as a server for every physical/virtual client machine, performance is improved in comparison to a sequential SIPPA approach. As we increase the client to server ratio by decreasing the amount of servers, the servers are more efficiently used which results in a dramatic decrease in completion time and full utilization of the CPU.

The data shows where our performance increase lies and where possible bottlenecks are. As seen in Figure 17, even when parallel SIPPA is run on a single quad core machine, a performance increase of 1600% is obtained when compared with using Two Dual Core machines engaging in SIPPA sequentially. Also notable is the performance increase by switching from the naïve parallel approach to an optimized parallel approach. With the VMs we had an increase of 75% in performance when comparing a naïve parallel approach to an optimized parallel approach. With the large data set experiment on the physical machines, we noticed an increase of 350%.

During Parallel SIPPA startup, there is an initialization and job distribution phase; this seems to take anywhere between 30 and 60 seconds depending on the size of image. Due to this, processing small images with a resolution 256x192 by parallel SIPPA do not seem to benefit much from increasing the number of physical/virtual machines within a certain setup.

In parallel SIPPA when an optimized setup is doubled and presented with an image of high resolution, we notice a reduction of processing time by about 40%; i.e., when the total number of machines is doubled from 8 (# 7) to 16 (# 6), the processing time for a 640X480 image decreases from 26 min to 15 min. We observe a similar improvement in the case with virtual machines (# 10 and # 9). Doubling of resources reduces processing time by approximately 40%. This indicates that given a reasonable size cluster of professional level servers, most of the typical biometric images could be processed in a Parallel SIPPA framework within a reasonable amount of time.

5.2 BioAPI Standard based SIPPA Implementation

When job requests are presented to Parallel SIPPA (PSIPPA), PSIPPA can achieve near real time performance as shown in the experiment results in the previous section. While the near real time results are noteworthy, our aim is to expand PSIPPA towards a true interoperable system. To achieve this we integrate PSIPPA into a BioAPI system which results in *BioSIPPA*. By integrating PSIPPA into a BioAPI 2.0 framework (May 2006), PSIPPA is made available in an interoperable environment allowing multiple users on various platforms to access PSIPPA through standard based service component. In order for BioSIPPA to handle simultaneous job requests from different users, we also investigate a Slice Based Architecture (SBA) design to optimize the resource utilization of *BioSIPPA*.

#	Machine Setup
1	Two DC's (Dual Core-3.2GHz, 2.75GB RAM).
2	Two VM-512's.(Virtual Machine-1GHz,.5G RAM).
3	One QC. (Quad Core Machine-3.2GHz, 4G RAM).
4	Eight DC's Running multiple SS(SIPPA-Server) Instances and Eight DC's Running multiple SC(SIPPA-Client) instances.
5	Five VM-512's Running multiple SS Instances and Five VM-512's Running multiple SC instances.
6	16 (2 used for SS and 14 used for SC) DC's Running Multiple Instances Of SS and SC.
7	8 (1 used for SS and 7 used for SC) DC's Running Multiple Instances Of SS and SC.
8	10 (2 used for SS and 8 used for SC) VM-512's Running Multiple Instances Of SS and SC.
9	10 (2 used for SS and 8 used for SC) VM-1G's Running Multiple Instances Of SS and SC.
10	5 (1 used for SS and 4 used for SC) VM-1G's Running Multiple Instances Of SS and SC.
Engaging in sequential SIPPA. Optimized Parallel SIPPA where all CPU's are maintained at 100% usage for most of the time. Naïve Parallel SIPPA, Equal number of Physical/Virtual Machines For SS & SC.	

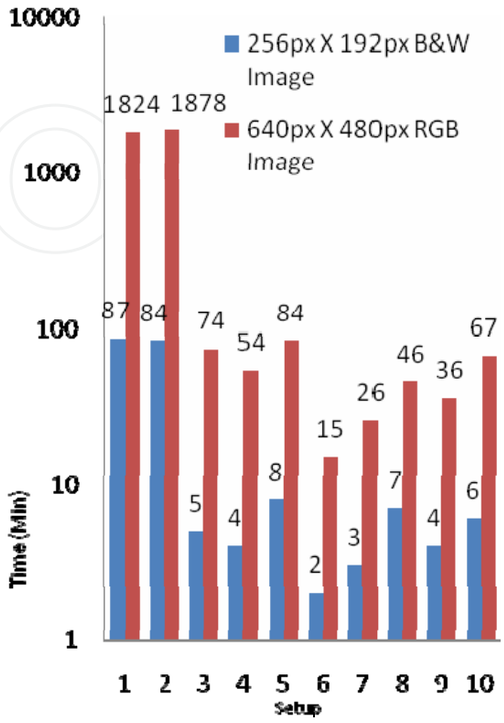


Fig. 17. Parallel SIPPA processing time performance

At the BioAPI level, realizing PSIPPA as a BioAPI service component involves a design and structural definition for various BioAPI framework components. At the minimum, we need to determine the appropriate level of PSIPPA instantiation in the BioAPI framework, and to define the corresponding framework components including Biometric Service Providers (BSP), Biometric Function Providers (BFP), and various BioAPI Units managed by the BSP or BFP such as Processing Algorithm Units (PAU) or Archive Units (AU). As defined in the BioAPI 2.0 standard, the PSIPPA schema and access interfaces are registered with the BioAPI component registry. This allows end users to search for our service.

Our targeted SBA-BioSIPPA system is composed of various elements that are the basis for its interoperable environment. Specific to biometric system development, the most atomic element in the SBA proposed in this project is *resource*. A resource could be a Java functional method exposed for an external call via Java RMI, a biometric software/hardware interface or a software/hardware processing unit for vector processing. A collection of resources for

the purpose of completing a specific task (e.g., enrollment) is referred to as an *aggregate*. An aggregate only defines the set of resources; it does not define how the resources of an aggregate should be used. The concept of a service can be thought of as one or more aggregates with additional information on how the resources may be used to accomplish a specific task. Lastly, a service *slice* is essentially one or more aggregates provisioned by a slice manager. A slice manager coordinates how these aggregates may be utilized; e.g., by whom, for how long, under what pre-conditions or constraints on bandwidth, memory, CPU, or storage size. In the application of SBA-BioSIPPA, the slice manager drives the provisioning of slices which creates the necessary element for a true parallel and interoperable environment.

An objective of SBA-BioSIPPA is to expose PSIPPA as an interoperable service component for a BioAPI 2.0 framework (May 2006). As a long term goal, we also aim for an easy adaptation of the service component for the standards developed for other application domains; e.g., ISO 19092:2008. Our strategy towards staging the PSIPPA service component for flexibility and adaptation is to employ RMI technology, which provides a lightweight, multithreaded, reliable communication bridge between the BioAPI and PSIPPA systems.

As in most biometric systems, SFA-BioSIPPA utilizes various resources similar to a face recognition system that has a Sensor Unit, a Matching Unit, and an Archive Unit. In SFA-BioSIPPA each necessary resource may be supplied by a different vendor and we provide a means for an end user to choose among the various resources.

To illustrate the utilization of SFA-BioSIPPA, consider a user A requests for privacy preserving verification using SIPPA. User A will interact with the Slice Manager, which queries the Component Manager. The slice manager then displays what resources are available to execute the job request. User A, with the assistance of the Slice Manager, chooses the appropriate resources for the job and sends a request for a slice allocation. The request is recorded in an internal registry within the Slice Manager and then sent to the Component Manager. The Component Manager accesses the Resource Management Database (RMDB) and acquires the details of resources specific to the slice request. The resources are accessed through their BSP, at which point the slice is in its initiation stage.

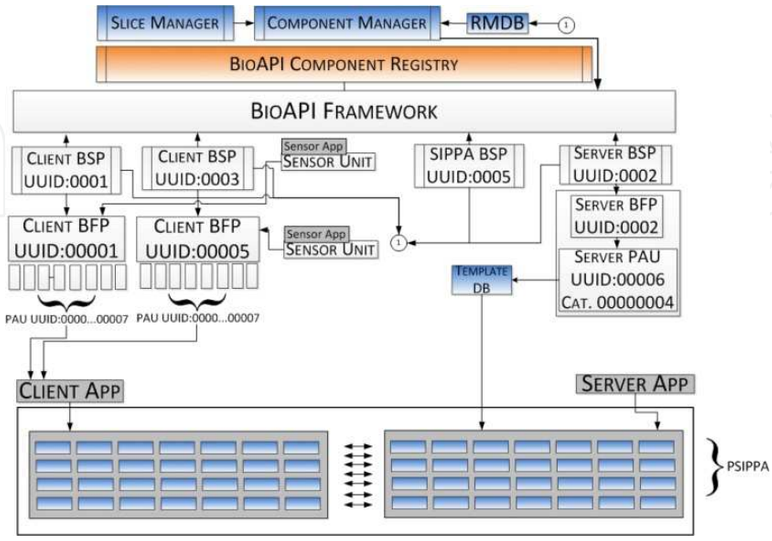


Fig. 18. SIPPA Deployment for BioAPI Framework

When a resource is accessed through its BSP, the BSP will initiate execution and immediately update a record in the RMDB incrementing the number of jobs executing on the particular resource. On the termination of a job on each resource, the resource's BSP will again update the database decrementing the number of jobs currently running on the resource. Any changes in the RMDB will allow future queries from the Slice Manager to reflect a near real time status of the system. Note that SFA-BioSIPPA aims at a standard based framework to facilitate not only interoperability, but also a scalable computation environment for PSIPPA where additional computing resources for SIPPA could be easily added to the framework via the registration of a RMI client with remote RMI server acting as a SIPPA client or SIPPA server. This is essentially a kind of cloud computing model to harass the computing power from multiple sources for PSIPPA. In this cloud computing model one must be careful in the design of SFA-BioSIPPA in regard to the risk in information leak and collusion when multiple parties are involved in PSIPPA. An alternative to this without the same degree of the risk just mentioned is MPSIPPA (Massive Parallel SIPPA) that relies on the GPU technology.

5.3 MPSIPPA

SIPPA, a client server mechanism for protecting privacy, is not merely a scientific project intended to run on clusters/supercomputers. Its true wide-ranging purpose is to protect the privacy of individuals as they interact with other individuals and large central databases. Ideally SIPPA would most effectively protect privacy if SIPPA clients/servers were deployed among personal computers without the need for third parties. However, sequential SIPPA, when used for comparing two 640X480 RGB images shown previously, takes a prohibitive amount of time (well over 1800 minutes). PSIPPA, as shown above, with a cluster of 16 Dual-Core machines reduces this time to around 15 minutes.

PSIPPA, as mentioned before, requires a cluster. PSIPPA could potentially be bottlenecked by network data transfer issues. Since GPU's are designed for around 500 cores running at the same time, trying to access memory; they have an asynchronous memory read/write bandwidth of around 180GB/s. This helps to reduce the data transfer workload to only the necessary cross network data transfer required for 2-party secure computation. Given the widespread availability of GPU's, which might facilitate wider adoption of SIPPA as a privacy protection mechanism in households and to examine its usability/performance over PSIPPA, we proceeded to implement SIPPA in CUDA-C (Nvidia 2010).

As shown in section 4, the optimal vector dimension for SIPPA is less than 20, which leads to a typical SIPPA session on a 640X480 RGB PPM image consisting of 92160, 10X10 individual matrices. Since these individual matrices are small for utilizing GPU LAPACKS, we proceeded to convert required matrix operations like SVD, Eigen Decomposition, Matrix Multiplication etc. from standard C LAPACKS to CUDA-C. These converted C functions now can be called by each GPU thread to allow for the simultaneous execution of operations like SVD on many different matrices at the same time. Each thread on the GPU which is sequential, takes ownership of one of those 92160 matrices and performs the required matrix operations of SIPPA iteratively. Thousands of these threads are scheduled in parallel and execute in parallel on the many cores available on a GPU.

5.3.1 NVIDIA CUDA-C enabled GPU thread execution overview

Unlike CPU threads, there is negligible overhead in creating thousands of CUDA threads. In fact, for the GPU to function at anywhere close to 100% occupancy, tens of thousands of

threads must be scheduled. This is due to the fundamental problem of parallel computing; i.e. memory access. Although CUDA architecture allows for 500 cores executing threads in parallel, allowing random access to GPU memory for each of these parallel threads creates bottlenecks where a thread executing in one of the 500 cores needs to wait on the memory access queue. By scheduling tens of thousands of threads, CUDA swaps threads waiting for memory access with other threads which might perhaps use the core more effectively.

A thread in CUDA executes a specified function iteratively. In CUDA when a function is launched by the Host (CPU) on the Device (GPU), the Host specifies the number of threads that need to be scheduled to launch the same function in parallel. Every thread is assigned a unique ID by CUDA; these ID's can be of one, two or three dimensions. Threads are further organized into blocks, each of these blocks can have up to a three dimensional ID. In modern GPU's the Host can schedule to execute a CUDA-C function on around 65000 blocks, with each containing up to 1024 threads. Threads in a block have access to certain CUDA privileges among threads in the same block; e.g., threads in the same block can cooperate to share limited low latency memory on each of the streaming multiprocessors. This is particularly useful when the output of threads in a block is input to other threads in the same block, or if a certain set of data is shared among threads, threads in a block can also be synchronized and be forced to execute certain parts of code before proceeding. The key here is, by utilizing these ID's assigned by the CUDA scheduler, one can precisely direct each of these thousands of threads to access, process and deposit to unique locations in memory, or perhaps to precisely choreograph a certain pattern of interaction with certain other specific threads.

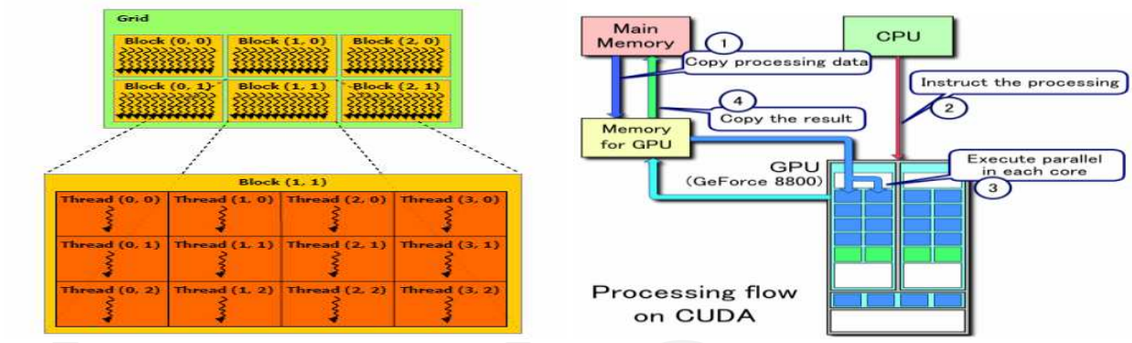


Fig. 19. Block, Thread Illustration (Kirk 2009)(Sander 2010)

5.3.2 Some notes on how we achieved MPSIPPA

To illustrate our process of achieving Massively Parallel SIPPA (MPSIPPA), perhaps it will be fruitful to describe how we implemented the simplest of our CUDA-C functions i.e. Vector Multiplication. As stated before, libraries for utilizing GPU's to perform matrix operations on multitudes of small independent matrices don't really exist. Our idea was, since usable SIPPA essentially leads to thousands of independent SIPPA sessions with each pair of sessions working on matrices ranging from 5X5 to 20X20, why not use the ability of modern GPU's to create and run thousands of parallel threads, and assign each of these threads ownership to one particular SIPPA session.

For the purposes of this example, let's assume that the Client would like to reconstruct a ppm image on the Server, and they both are using a candidate 640X480 RGB PPM image. Let's also assume that they settled on a vector size of 10X1(V). Processing this 640X480 RGB

PPM image would require 92160 independent client server SIPPA sessions. As the initial step, both the Client and Server machines read a particular candidate file in their disk drives into Host Memory (CPU Memory). All 92160 10X1 vectors are stored in a long one dimensional array, both on the client and server. On both client and server, The Host (CPU) then allocates a similar sized one dimensional array on Device (GPU) memory. Following allocation, the entire array of 921600 elements are copied from Host memory to Device memory. The Host also allocates on both Host and Device memory another long one dimensional array of size 92160*100 to store the results of $V.V^T$ (i.e. a 10X10 matrix), 100 values for each of the 92160 threads. Certainly a two dimensional or even a three dimensional array would be more intuitive. However, we noticed that a single malloc on the GPU, no matter the size of memory allocated, takes a constant amount of time. Allocating multiple smaller regions of memory and then storing their pointers in other arrays was too time prohibitive. Since threads can be arranged in two or ever more dimensions, their multi-dimensioned ID's can be used to specify what Vector they should work on.

```
VecMul<<<MAXTHR/THDSIZE, THDSIZE >>>(InputV, InputMat);
```

Once all required memory is allocated on the Host and Device, and Vector values copied from Host to Device, the Vector Multiplication function which multiplies the vector with its transpose to produce a 10X10 matrix for each of the required 92160 sessions is called by the host. Just shown above is the function call. The values inside "<<<>>>" specify the number of threads to create on the GPU, to execute the function in parallel. We surely require 92160 sessions, and therefore 92160 threads in total on the client and a corresponding number of 92160 threads on the server side; i.e. each thread assumes all computation iteratively for one SIPPA session. The first value in the angle bracket specifies the number of blocks to create, while the second value specifies the number of threads per block to create. Certainly there is a limit of 1024 threads per block; therefore multiple blocks need to be created. The optimal setting for the number of blocks to create and the number of threads to assign to each block to achieve optimal performance is not a precise endeavor, and requires experimentation. Factors that influence this are the number of registers used by a function, amount of shared memory used per block etc. For this function, our experiments show that optimal performance is reached (i.e. most cores are occupied for the majority of time) when the number of threads per block is 256 and the number of blocks are (No of threads required, 92160 in this case)/256. In this straightforward implementation, one entire SIPPA session is executed in a single thread. There are conceivably other implementations. For example, there are roughly six steps that SIPPA performs according to its algorithm, and one of these steps involving PPCSC is an aggregate of seven other steps for message exchange under a 2-party secure computation scenario. Each of these (12) steps could be implemented to run as a thread, thus the corresponding (12) threads in the same block realizes the implementation of one SIPPA session, and the parallel computing process for multiple SIPPA sessions is realized by scheduling multiples of these blocks to run in parallel. There are also other implementations that could be studied in future research to determine the optimal realization of the SIPPA implementation in a GPU environment.

The CUDA API takes care of all scheduling, and will only run threads as resources become available. When the Host calls Vector Multiplication, 92160 threads are scheduled on the GPU to execute the above function. Many of these threads execute in parallel, i.e. many of these SIPPA sessions (with each thread representing one particular SIPPA session) run in

parallel on the GPU. But from the programmer's perspective they all execute in parallel, unless any block specific synchronization is requested. In order to determine the place in memory to obtain the specific 10X1 vector a particular thread owns, the Unique Thread id (Utid in the above function) is needed. Utid is calculated utilizing the block id of the specific thread multiplied by the thread id of the particular thread. Using this Utid offsets one can calculate where to read the vector from, and where to store the resulting matrix (as shown in Fig. 20). Each thread block has the privilege of utilizing a 16 Kilobyte area of on-chip low latency memory, which a thread can access in around one to two clock cycles. In contrast, on-board device memory takes about 20-30 cycles to access. Given the fact that during Vector multiplication the same 10 numbers will be accessed repeatedly, creating a shared memory array of size 256*10 (when vector size is 10) will lead to a noticeable performance increase. It is because each of the threads in the block can copy over the 10X1 vector and perform multiplications utilizing the shared memory. Sure enough doing so leads to a performance increase of over 50% in vector multiplication. Since CUDA-C is essentially a subset of C, i.e. the GPU instruction set can essentially perform most functions a general purpose i386 processor can perform, we implemented SIPPA required functions like SVD, Calculating Eigenvectors, Pseudo Inverses using SVD etc. in CUDA-C by essentially translating widely available Open Source C LAPACK code. Translations were essentially trivial - mostly replacing standard C math functions with CUDA-C provided math functions. These functions were called in a similar manner to Vector Multiplication with each thread being assigned a particular matrix and that thread will call these iterative functions as necessary. Eigenvalues were calculated using the Power method since at this time SIPPA only uses the largest Eigenvalue and its corresponding Eigenvector. Inverses were Moore–Penrose pseudo-inverses calculated for precision utilizing the SVD method. Once required calculations are complete, and results are copied back to Host memory, on both the Client and Server Side. The client initiates a communication request with the server, and sends over $V_1^T A_1$ It sends the entire packet of 92160 10X10 matrices, in one stream of bytes. This is because sending individual matrices does not make sense when in GPU one has to provide all the matrices at the same time to utilize the power of parallel computing. Moreover, all results are also obtained at the same time when a GPU function is called. After a series of back and forth communication between the Server and Client as detailed before in the section discussing SIPPA algorithm, the server proceeds to send the entire block of X vectors (i.e. an array of size 92160*10 elements) to the client when the threshold condition is met. In other words, the server determines whether the threshold condition is met by comparing the predefined threshold to the distance between its Eigenvector and its estimate of the client's Eigenvector. The client also receives two other arrays from the client, a 92160 element array containing all server Eigenvalues and a 92160 element array containing the helper data $De^T \cdot x$ for all 92160 Vectors. Once the Client receives these three sets of data from the Server, it can proceed to reconstruct the server side image. One interesting issue encountered during the display of reconstructed images was that if the value of the reconstructed pixel was even slightly above the max intensity of 255 (in the case of ppm) the pixel would be displayed as black. This was remedied by re-writing all values above 255 as 255. This slight change has led to improvements in reconstruction.

5.3.3 MPSIPPA performance

We were surprised to note the immense increases in performances when using GPU's for SIPPA. The same task i.e. running SIPPA on a 640X480 RGB PPM image that took 1800

```
__global__ void VecMul(float* InputV, float* InputMat, unsigned int total){
    unsigned int Utid = blockDim.x * blockIdx.x + threadIdx.x;
    if (Utid < total){
    __shared__ float block[THDSIZE*VECTORSIZE];
    unsigned int count = 0;
    while(count<VECTORSIZE){
        block[(VECTORSIZE*threadIdx.x)+count] = InputV[(blockIdx.x*VECTORSIZE*blockDim.x)+((VECTORSIZE*threadIdx.x)+count)];
        count++;
    }//end Populate Shared Memory
    int col = 0;
    int ite = 0;
    for(int row=0; row<VECTORSIZE; row++)
    {
        col=row;
        ite = 0;
        while(col<VECTORSIZE){
            InputMat[((blockIdx.x)*VECTORSIZE*VECTORSIZE*blockDim.x)+(VECTORSIZE*VECTORSIZE*threadIdx.x)+(VECTORSIZE*row+col)]
            = block[(VECTORSIZE*threadIdx.x)+row] * block[(VECTORSIZE*threadIdx.x)+col];
            InputMat[((blockIdx.x)*VECTORSIZE*VECTORSIZE*blockDim.x)+(VECTORSIZE*VECTORSIZE*threadIdx.x)+(VECTORSIZE*row+col)+ (ite*(VECTORSIZE-1))]
            = block[(VECTORSIZE*threadIdx.x)+row] * block[(VECTORSIZE*threadIdx.x)+col];
            ite++;
            col++;
        }//end while
    }//end for
  }//end If
}//end VecMul
```

Fig. 20. CUDA Vector Multiplication code illustration.

minutes on iterative SIPPA, which later took 15 minutes on PSIPPA, now takes a mere 142 seconds on MPSIPPA. Two consumer level desktop computers each retailing for no more than $1000 equipped with NVIDIA GTX 480 GPU's were used. The following table details the performance of a few key functions used in SIPPA, each function was called 92160 times. The GPU used was a NVIDIA GTX 480, while the CPU used was a Dual Core AMD. The C-CPU functions shown below are almost exactly similar to the functions that are called in parallel on the GPU. The entire session was SIPPA among two 640X480 RGB PPM images with threshold set to ∞.

	Eigenvector Calculation (Seconds)	**SVD (Sec)**	**Pseudo-Inverse Of a Matrix (Seconds)**	**Overall Time for 92160-SIPPA-sessions**
MPSIPPA	0.045	0.06	0.08	0.616 sec of GPU time + 141 sec for data transfer, file read write , and read from disk to memory.
C-CPU LAPACK	0.5	8	14	N/A
JAVA-SIPPA	91	116	93	Over 1800 Minutes
PSIPPA	N/A	N/A	N/A	15 Minutes

Table 1. MPSIPPA Performance.

5.4 MPSIPPA and surveillance video footage

With the new performance increase provided with MPSIPPA, we are now able to expand SIPPA into realms that were previously only ideas. One of these expansions is regeneration of video footage captured by surveillance cameras.

The surveillance footage is captured using a network camera capable of 640x480 video capture. After the surveillance system captures a video, MPSIPPA can obtain this video and modify it for a SIPPA session. The SIPPA sessions however are not for a comparison between a sample video and template video, but are for other purposes. For example in a scenario regarding face biometrics, the sample video contains a specific number of frames

Image 1

Image 2

Reconstruction of server image 2 with client using image 1

Reconstruction of server image 1 with client using image 1

Reconstruction of server image 1 with client using image 3

Image 3

per second which vary from 5-60 with each frame representing one image. With each of these images a SIPPA session can be performed to reconstruct an estimate of an *image* located on the server side. After each image is reconstructed it may be sent to a matching engine to compare the template image with the reconstructed image. If this process happens 100 times, the matching engine may provide the system with 100 different scores each representing the similarity of the reconstructed image versus the template image. Doing this process multiple times may provide the system capabilities for better true acceptance rates and lower false rejection rates. The larger the sample of video results in a larger test bed of reconstructed images. As this test bed grows larger, the performance of the system may improve.

In testing MPSIPPA's performance with image reconstruction from surveillance footage we noticed a slight increase in performance in comparison to the performance for image reconstruction from static images. The performance time for a surveillance footage providing 50 images (5fps at 10seconds), resulted in approximately 5,000ms per image with a total run time of 246,380ms. Per image, this is approximately 2,000-3,000ms faster than static image comparison (Both Client & Server were on the same machine i.e localhost). With the initial experimental results, using MPSIPPA as a component in a real-time video surveillance face recognition and matching system seems promising. A short video of 5 seconds may be captured in 640x480 quality with 5fps and SIPPA would be capable of reconstructing estimates of the server side data within 5000ms per image. In approximately 120 seconds MPSIPPA would have generated 25 images for use in an efficient matching engine. These results do not consider time to write to file, however when using MPSIPPA as a component in a biometric system, it may not be necessary to write the file to disk, but merely keep it in memory and give the next component a stream of data directly from

memory. Eliminating the write to file seems a more practical and likely setup and with this setup MPSIPPA will provide close to 1400 images for verification in the same time it took MPSIPPA to write 25 images. If 25 images are all that are required to increase the performance of a biometric system, they may be reconstructed in approximately 2 seconds.

6. Conclusion

In summary, we present a secure computation technique that guarantees security and privacy for the reconstruction of voiceprints and images containing biometric information. Our secure computation technique, referred to as Secure Information Processing with Privacy Assurance (SIPPA), aims for sufficient privacy homomorphism and computational efficiency. Although it is not complete privacy homomorphism, SIPPA can be applied to the class of secure computation problems that is linearly decomposable. For proof of concept, we conducted both a simulation study and an application of SIPPA to reconstruct images with face biometrics as well as voice biometrics. Our experimental study showed that the results were consistent with the theory behind SIPPA. To achieve near real time performance, PSIPPA is studied and integrated in a slice-based architecture for realizing the full potential of SIPPA to achieve real time performance.

As a closing remark, we would also re-iterate many interesting research problems arose from this project; for example, what is the impact on the security and privacy when the original SIPPA protocol for 2-party secure computation is expanded to multi-party scenario in SFB-BioSIPPA? How do we handle failover in PSIPPA and SFB-BioSIPPA environment? What real time computation performance impact should be anticipated when MPSIPPA is scaled to operate in a wide-LAN environment?

Notwithstanding these astronomical increases in performance, NVIDIA CUDA provides mechanisms to further optimize performance. One might look into utilizing the ability of the newer CUDA enabled GPUs, which allow for asynchronous write to Host memory, or perhaps the ability to both load data into GPU memory while the GPU cores are simultaneously busy executing a function; or perhaps the ability of GPU's to directly display content on screen at over 300 Frames per second eliminating any file write (which seems to have taken over 8 seconds in our particular MSIPPA case). GPU programming surely holds promise in various areas including the field of biometrics, privacy protection including privacy preserving surveillance and computing: Real time multiple Face tracking/recognition, Object tracking, Biometric Intelligence involving identification of attributes such as gender, ethnicity, age etc. The possibilities are truly endless.

7. Acknowledgment

This work is supported in part by a grant from PSC-CUNY Research Award, PSC-CUNY CIRG17, and NSF DUE 0837535.

8. References

BioAPI 2.0 (May 2006), ISO/IEC 19784-1, Information Technology - BioAPI – Biometric Application Programming Interface - Part 1: BioAPI Specification (http://www.itl.nist.gov/div893/biometrics/standards.html)

Du, W.; Atallah M.J. (2001a). Secure Multi-Party Computation Problems and Their Applications: A Review and Open Problems. In *Proceeding of New Security Paradigms Workshop*, pp. 11-20.

Du, W.; Atallah, M.J. (2001b). Privacy-Preserving Cooperative Scientific Computations. In *Proceeding of 14th IEEE Computer Security Foundations Workshop*, pp. 273-282.

Gentry. C. (2009). Fully Homomorphic Encryption Using Ideal Lattices, In the *Proceeding of 41st ACM Symposium on Theory of Computing (STOC)*.

Goldreich, O.; Micali, S.; Wigderson A. (January 1987). How to play ANY mental game, *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, p.218-229, New York, New York, United States

Goldwasser, S. (1987). Multi party computations: past and present, *Proceedings of the sixteenth annual ACM symposium on Principles of distributed computing*, p.1-6, August 21-24, 1997, Santa Barbara, California, United States

Gross, R; Airoldi, E; Malin, B.; Sweeney, L. (2005) Integrating Utility into Face De-Identification.

Kirk, D.B.; Hwu, W.W; (2009). Programming Massively Parallel Processors: A Hands-on Approach, ISBN 0123814723, Morgan Kaufmann.

Nvidia (2010) http://www.nvidia.com/object/cuda_showcase_html.html as seen Oct 2010.

Newton, E.; Sweeney, L.; Mali, B. (February 2005). Preserving Privacy by De-identifying Facial Images. *IEEE Transactions on Knowledge and Data Engineering*, 17 (2), pp. 232-243.

Sanders, J.; Kandrot, E; (2010). CUDA by Example: An Introduction to General-Purpose GPU Programming, ISBN 0131387685, Addison-Wesley Professional.

Solove, D.J.; Rotenberg, M.; Schwartz P.M. (2006). Privacy, Information, and Technology, ISBN 0-7355-6245-8, Aspen Publishers, Inc.

Sy, B.K. (2009a). Slice-based Architecture for Biometrics: Prototype Illustration on Privacy Preserving Voice Verification. *Proceeding of Biometrics: Theory, applications and systems Conference*, Washington D.C.

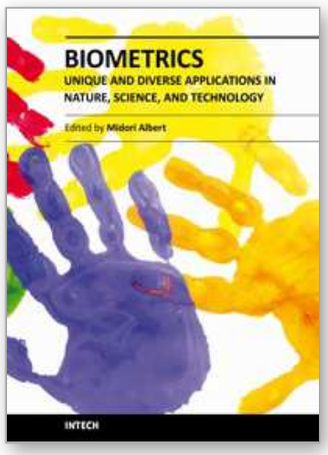
Sy, B.K. (2009b). Secure Computation for Biometric Data Security Application to Speaker Verification. *IEEE Systems Journal*, Vol. 3, Issue 4.

Sy, B.K. (2009c) http://www.qcwireless.net/biometric_ppr/he_primer.pdf

Sy, B.K.; Ramirez, A.; Arun P. Kumara Krishnan (2010). Secure Information Processing with Privacy Assurance Standard based Design and Development for Biometric Applications, *Proceeding of 8th International Conference on Privacy, Security and Trust*.

Turk and A. Pentland (1991). Face recognition using eigenfaces. *Proc. IEEE Conference on Computer Vision and Pattern Recognition*. pp. 586–591. Available online at: http://www.cs.ucsb.edu/~mturk/Papers/mturk-CVPR91.pdf.

Yao. A.C. (1982). Protocols for secure computations. In Proceeding of 23rd IEEE Sym. On Foundations of Computer Science.



Biometrics - Unique and Diverse Applications in Nature, Science, and Technology

Edited by Dr. Midori Albert

ISBN 978-953-307-187-9
Hard cover, 196 pages
Publisher InTech
Published online 04, April, 2011
Published in print edition April, 2011

Biometrics-Unique and Diverse Applications in Nature, Science, and Technology provides a unique sampling of the diverse ways in which biometrics is integrated into our lives and our technology. From time immemorial, we as humans have been intrigued by, perplexed by, and entertained by observing and analyzing ourselves and the natural world around us. Science and technology have evolved to a point where we can empirically record a measure of a biological or behavioral feature and use it for recognizing patterns, trends, and or discrete phenomena, such as individuals' and this is what biometrics is all about. Understanding some of the ways in which we use biometrics and for what specific purposes is what this book is all about.

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen