ON THE IMPLEMENTATION OF A SECURE MUSICAL DATABASE MATCHING

José Portêlo[1 2], *Bhiksha Raj*[3], *Alberto Abad*[1], *Isabel Trancoso*[1 2]

[1]INESC-ID Lisboa
[2]Instituto Superior Técnico, Lisboa, Portugal
[3]Language Technologies Institute, Carnegie Mellon University, Pittsburgh PA, USA
e-mail: {Jose.Portelo,Alberto.Abad,Isabel.Trancoso}@inesc-id.pt, bhiksha@cs.cmu.edu

ABSTRACT

This paper presents an implementation of a privacy-preserving music database matching algorithm, showing how privacy is achieved at the cost of computational complexity and execution time. The paper presents not only implementation details but also an analysis of the obtained results in terms of communication between the two parties, computational complexity, execution time and correctness of the matching algorithm. Although the paper focus on a music matching application, the principles can be easily adapted to perform other tasks, such as speaker verification and keyword spotting.

1. INTRODUCTION

Over the past few years there has been an exponential growth of security and privacy concerns motivated by an unprecedented level of access to multimedia contents and on-line services. This has led to the development of methods to protect the privacy of the persons using these resources and the data being exchanged.

In order to protect a person's individual privacy, blinding signature schemes have been proposed which hide that person's identity from the party with whom he/she needs to interact with. Regarding the privacy of the data being exchanged, there are two different situations: one where both parties require access to information and just need to communicate it in a safe way (e.g. e-banking, on-line shopping), and the other where both parties want to hide their personal information from the other but still be able to work together to achieve a specic goal (e.g. database query). The first can be achieved by encryption key agreement between the two parties and encrypting every thing they want to transmit. The second can be achieved by using Secure Multiparty Computation (SMC) protocols. The introduction of privacy does, however, come with a cost. Performing any of these operations in a secure way results in a considerable increase of both computational time and complexity.

In this paper we will focus on the implementation of a music matching task where both parties preserve the privacy of their data. This implementation differs greatly from the one where no privacy is preserved, because not only there is an increase in the complexity of the equations that represent each individual algorithm, but we also have to perform all the computations using ciphered values, which means restricting ourselves to operations based on modular arithmetic using an homomorphic cryptosystem. Our implementation will follow the algorithm described in [1]. We will consider a particular case of SMC where only two parties are involved in the secure computations. Although we present a specific application of SMC, the principles considered here can be easily adapted to perform other tasks, such as speaker identification/verification and keyword spotting.

In section 2, we describe the music matching problem and present the two main concepts necessary for its implementation in a privacy-preserving way. In section 3, we present a detailed implementation of a solution to the problem at hand. In section 4, we discuss a series of aspects related to the implementation itself. Finally we present some conclusions.

2. MUSIC MATCHING PROBLEM

A quick description of the problem is to consider a party (Alice) that has access to a music recording and wishes to find out relevant information about it (i.e. music title, artist, year of release, etc.) by querying an on-line database (Bob). An easy and efficient way of doing this can be achieved by Alice providing Bob with a small segment of her music, for instance a 5 seconds segment, and then he compares her music segment with the entire collection on his database. This can be done, for instance, by performing a cross-correlation between the signals. Consider, however, that both Alice and Bob have privacy concerns about the query. Alice may wish that Bob does not know which music she is querying and Bob may not wish his database to be exposed to direct access queries. This means that Alice and Bob should be able to perform the computations in such a way that they learn nothing about each other's data. Our implementation uses both SMC in the two party situation and an homomorphic cryptosystem to do just that. We now briefly present these two concepts.

2.1 Secure two party computations

Consider Alice and Bob have private data a and b respectively, and they wish for a trusted third party to perform an operation with their data such that $c = f(a,b)$, and then communicate the result back to them. The algorithm that computes $f(a,b)$ is *secure* only if it does not reveal more information about a and b than what could be gained by getting the result c directly from the trusted third-party. Such an algorithm should have the following properties:

- every step has to be expressed in terms of a few basic operations for which privacy preserving implementations already exist, and
- intermediate results have to be distributed randomly between the two parties, so that neither party is able to compute the entire result. This can be done by providing each party with *random additive shares* c_1 and c_2 such that $c_1 + c_2 = c$.

2.2 Homomorphic public-key cryptosystem

A public-key cryptosystem consists of a triple of probabilistic polynomial time algorithms for key generation (*KG*), encryption (*EN*) and decryption (*DE*).

- The *KG* algorithm generates a pair of private and public keys, *sk* and *pk* respectively.
- The *EN* algorithm creates a ciphertext (encryption) of a plaintext (input) *msg* using public key *pk*, $EN(msg, pk)$.
- The *DE* algorithm computes the original input *m* given the ciphertext of that input and the private key *sk*.

An homomorphic cryptosystem is a special type of cryptosystems that allows for specific algebraic operations to be performed indirectly in the plaintext by manipulating the ciphertext. We will consider an homomorphic cryptosystem with the following properties, which are crucial for performing the privacy preserving computations we are interested in:

$$EN(a, pk) \times EN(b, pk) = EN(a+b, pk) \tag{1}$$

$$(EN(a, pk))^b = EN(a \times b, pk) \tag{2}$$

The Paillier cryptosystem [2] is an example of such a cryptosystem, and we will use it in our implementation.

3. IMPLEMENTING A SECURE MUSIC MATCHING ALGORITHM

In this section we will describe how Alice and Bob can perform an on-line music query in such a way that they both preserve the privacy of their data. A theoretical description on how this should be done is presented in [1]. We will focus primarily on the implementation aspects. As in [1], we will also structure our implementation in steps, and for each step we will provide details on both the required operations and privacy-preserving algorithms we considered. The first step describes the computation of the cross-correlation, the second step shows how to obtain the maximum element of a given vector, the third step describes how to find the index of the maximum element of a vector and finally the fourth step shows how to obtain the information in a specific database entry.

3.1 Step 1: Cross-correlation of the music signals

In the first step we will perform the cross-correlation between Alice's music segment and each of the musics in Bob's database. Alice starts by generating a private and public key pair (*sk*,*pk*) and sends the public key to Bob. She then encrypts each element of her music segment (x_t) with *pk*, obtaining $e_t = EN(x_t, pk)$ for $t = \{1, ..., T\}$, T is the number of samples in her music segment, and sends them to Bob. He can now compute the cross-correlation in a privacy-preserving manner.

In order for Bob to compute the cross-correlation between Alice's values and his own values (y_t) there are two possible options: he can compute it either in the time domain or in the spectral domain. The first option is easier to implement, as the cross-correlation can be computed by:

$$z_n = \prod_{t=1}^{T} (e_t)^{y_{n+t}} \tag{3}$$

which is the ciphertext version of the expected equation. This solution is not usually used, because it is very time consuming. The execution time of the full cross-correlation in the time domain is $O(T^2)$. The second option consists of computing the Fast Fourier Transform (FFT) of both Alice's segment and each music in Bob's database, performing the cross power spectrum (CPS) on the resulting signals and finally computing the inverse FFT (IFFT). In terms of ciphertext this is represented by:

$$z = IFFT_{secure}\left((FFT_{secure}(e))^{FFT(y)}\right) \tag{4}$$

where FFT_{secure} denotes the privacy-preserving version of the FFT. This solution is less straightforward but faster than the first one. The execution time of the full cross-correlation in the frequency domain is $O(T \log_2(T))$.

In the case of desiring one sample precision in the computation of the cross-correlation between Alices music segment and the musics in Bobs database, we would definitely implement the second solution. However, there is no need for such precision when computing the cross-correlation between two music segments that are sufficiently long because consecutive cross-correlation results are not significantly different. Alternatively, we can choose a fixed larger step size for the computation of consecutive cross-correlations and, thus, we can save a lot of computational time. In particular, it would take the same amount of time to compute the cross-correlation either in the time domain or in the frequency domain when $T/\text{step} = \log_2(T)$. An analysis on the choice of value for the step is presented later on. There is also a problem with the complexity inherent to performing a FFT in a privacy-preserving way, for which an interesting analysis is presented in [3]. Computing the FFT in ciphertext not only requires much more operations than computing its equivalent in plaintext, but also each operation is more complex, which further contributes to avoid this solution. These two reasons led us to implement the privacy-preserving cross-correlation in the time domain.

After computing the cross-correlation, Bob generates random numbers b_i, one for each cross-correlation values, which will become his part of the random additive shares. Using these numbers, he then computes Alice's shares of the cross-correlation a_i such that $z_i = a_i + b_i$, and sends them to her.

3.2 Step 2: Obtaining the cross-correlation peaks

In the second step, Bob must compute the peaks of the cross-correlation for each music in his database. Since the result is randomly distributed between him and Alice, he cannot perform greater-than comparisons directly. He must take advantage of the fact that $z_i \geq z_j \iff (a_i - a_j) \geq (b_j - b_i)$. To perform the comparisons in a privacy-preserving manner, one can implement any solution to the Yao's millionaire problem [4]. In this paper we implemented an adaptation of the Blake-Kolesnikov algorithm [5]. This algorithm works by comparing two values at the bit level. Consider the bit representation of values $\alpha = a_i - a_j$ and $\beta = b_j - b_i$ to be $\alpha = \alpha_n \alpha_{n-1} ... \alpha_1$ and $\beta = \beta_n \beta_{n-1} ... \beta_1$ respectively. For each bit m, $m = 1, ..., n$, Bob computes:

$$e_{d_m} = e_{\alpha_m} \times (-e_{\beta_m}) \tag{5}$$

$$e_{f_m} = e_{\alpha_m} \times ((e_{\alpha_m})^{\beta_m})^{-2} \times e_{\beta_m} \tag{6}$$

$$e_{\gamma_m} = (e_{\gamma_{m-1}})^2 \times e_{f_m},\ e_{\gamma_0} = e_0 \tag{7}$$

$$e_{\delta_m} = e_{d_m} \times (e_{\gamma_m} \times e_{-1})^{r_m} \tag{8}$$

with r_m a random number. These equations represent $d_m = \alpha_m - \beta_m$, $f_m = \alpha_m \oplus \beta_m = \alpha_m - 2\alpha_m\beta_m + \beta_m$, $\gamma_m = 2\gamma_{m-1} + f_m$ and $\delta_m = d_m + r_m(\gamma_m - 1)$ in the plaintext, respectively. Bob then permutes all the δ_m and sends them to Alice. Alice now only has to check each δ_m and see if any of them is 1 or -1. If $\exists m : \delta_m = 1$, then $\alpha_m > \beta_m \Rightarrow \alpha > \beta$; if $\exists m : \delta_m = -1$, then $\alpha_m < \beta_m \Rightarrow \alpha < \beta$; else $\alpha = \beta$. The permutation we implemented was the Fisher-Yates shuffle, first presented in [6].

3.3 Step 3: Finding the most likely music index

In this step Alice's wishes to know which index in Bob's database corresponds to her music. This index is the one for which the vector of cross-correlation peaks has its maximum. Alice can obtain this information without Bob knowing which song she has by means of a *permute protocol* [7]. The main idea is that given Alice's and Bob's additive secret share vectors a and b they can compute different secret share vectors $\overline{a}$ and $\overline{b}$ such that $\overline{a} + \overline{b} = \pi(a) + \pi(b)$, where π is a random permutation.

The permute protocol works as follows. After Bob generates a private and public key pair (sk,pk), he encrypts each of his random additive shares corresponding to the cross-correlation peaks and sends them to Alice. She generates a random number s_k for each value from Bob and computes $b'_k = EN(b_k, pk) \times EN(s_k, pk)$ and $a'_k = EN(a_k, pk) \times EN(-s_k, pk)$. She then generates another random number r and computes $a''_k = a'_k \times EN(r, pk)$. She generates a random permutation π and computes $\overline{a} = \pi(a'')$ and $\overline{b} = \pi(b')$. Finally she sends $\overline{a}$ and $\overline{b}$ to Bob. Notice that Bob now has access to $\tilde{z} = \pi(a) + \pi(b) + r$ in the plaintext for each music, but he cannot know any of the values of the cross-correlations because he does not know r and he does not know which index corresponds to which music in his database because of the permutation π. He can compute the maximum of these values and send the permuted index back to Alice. After she undoes the permutation, she now knows the index of her music in the database.

3.4 Step 4: Obtaining the desired information

In the final step Alice finally retrieves from Bob's database the information about her music. If the database was public, she could just browse it until she found her music index. But since Bob wants to preserve the privacy of his database, except of course the information on Alice's music, they must exchange the information using *oblivious transfer* [8]. Consider l to be the index of Alice's music on the database and $\mathbf{u} = \{u_1, ..., u_K\}$ a vector representing the relevant information of the K musics in the database. If Alice encrypts her music index and send it to Bob, he can just generate random numbers r_k and compute:

$$v_k = EN(u_k, pk) \times (EN(l, pk) \times EN(-k, pk))^{r_k} \qquad (9)$$

with $k = 1, ..., K$, which in the plaintext is equivalent to $v_k = u_k + r_k(l - k)$. He then sends them to Alice. Notice that the privacy of the database is preserved, because even if Alice decrypts every message she receives from Bob, all but the information corresponding to her music will be random numbers to her.

4. EXPERIMENTAL RESULTS

In this section we describe the experiments we performed in order to obtain an implementation of a music matching algorithm which is fast enough to be used as an on-line application without compromising the correctness of the music matching process. We will focus on the following aspects: communication between the two parties, complexity and time consumption of the algorithms and correctness of the matching result.

The results presented in this section were performed using encryption keys with $N_{bits} = 512$ bits in length and a music database of 50 16-bit WAVE files of different genres, including celtic, pop and metal, and with an average duration of 3 minutes and 30 seconds. In order to reduce the size of the musics in terms of number of samples, since it greatly affects the execution time of the algorithm, we downsampled all the musics to $F_s = 8000$Hz.

4.1 Communication between the two parties

Regarding the communication exchanges between Alice and Bob, there is a major difference between the situation where the music matching occurs without the use of privacy-preserving techniques and the situation where cryptography is used. In the first situation, communication between the parties occurs only at the start of the interaction, when Alice sends her music segment to Bob, and at the end of that interaction, when Bob returns the information corresponding to her music. As it can be observed, the total amount of communication in this situation is very small. In the second case, mainly due to the use of cryptography and random additive shares at each computation step, the need for communication is much larger. We will now analyse this situation in more detail.

In the second situation there are three classes of values being exchanged between Alice and Bob: control values used for exchanged messages synchronisation, such as the size of Alice's music segment, etc., cryptographic keys and the encrypted values themselves. Since the third class overwhelms the other two in terms of total amount of communication required, we will examine only that one.

Like in the first case, performing the music matching in a private way starts by Alice sending her music segment to Bob. The difference is that in this case every sample has to be encrypted. If we consider S to be the size of Alice's segment, she has to send $2N_{bits}S$ bits[1] instead of the original $16S$ bits.

For each cross-correlation Bob computes, he has to send a random additive share to Alice. The average length of the musics in Bob's database is much larger than the length of Alice's music segment, so he has to compute on average N_{step} cross-correlations for each music, where the value of N_{step} will depend on the chosen step between cross-correlations. This means that at this stage he has to send a total of $2N_{bits}KN_{step}$ bits to Alice. For obtaining the cross-correlation peaks, Alice has to send to Bob the encryption of each bit of each of her random additive shares for the cross-correlation and Bob has to send back the greater-than result for each comparison he performs. This means that the total amount of bits exchanged in this step is $2(2N^2_{bits}KN_{step})$. For finding the

[1] we have to consider $2N_{bits}$ bits per message instead of just N_{bits} because the Paillier cryptosystem performs the following mapping: $p \in Z^*_{N_{bits}} \rightarrow c \in Z^*_{N^2_{bits}}$, $p-$plaintext message, $c-$ciphertext message

most likely song index, there are three rounds of encrypted random additive shares exchange between Alice and Bob. In this step they exchange between themselves $3(2N_{bits}K)$ bits. Finally in the last step, Bob sends to Alice the information on his database. Assuming that the information of each song can be represented in a single number of N_{bits} bits, he sends $2N_{bits}K$ bits to Alice. The summary of the communication required between the two parties is presented in Table 1.

situation	no privacy	with privacy
Step 1	$16S$	$2N_{bits}S + 2N_{bits}KN_{step}$
Step 2	-	$2(2N^2_{bits}KN_{step})$
Step 3	-	$3(2N_{bits}K)$
Step 4	N_{bits}	$2N_{bits}K$

Table 1: Summary of communication analysis, results presented in number of bits.

4.2 Computational complexity

Another aspect where there is a significant difference between the two implementation approaches is the computational complexity, represented by the type and number of operations required for each algorithm. We will use the abbreviations *CA* for complex addition, *CM* for complex multiplication, *MM* for modular multiplication and *ME* for modular exponentiation. Notice that *MM* is the ciphertext equivalent of *CA* in the plaintext and that *ME* is the ciphertext equivalent of *CM* in the plaintext. Also, *CA* is composed by 2 additions and *CM* is composed by 2 additions and 4 multiplications. The computational complexity of the algorithms is summarised in Table 2.

situation	no privacy	with privacy
cross-correlation	$T(1CM + 1CA)$	$T(4ME + 2MM)$
greater-than	1	$N(6MM + 4ME)$

Table 2: Summary of computational complexity analysis, results presented in number of operations.

4.3 Execution time

We now analyse the implementation of the algorithms used for the privacy-preserving music matching in terms of execution time. All the execution times presented were obtained running the algorithms on a Intel Core2 Quad CPU Q6600 @ 2.40GHz. We will focus on two main algorithms, which are the privacy preserving versions of the cross-correlation and greater-than algorithms.

We considered three different values, $T = 8000$, $T = 40000$ and $T = 80000$, because we wanted to evaluate the obtained results for music samples of 1, 5 and 10 seconds, respectively. The execution time, in seconds, for each algorithm and for the values of T considered is presented in Table 3.

For comparison, we also present the execution times, in seconds, for the non-private implementations of the same algorithms in Table 4.

The results in Table 3 immediately raise some concerns. If we consider the music matching of Alice's music segment with a single music in Bob's database, we need to compute N_{step} privacy-preserving cross-correlations and N_{step} privacy-preserving greater-than comparisons. If we consider

T	cross-correlation	greater-than
8000	4-5	1-2
40000	18-20	1-2
80000	35-40	1-2

Table 3: Summary of execution times for privacy-preserving algorithms, results presented in seconds.

T	cross-correlation	greater-than
8000	$\ll 0.01$	~ 0
40000	< 0.01	~ 0
80000	0.01	~ 0

Table 4: Summary of execution times for non-private algorithms, results presented in seconds.

that the average music has a 3 minutes and 30 seconds duration and a step between cross-correlations of 1 second, it would take around 1 hour and 15 minutes in the situation where $T = 40000$ for a single computer to perform the necessary computation on a single music. Considering that a database may contain thousands of musics, the whole process would required an unacceptably large amount of time. However, we can take advantage of cluster computing [9] in order to solve this problem. Many of the companies that might be interested in implementing privacy-preserving music matching algorithms are likely to have tens of thousands if not hundreds of thousands of servers at their disposal, which would allow them to fully parallelize all the computation of the privacy-preserving algorithms. Given a sufficient amount of servers, each could compute a single cross-correlation and a single greater-than comparison, and then merge all the results together. Although we did not test any form of parallel computing, we reckon that from the user's perspective, the time it takes for him/her to get the information he/she needs should reduce from several days to only a few minutes.

4.4 Correctness of the results of the music database matching algorithm

The correctness of the matching results is also a very important aspect to take into account, as the whole music matching process is useless if it does not provide a correct result. In order to find out an adequate value for the step between the cross-correlations we chose several random music segments of 1, 5 and 10 seconds from the musics in the database and used them to perform the query using different values for the step between cross-correlations in order to find out when a visible degradation of the results occurred. This was evaluated by analysing the ratio between the maximum of the cross-correlation for the correct music and the maximum of the cross-correlation for all the remaining musics. A ratio lower than 1 means a wrong identification. This ratio was computed over a wide range of values for the step, and the results we obtained are presented in Figure 1.

By analysing the results, we can see that adequate values for the step between cross-correlations are 80, 400 and 2000 samples for $T = 8000$, $T = 40000$ and $T = 80000$, respectively. We can now compute the average number of cross-correlations we have to compute for each music in the database. These results are presented in Table 5. As expected, for larger music segments, one can compute much less cross-correlations.

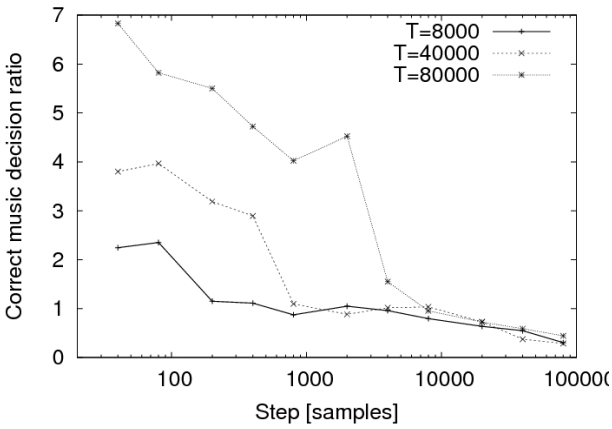

Figure 1: Music matching results as a function of step.

T	step [samples]	N_{step}
8000	80	21000
40000	400	4100
80000	2000	800

Table 5: Number of cross-correlations as a function of step.

5. CONCLUSIONS

In this paper we presented an implementation of a privacy-preserving music database matching algorithm. We tackled not only the implementation details but also presented a thorough analysis of the obtained results in terms of communication between the two parties, computational complexity, execution time and correctness of the matching algorithm. Although a solution for this problem in a privacy preserving manner brings a considerable increase in the amount of resources necessary to perform it in a satisfactory way, we also notice that this is the cost of privacy. Hiding information from another party and still allow it to perform useful computations over it comes as a trade-off for computational complexity and execution time.

6. FUTURE WORK

The earlier version of this work was focused only in the detailed implementation of the algorithm presented in [1]. However, we found out later that it is possible for Bob to attack the algorithm in Step 3 and find out which song Alice has. A simplified presentation of the attack is as follows. Since Bob has access to all $\tilde{z}$ in the plaintext for each music, he can compute $\tilde{z}_{ij}^{diff} = \tilde{z}_i - \tilde{z}_j$, $i, j = 1, ..., K$, thus removing the random number $EN(r, pk)$. From there he can compute all the cross-correlations between all the music segments in his database, and match them to each $\tilde{z}$, When he finds all the matches, he can reverse the permutation. Because the attack uses exclusively operations in the plaintext, Bob can perform it in a very time and resource efficient way.

We now present a sketch of a possible solution to counter this attack. In order to effectively prevent Bob from knowing which song Alice has, he must compute the most likely song index in the ciphertext, which means using once again secure greater-than comparisons. An initial approach could be for Alice and Bob to perform $K-1$ greater-than comparisons, which is the minimum required amount for an array of size K. The problem with this approach is that Bob knows that the index of the maximum will be one of the two values present in the last comparison, and therefore he will have a 50/50 percent chance of guessing correctly which song Alice has. For Bob to know nothing about the index of the maximum, he would have to compute all the possible greater-than comparisons, which sum up to $\frac{K(K-1)}{2}$ comparisons. The problem with this approach is that it takes too much time to perform. We found out that a solution using $\frac{K}{2}\log(k)$ comparisons not only takes an acceptable time to perform but also provides enough security to prevent Bob from getting any information he is not supposed to.

7. ACKNOWLEDGEMENTS

This research was partially supported by FCT grant SFRH/BD/71349/2010.

REFERENCES

[1] M. Shashanka and P. Smaragdis, "Privacy-Preserving Musical Database Matching", in *2007 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, New Platz, NY, October 21-24, 2007, pp. 319-322.

[2] P. Paillier, "Public-key Cryptosystems based on Composite Degree Residuosity Classes", in *Proceedings of Advances in Cryptology - EUROCRYPT'99*, ser. Lectures Notes in Computer Science, J. Stern, Ed., vol. 1592, 1999, pp. 104-120.

[3] T. Bianchi, A. Piva and M. Barni, "Implementing the Discrete Fourier Transform in the Encrypted Domain", in *Proceedings of the 33rd International Conference on Acoustics, Speech, and Signal Processing*, Las Vegas, Nevada, March 30-April 4, 2008, pp. 1757 - 1760.

[4] A. C.-C. Yao, "Protocols for Secure Computation", in *Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science*, 1982, pp. 160-164.

[5] I. Blake and V. Kolesnikov, "Strong Conditional Oblivious Transfer and Computing on Intervals", in *Proceedings of Advantages in Cryptology - ASIACRYPT'04*, volume 3329 on LNCS, pp. 515-529, Springer-Verlag, 2004.

[6] R. Fisher and F. Yates, "Statistical Tables for Biological, Agricultural and Medical Research", London: Oliver & Boyd, 1938.

[7] M. Atallah, F. Kerschbaum and W. Du, "Secure and Private Sequence Comparisons ", in *Proceedings of Workshop on Privacy in the Electronic Society*, Washington, DC, October 2003.

[8] M. Naor and B. Pinkas, "Oblivious Transfer and Polynomial Evaluation", in *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, 1999, pp. 245-254.

[9] M. Baker and R. Buyya, "Cluster Computing at a Glance, High Performance Cluster Computing: Architectures and Systems ", Vol. 2, Rajkumar Buyya (ed), pp. 3-47 (Chapter 1), ISBN 0-13-013784-7, Prentice Hall, NJ, USA, 1999.