A Multi-User Key and Data Exchange Protocol to Manage a Secure Database

João Nunes Souza Márcio Aurélio Ribeiro Moreira Ilmério Reis Silva

Faculdade de Computação
Universidade Federal de Uberlândia
Uberlândia – Minas Gerais – Brasil
{nunes, ilmerio}@ufu.br marcio@acc.com.br

Abstract

We propose a multi-user key and data exchange protocol, which is able to manipulate stored encrypted data without need to decipher. This protocol can be used by several clients at any given time, which are connected to a server, for keys and data exchanges. The ciphered data located in client and that located in the server is exchanged, stored, compared and processed without the need to decipher. To clarify the protocol explanation we use it in a Database Management System (DBMS). The relational algebra operations, which include projection, selection, among others, are evaluated using the cryptographic protocol on encrypted data. Let us suppose a relation instance I_s ciphered and stored on secondary memory. The protocol can evaluate queries such as selection $\sigma_{Ai > v}(I_s)$ on the encrypted stored version. In this selection no data is deciphered onto main or secondary memory and therefore remains encrypted. Different users can submit queries, thus the protocol must have a strong key management.

Keywords: Cryptography, secure DBMS, multi-user secure protocol, queries over encrypted data.

1. Introduction

Nowadays, the Internet interconnects many computers and this raises a key security issue of intruders accessing data. Many communication protocols have been proposed and implemented to protect information from attacks by intruders (such as safe bank systems through the Internet) [15]. However, accessing the main or secondary memory of the server can break the security system. So, a protocol to control the access to data must improve the security of processing and storing data. Besides standard security controls (authorizations, privileges and grants), several database management systems use cryptographic protocols to improve the security of stored data. In general, to process a query, these systems decrypt data blocks. Although there are systems that do not require decryption to process queries based on equal comparisons [1], to process more complex queries the database management system must decipher data blocks [12].

There are two vulnerable aspects when dealing with the system mentioned previously. First, any user who has permission to read part of a data block can also access the remainder of the block, through monitoring the main memory during query processing. Second, as above an intruder can read the deciphered data block in the main memory. Notice that, these problems can't to solve by tools like firewall or traditional cryptographic techniques applied to databases. Would be necessary a multi-user key and data exchange which is able to manage data safely among many users and a server to solve these problems. In this context the data has to be stored, processed and exchanged in ciphered form, in other words, the data can be

visible to anyone but only understandable to authorized users [4]. To avoid ciphered data vulnerability it has to be manipulated in ciphered form even in main memory. The ciphered data block, stored in secondary memory, has to be processed in main memory without deciphering.

In this paper, we propose a cryptographic protocol to evaluate relational algebra, which is not vulnerable as in the above method. The cryptographic protocol is able to evaluate the relational algebra operations, which include projection, selection, among others, using encrypted data. Let us take a relation instance $\boldsymbol{I_s}$ ciphered and stored on secondary memory. The protocol can evaluate queries like selection $\boldsymbol{\sigma_{Ai > v}(I_s)}$ on the $\boldsymbol{I_s}$ encrypted version which is stored without deciphering any of the encrypted data or metadata in main or secondary memory. The database server is secure enough to answer queries without deciphering any data. The security of the data system is equivalent to the cryptographic system it utilizes. In this case, the cryptographic system is based on elliptic curves. The main contribution of this paper is a multi-user key and data exchange protocol, which manipulates stored encrypted data without deciphering. This protocol answers some of the limitations set out in [1].

The proposed protocol has many applications. Let's suppose that a database containing very secret information where users can access it remotely, for example, a Kerberos Key Server [13] or the database containing secret keys of other applications such as in a key distribution center. This database can be encrypted and stored in an Internet server as well. The users could be authorized to access the server remotely. However, in this case, the server must be able to process the users query without decryption of any data. So, the server sends to the users the query result in a ciphered form. Users can get the plain information safely, by disconnecting his machine from the net and deciphering the query result.

Comparing the DBMS using the proposed protocol and usual DBMS, the former has a data storage overload and processing overheads, caused by additional data storage and by the encrypting and decryption process. So, the proposed DBMS is recommended for high security databases, which are not very large. It's recommended for databases containing very secret information.

The paper is organized as follows. After this introduction, we discuss related work. Then, the basic concepts are presented relating them to elliptic curves and their application in cryptographic systems. Then, our protocol description is presented. We discuss how to encrypt and store the data, and how a query can be processed without decrypting data. The considered query set is a complete set of relational algebra operations, which include project, union, difference, selection, and cartesian product. Finally, we present our conclusions followed by an example of didactical application.

2. Related Work

A multi-user key exchange protocol such as Kerberos [13] has as one of its main functions the protection of data during its transmission, as the means of transmission are considered vulnerable. On the other hand the protocol considers the user and the server main memories as protected against intruders and therefore does not act upon them in anyway. In the proposed protocol we consider as insecure user and server main memories. In our protocol the server can receive requests and process them without decrypting any data in main memory.

Traditionally, access control in database management systems is based on authorizations, privileges and grants [4, 5]. Several relational database management systems use cryptographic techniques to improve data security. The motivation for these protection systems is the presence of intruders that can access the database files and obtain information

to which they do not have permission of access. But, as far as we know, when a query is evaluated by the system, it decrypts data in the main memory and thus, expert intruders could therefore read the main memory and access unauthorized data. An alternative, but limited approach, is processing queries by using hash functions [6, 7]. Unfortunately, hash functions are limited and can be used only in single selections with equal (or non equal) operation. For more elaborated queries we need more complex cryptographic protocols. Here, we extend these cryptographic protocols presented in [6, 7] to allow an evaluation of more complex queries without decrypting data.

We propose a DBMS using encrypted data. The system stores and processes ciphered data without decrypt it in the main or secondary memory. To the best of our knowledge, the common systems store ciphered data in secondary memory and process it in plain text form in main memory. In some commercial DBMS, it's possible to check this by monitoring main memory during a query process. We are using cryptographic protocols based on public keys to solve this problem. Public keys cryptographic systems are very popular to encrypt digital signals. The most common public key based system is the RSA [9, 11, 13, 14]. However, over the last few years, cryptographic systems based on elliptic curves have become more popular due to the higher levels of security and performance offered by these systems. An elliptic curve cryptographic system (ECC system), which uses a key with 155 bits, has the same security level as a RSA system, which uses a key with 512 bits [2, 8, 13]. Furthermore, the ECC system has a better performance than a RSA system because the multiplication operations in RSA systems are replaced by addition operations in ECC systems [10].

Next, we present a brief background on elliptic curves cryptographic systems. Immediately after, we present our protocol to process relational algebra operations using encrypted data.

3. Elliptic Curves and Cryptography

3.1. Elliptic Curves

The ECC systems are based on the elliptic curve theory. An elliptic curve $\mathcal{C}$ has cubic equations

$$y^2 + a\,x\,y + b\,y = x^3 + c\,x^2 + d\,x + e$$

where a, b, c, d and e are real numbers. Further, an elliptic curve has a point θ in the infinite [2, 8, 13]. The addition of two points P and Q in an elliptic curve $\mathcal{C}$ is a closed operation (i.e., if $P, Q \in \mathcal{C}$ then $P + Q \in \mathcal{C}$) with the following properties:

1. given points P, Q, and $R \in \mathcal{C}$, such that they are on a straight line, then $P + Q + R = \theta$;
2. the point θ in the infinite is the addition neutral element, then $-\theta = \theta$ and if $P \in \mathcal{C}$ then $P + \theta = P$;
3. given points $P_1 = (x, y)$ and $P_2 = (x, -y)$ on a vertical line, such that $P_1, P_2 \in \mathcal{C}$, then $P_1 + P_2 = \theta$, and $P_1 = -P_2$;
4. given points $P, Q \in \mathcal{C}$, the point $P + Q$ can be obtained as follows. Let R be the third intersection point between the straight line passing by P and Q and the elliptic curve $\mathcal{C}$. Then $P + Q + R = \theta$ and $-R = P + Q$;
5. given point $P \in \mathcal{C}$ the point $P + P$ (i.e., $2\,P$), can be obtained by tracing the elliptic curve tangent which intercepts $\mathcal{C}$ in P. If this tangent also touches the elliptic curve at another point we call R. We therefore have $2\,P + R = \theta$, and thus, $2\,P = -R$.

The add operation of points in an elliptic curve is commutative and associative. Then we can compute $m\,P$, $m \geq 3$, adding two points at a time.

3.2. Cryptographic Systems based on Elliptic Curves

Elliptic Curve Cryptographic Systems are based on the structure of finite algebraic groups defined by the elliptic curves, as follows.

Let a and b be integers and p a prime number, such as $0 \leq a, b < p$ and $(4\,a^3 + 27\,b^2) \pmod p \neq 0$. The elliptic group module p satisfying an elliptic curve $y^2 = x^3+ax+b \pmod p$, is the set of points defined by $E_p(a, b) = \{(x, y) \mid 0 \leq x < p, y^2 = x^3 + a\,x + b \pmod p\} \cup \{\theta\}$, and the previously defined add operation for points in an elliptic curve.

Given two points $P_1=(x_1, y_1)$ and $P_2=(x_2, y_2)$ of the elliptic group, then $P_1+P_2=(x_3, y_3)$ such that:

$$x_3 = \lambda^2 - x_1 - x_2 \pmod p$$
$$y_3 = \lambda\,(x_1 - x_3) - y_1 \pmod p$$

where
$$\lambda = (y_2 - y_1) / (x_2 - x_1) \pmod p \text{ if } P_1 \neq P_2$$
$$\lambda = (3\,x_1^2 + a) / (2\,y_1) \pmod p \text{ if } P_1 = P_2$$

There are several ways to define a cryptographic system based on elliptic curves. Here, we do not explore these alternatives, but we only use one of them to define our protocol (to study other alternatives we refer the reader to [2, 8, 9]).

In general, the elliptic curve cryptographic systems use points in an elliptic group $E_p(a,b)$ to represent symbols to be encrypted. The security of these systems is based on the following assertion: *Given two points P and Q of an elliptic group, such that $Q = k\,P$, where $k < p$, it is easy to compute Q from P and k. However, it is very hard to compute k from P and Q*. This problem corresponds to the well-known *discrete logarithm problem*. Here, easy and hard concepts are those defined by cryptology. The ECC system parameters can be defined as follows.

1. choose a prime number p such that $p \cong 2^{180}$;
2. choose the integer numbers a, b in the equation $y^2 = x^3 + a\,x + b \pmod p$, where $0 \leq a, b < p$ and $(4\,a^3 + 27\,b^2) \pmod p \neq 0$;
3. define the elliptic group $E_p(a, b) = \{(x, y) \mid 0 \leq x < p, y^2 = x^3 + a\,x + b \pmod p\} \cup \{\theta\}$;
4. choose a generator point $G \in E_p(a,b)$ such that G satisfies the following assertion: *if n is the least number such that $n\,G = \theta$, then n is a large prime number and n is the order of G*.

Given the elliptic group $E_p(a,b)$ and the point G, both public, we can cipher data M as follows.

1. choose a private key P_v such that P_v is an integer less than n;
2. define a public key given by the equation $P_b = P_v\,G$ (notice that G and P_b are public, however it is hard to compute P_v).
3. define the pair $(X, Y) = (k\,G, M + k\,P_b)$ as the encrypted form of M, where k is an arbitrary positive integer.

The k value is also secret. To decipher M we compute $Y - P_v\,X$, as follows,

$$\begin{aligned} Y - P_v\,X &= M + k\,P_b - P_v\,k\,G \\ &= M + k\,P_v\,G - P_v\,k\,G \\ &= M \end{aligned}$$

4. Relational Algebra using Encrypted Data

In this section we present a protocol using ECC system to encrypt and store data and to evaluate relational algebra operations without data deciphering. We describe the key exchange by the server and a new user A, how the data should be stored in the server, how the user A and the server can make the data exchange and query evaluation.

4.1. Introduction

In our proposed system we are working with secret and public data. The secret data (server's secret key, secret keys of server-users communication, etc.) one therefore needs to store in ciphered form to avoid attacks by intruders. Besides this, the manipulation of select data, in main memory, has to be a very careful operation. The secret key used to cipher the secret data has to be stored outside the system, so not to leave the whole system vulnerable. So, this secret key is the master key and has to be supplied offline by the DBA (Database Administrator), in this sense, it's denoted by the DBA Key. The secret and public data has to be stored in different table spaces. The secret data, ciphered using the DBA Key, is stored in a Secret Table Space and the public data is stored in a Public Table Space, as showed in the Figure 1.

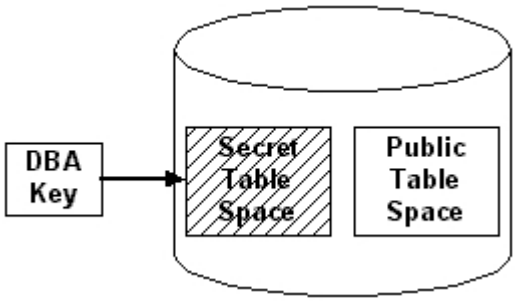

Figure 1: Table Spaces.

To access the secret data the DBA needs to disable the server communications, interrupt all other user's level processes and supply the DBA Key. When the use of secret data is finished, the access to Secret Table Space should be closed and the memory area, where the secret data and the DBA Key were stored, should be cleaned and released. So, the other user processes can be released and the server communications can be reactivated.

First, we tried to use the traditional ECC method i.e., store the message $\boldsymbol{M}$ in the form of a pair $\boldsymbol{(k\,G,\ M + k\,P_b)}$. However, this ECC method is mono-user. When we tried to use this technique in client-server architecture, it was shown highly vulnerable to intruder attack. This means, a legal user can access all data in the system, even without the rights to do so. Besides this, in the Man-In-The-Middle (MIM) attack an intruder could easily break into the whole system. Then we tried several other methods, even so, when we tried to protect the data from the intruders, in the MIM attack, the user also couldn't decipher the data. On the other hand, when the user managed to decipher, the intruder also managed to break into the system. To solve this problem it's necessary to differentiate the intruder from the users, and the users from each other. We therefore found it appropriated use the public key exchange concept with a shared key. Now, each user has a shared key with the server, besides their secret and public keys.

To solve the problem of safety during communication and query evaluations, we propose using data in ciphered form the whole time i.e., the data has to trafficked, be stored and be processed in the ciphered form. The system should be capable of evaluating queries without deciphering the data. To differentiate one user from another, and these from the

intruder, we propose the transformer cipher view. In this mechanism a ciphered message in a certain user's view is transformed into the ciphered message in the server view and vice-versa, without deciphering the original message. Blaze and Strauss [3] investigates the possibility of atomic proxy functions that convert cipher text for one key into cipher text for another without revealing secret decryption keys or clear text messages. However, their solution does not consider the security of processing in the main memory of the server.

4.2. Representation in the Elliptic Group

In our system we can represent the natural numbers $0, 1, \ldots r$ as points of the Elliptic Group. We can take a generator point G as a representation of the number 0 denoted by $E(0) = G$. Consider $E(1) = G + G = 2G$. In general we have $E(r) = (r + 1) G$. Let r is the maximum natural number represented in the system. The representation function $E(x)$ has the following properties: $E(x) + E(y) - G = E(x + y)$ and $E(x) - E(y) + G = E(x - y)$. The last property is demonstrated:

$$\begin{aligned} E(x) - E(y) + G &= (x + 1) G - (y + 1) G + G \\ &= (x - y + 1) G \\ &= E(x - y) \end{aligned}$$

Taking r less than $n/2$, where n is the elliptic group order, we have: if $(x - y) < 0$ then $E(x - y)$ does not belong to the set of representation points $\{E(0), E(1), \ldots E(r)\}$. We will use this property in the selection operation.

4.3. Key Exchange

In this section, we describe the key exchange protocol used by the server and user A, which is analog to the Diffie-Hellman Key Exchange protocol [13]. First we make a summary of the notations for the chosen and calculated variables. In Table 1, the first and second columns contain chosen and calculated values by user A and by the server, respectively. Some of these values are secret and others are public, as described in the table. The lower case letters are used for numbers and the capital letters for points on the ECC.

Chosen by user A:		Chosen by server:	
k_A	User A secret key	k_S	Server secret key
k_1	User A secret session key	k_2	Server secret session key
		n_A	Secret key to communicate with user A
		n_{Sti}	Secret key used by server to store data
		G	Generator point (shared with all users)
Calculated by user A:		Calculated by server:	
P_A	User A public key	P_S	Server public key
P_1	User A public session key	P_2	Server public session key
S_A	Shared secret key	S_A	Shared secret key (shared by user A and server)
		W_A	Point used by user A to cipher data
		X_A	Point used by user A to decipher data
		T_{ASti}	Points that transform data from user A to server
		T_{SAti}	Points that transform data from server to user A

Table 1: Summary of the chosen and calculated variables.

In Table 2, the columns show the user A private knowledge, the server private knowledge and the public knowledge. As we described in subsection 3.2, n is the order of G.

User A knowledge	Public knowledge	Server knowledge
k_A, k_1, S_A, W_A and X_A	n, G, P_A, P_S, P_1, P_2, T_{ASti} and T_{SAti}	k_S, k_2, S_A, n_A and n_{Sti}

Table 2: Summary of what each entity knows.

Phase	User A	Server
Database creation		Select k_S, G and n_{Sti} Calculate and distribute $P_S = k_S\ G$ Encrypt DB computing $Y_{ti} = E(v_{ti}) + n_{Sti}\ P_S$
User A registration	Select k_A Calculate and distribute $P_A = k_A\ G$ Calculate $S_A = k_A\ P_S$	Calculate $S_A = k_S\ P_A$ Select n_A Calculate and give to the user A $W_A = n_A\ P_A$ and $X_A = n_A\ G$ Calculate $T_{ASti} = n_{Sti}\ P_S - n_A\ P_A + S_A$ and $T_{SAti} = n_A\ P_A - n_{Sti}\ P_S - S_A$
User A log in	Select k_1 Calculate and distribute $P_1 = k_1\ G$	Select k_2 Calculate and distribute $P_2 = k_2\ G$

Table 3: Key Exchange Summary.

As showed in Table 3, to create a database, before any user insertion, the server chooses a random positive integer k_S, less than n. This is the server's secret key. The server selects a point $G \in E_p(a,b)$, that will be the generator point of ECC system. Then the server generates his public key $P_S = k_S\ G$.

When a new user A has to be registered in the system, it selects an arbitrary positive integer k_A, less than n. This is user A's secret key. The user A generates $P_A = k_A\ G$, your public key. The user A computes the shared secret key $S_A = k_A\ P_S$, that is shared by it and server. The server generates the shared secret key $S_A = k_S\ P_A$, which have the same value of the shared secret key generates by the user A. As we can see: $S_A = k_S\ P_A = k_S\ k_A\ G = k_A\ P_S$.

The server chooses a random positive integer n_A, less than n. This is a secret number and will be used by the server to communicate with user A. The server computes and gives to user A the following values: $W_A = n_A\ P_A$ and $X_A = n_A\ G$.

For each relation, tuple and attribute, the server chooses another random positive integer n_{Sti}, less than n. This secret number will be used to store data.

The server has to compute the transformers points for the user A to each relation, tuple and attribute given by $T_{ASti} = n_{Sti}\ P_S - n_A\ P_A + S_A$ and $T_{SAti} = n_A\ P_A - n_{Sti}\ P_S - S_A$. The server uses these points to transform data received from user A and sent to it respectively.

When the user A has to log in the server, it selects a random positive integer k_1, less than n. This is the user A secret session key. Then, the user A computes and distributes your public session key $P_1 = k_1\ G$. Analogously, the server selects k_2, compute and distribute P_2. Notice that, k_2 and P_2 are valid just for one session, if it's necessary another session, the server and the user has to select others secret and public sessions keys.

4.4. Storing Encrypted Data

In this subsection we describe how the data is stored in the server. Let $S(A_1, ..., A_m)$ be a relation scheme, $(A_1, ..., A_m)$ be its attributes, and I_s be an instanced relation over the scheme S. Given a tuple $t \in I_s$, the value of the attribute A_i in t, denoted by v_{ti}, is stored in the ciphered form $Y_{ti} = E(v_{ti}) + n_{Sti}\ P_S$, where $E(v_{ti}) \in E_p(a,b)$ is the point, on the elliptic curve,

corresponding to v_{ti}. The n_{Sti} value is a secret key used by the server to store data. For security reasons, the n_{Sti} value must be different for each relation, tuple and attribute. We denote also that P_S is the server's public key.

I_s			⇒	Encrypted I_s (EI_s for short)		
A_1	…	A_m	⇒	$E(A_1) + n_{S1}\ P_S$	…	$E(A_m) + n_{Sm}\ P_S$
v_{11}	…	v_{1m}	⇒	$E(v_{11}) + n_{S11}\ P_S$	…	$E(v_{1m}) + n_{S1m}\ P_S$
v_{21}	…	v_{2m}	⇒	$E(v_{21}) + n_{S21}\ P_S$	…	$E(v_{2m}) + n_{S2m}\ P_S$
…	…	…	⇒	…	…	…

Table 4: An instance of a relation with scheme $S(A_1, ..., A_m)$ (denoted by I_s) and their correspondent encrypted relation instance (denoted by EI_s).

Notice that in the first row of Table 4, we encrypt the relation scheme i.e., the name of each attribute A_i, as $Y_i = E(A_i) + n_{Si}\ P_S$. The tuple $(v_{t1}, ..., v_{tm})$ is encrypt as $(E(v_{t1}) + n_{St1}\ P_S, ..., E(v_{tm}) + n_{Stm}\ P_S)$.

T_{AS1}, T_{SA1}	…	T_{ASm}, T_{SAm}
T_{AS11}, T_{SA11}	…	T_{AS1m}, T_{SA1m}
T_{AS21}, T_{SA21}	…	T_{AS2m}, T_{SA2m}
…	…	…

Table 5: User A Transformer Table (T_AEI_s for short).

Considering a encrypted relation EI_s and a user A, the server has to store for each tuple and attribute two transformer points, T_{ASti} and T_{SAti}, defined by: $T_{ASti} = n_{Sti}\ P_S - n_A\ P_A + S_A$ and $T_{SAti} = n_A\ P_A - n_{Sti}\ P_S - S_A$. In the previous subsection we described how to compute these transformer points. These points are stored in a table called User A Transformer Table (T_AEI_s), shown in Table 5. Each cell of this table contains: T_{ASti} and T_{SAti}.

4.5. Data Exchange

In this subsection we present how the user A and the server can make data exchange. In the Table 6 we presents a summary of Data Exchange. In the Figure 2 we show a diagram of Data Exchange.

Phase	User A	Server
Send	Encrypt a computing $Y_{Aa} = E(a) + W_A - S_A$ Encrypt Y_{Aa} computing $X_1 = k_1\ G$ and $Y_1 = E(Y_{Aa}) + k_1\ P_2$ Send to server X_1 and Y_1	
Process		Decrypt X_1 and Y_1 in $Y_{Aa} = Y_1 - k_2\ X_1$ Process the query transforming Y_{Aa} in Y_i computing $Y_{ti} = Y_{Aa} + T_{ASti}$ Transform the select values to user A view computing $Y_{Aa} = Y_{ti} + T_{SAti}$ Encrypt Y_{Aa} computing $X_2 = k_2\ G$ and $Y_2 = E(Y_{Aa}) + k_2\ P_1$ Send to user A X_2 and Y_2
Receive	Decrypt X_2 and Y_2 in $Y_{Aa} = Y_2 - k_1\ X_2$ Decrypt Y_{Aa} in $a = Y_{Aa} - k_A\ X_A + S_A$	

Table 6: Data Exchange Summary.

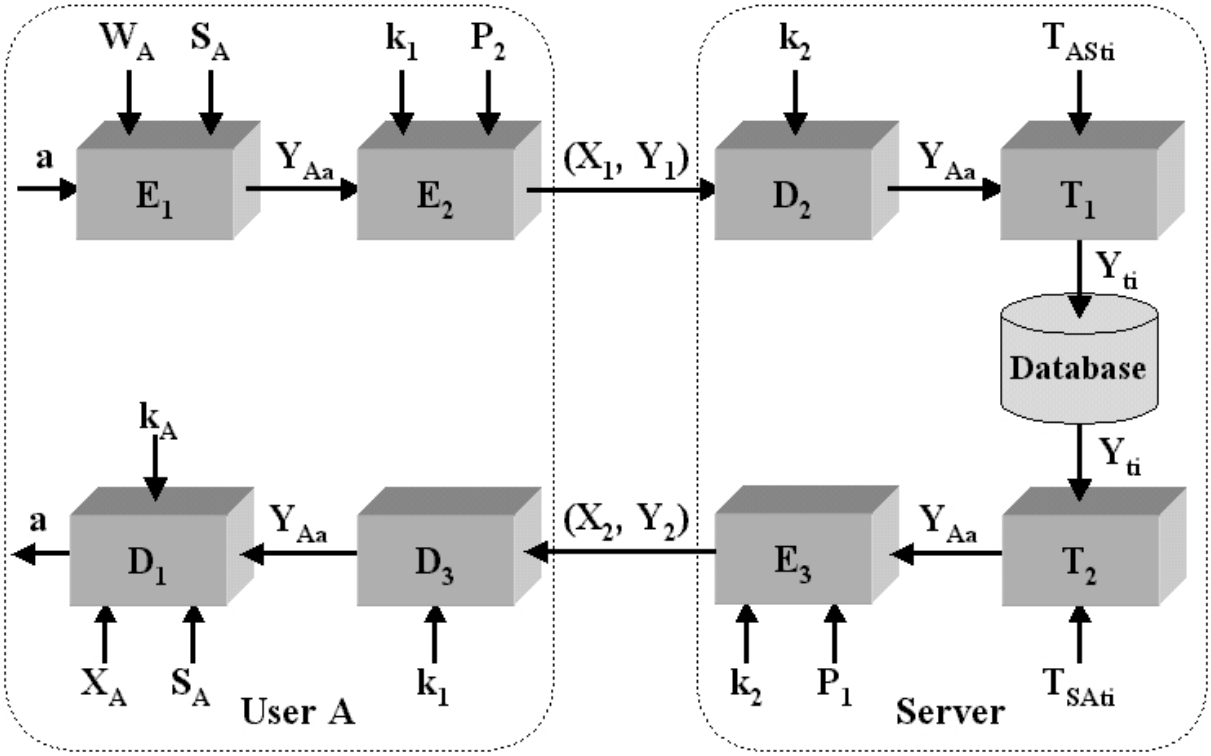

Figure 2: Data Exchange Diagram.

In Figure 2, operation **E1** encrypts the plain text a in Y_{Aa}, computing $Y_{Aa} = E(a) + W_A - S_A$, and operation **D1** decrypt it, computing $a = Y_{Aa} - k_A X_A + S_A$. These operations need to be made completely offline (without communication). We need to protect a user A's workstation during these operations. We do not discuss here how to guarantee the workstation security at the moment of these encrypting or decrypting. The operation **E2** encrypts Y_{Aa} in (X_1, Y_1), computing $X_1 = k_1 G$ and $Y_1 = E(Y_{Aa}) + k_1 P_2$, and then sends this pair of points to the server. The operation ***D2*** is the inverse of ***E2***, it decrypt (X_1, Y_1) in Y_{Aa}, computing $Y_{Aa} = Y_1 - k_2 X_1$. Then operation T_1 transforms Y_{Aa} in the server view Y_{ti}, computing $Y_{ti} = Y_{Aa} + T_{ASti}$. At this time the server can process the query. The query result Y_{ti} is transformed in Y_{Aa} by T_2 operation, computing $Y_{Aa} = Y_{ti} + T_{SAti}$. The point Y_{Aa} is encrypted in (X_2, Y_2) and sent to the user A by E_3 operation, computing $X_2 = k_2 G$ and $Y_2 = E(Y_{Aa}) + k_2 P_1$. The user A receives (X_2, Y_2) and decrypts it into Y_{Aa} by D_3 operation, computing $Y_{Aa} = Y_2 - k_1 X_2$. The operation D_1 decrypts Y_{Aa} in plain text a, computing $a = Y_{Aa} - k_A X_A + S_A$.

The operations T_1 and T_2 transform Y_{Aa} in Y_{ti} and vice-versa using T_{ASti} and T_{SAti}. The points T_{ASti} and T_{SAti} are public, so these operations can be made online. The operation T_1 has the following result:

$$Y_{ti} = Y_{Aa} + T_{ASti}$$
$$Y_{ti} = E(a) + W_A - S_A + n_{Sti} P_S - n_A P_A + S_A$$
$$Y_{ti} = E(a) + n_A P_A + n_{Sti} P_S - n_A P_A$$
$$Y_{ti} = E(a) + n_{Sti} P_S$$

In this form Y_{ti} is ready to be stored or to be processed by the server. The point Y_{Aa} is transformed in the point Y_{ti} without deciphering any data. Periodically, to enforce system security, the server can change the value n_{Sti} to a new n_{Sti1} computing a new Y_{ti1}, such that $Y_{ti1} = Y_{ti} - n_{Sti} P_S + n_{Sti1} P_S$.

The operation T_2 has the following result:

$$Y_{Aa} = Y_{ti} + T_{SAti}$$
$$Y_{Aa} = E(a) + n_{Sti}\, P_S + n_A\, P_A - n_{Sti}\, P_S - S_A$$
$$Y_{Aa} = E(a) + W_A - S_A$$

After D_3 operation, to decrypt Y_{Aa} received from the server, the user A needs his secret key k_A, the point X_A previously sent by the server and the shared (by user A and the server) secret key S_A. To decipher Y_{Aa} user A computes $Y_{Aa} - k_A\, X_A + S_A$.

$$\begin{aligned} Y_{Aa} - k_A\, X_A + S_A &= E(a) + W_A - S_A - k_A\, n_A\, G + S_A \\ &= E(a) + n_A\, P_A - k_A\, n_A\, G \\ &= E(a) + n_A\, k_A\, G - k_A\, n_A\, G \\ &= E(a) \end{aligned}$$

4.6. Query's Evaluation

Given an encrypted relation instance, to guarantee the database system security, the server has to compute the relational algebra operations without deciphering the stored data in the main or secondary memory. Next, we present a cryptographic protocol, which evaluate a complete set of relational algebra operations, which includes: project, union, difference, select, and cartesian product. This protocol maintains all the data in the encrypted form, even in the main memory.

4.6.1. The Project Operation

If I_s is an instance over a scheme $S(A_1, ..., A_m)$ then the project operation $\Pi_{Ai}(I_s)$ returns all values of the attribute A_i in the relation I_s. In the encrypted instance EI_s, the names of the attributes, denoted by $(A_1, ..., A_m)$, are also encrypted (see the first line in Table 4). To send an attribute name A_i to the server, the user A must cipher (offline) the attribute name computing: $Y_{Ai} = E(A_i) + W_A - S_A$. On the other hand the server searches the attribute name in the first tuple of EI_s. It transforms Y_{Ai} in Y_i computing $Y_i = Y_{Ai} + T_{ASi}$, as we described in subsection 4.5 (Data Exchange). In other words, the server receives the attribute name A_i ciphered in user A view (Y_{Ai}). After that, the server transforms to his view (Y_i) and it can find the attribute name in the first tuple of EI_s, just by comparing the points. If the server finds the right, it then returns all values of the attribute. As one can see, the server can process any project operation without decrypting any received or stored data.

4.6.2. Union and Difference Operations

Let be two encrypted instances EI_{s1} and EI_{s2}, the union $I_{s1} \cup I_{s2}$, and the difference $I_{s1} - I_{s2}$ can be evaluated by using attribute identification over encrypted instances as described in the project operation. After attribute identification, union and difference operations go on analogously to traditional operations. In these operations, the server does not have to decrypt any stored data.

4.6.3. The Select Operation

Let be a relation instance I_s, over a relation scheme $S(A_1, ..., A_m)$, the select operation $\sigma_{pred}(I_s)$ selects tuples in I_s which satisfy the predicate ***pred***. If ***pred*** is an equal or different predicate, $\sigma_{Ai = v}(I_s)$ or $\sigma_{Ai \neq v}(I_s)$, the select operation can be evaluated as follows. In this case, the user A sends the attribute name A_i and the value v to the server. For security reasons, user A must

encrypt these data as described in subsection 4.5 (Data Exchange). The values v and A_i are encrypted producing points Y_{Av} and Y_{Ai} respectively.

When the server receives the points Y_{Av} and Y_{Ai}, it identifies the A_i attribute name in encrypted instance EI_s, as described in project operation. The server searches the value v in a tuple t, transforming the point Y_{Av} (received from user A, corresponding to the value v) in Y_v computing $Y_v = Y_{Av} + T_{ASti}$. Finally, the server can compare the point Y_v (computed in previous step), with the point Y_{ti} (stored in tuple t and attribute i of EI_s), and return the tuples where $Y_v = Y_{ti}$ or $Y_v \neq Y_{ti}$, respectively.

The select operation $\sigma_{Ai \geq v}(I_s)$ returns all tuples $t \in I_s$ such that $t[i] \geq v$. We propose a cryptographic system protocol to deal with select operations. In this protocol the A_i attribute domain is limited to natural numbers (this is not a serious limitation, and the protocol can be extended to other enumerable sets).

Let $\sigma_{Ai \geq v}(I_s)$ be a select operation where v is a natural number such that $0 \leq v < r$, (r is the maximum natural number represented in the system). The select operation $\sigma_{Ai \geq v}(I_s)$ can be evaluated in the server by the protocol.

1. Identify the A_i attribute in the encrypted instance EI_s as in a project operation;
   If t is a tuple in EI_s, let Y_{ti} be the value of tuple t in the position of attribute A_i.
2. For each tuple t in EI_s;
   a. Transform Y_{Av} into Y_v, the server viewpoint, computing $Y_v = Y_{Av} + T_{ASti}$. Where T_{ASti} is the point that transform the data Y_{Av} from the user A to the server view Y_v.
   b. Compute $Y_{ti} - Y_v + G$, where Y_{ti} is the stored point in tuple t, attribute A_i in relation EI_s. In this case we have
      $Y_{ti} - Y_v + G = [E(v_{ti}) - n_{Sti} P_S] - [E(v) - n_{Sti} P_S] + G = E(v_{ti}) - E(v) + G$
      $= E(v_{ti} - v) = (v_{ti} - v + 1) G$.
   c. If the point $(v_{ti} - v + 1) G$ corresponds to a point of type $(j + 1) G$, $0 \leq j < r$, then $v_{ti} > v$. Remember that r is greater than the maximum integer value stored in the instanced relation I_s, and less than $n/2$ (n is the order of G).
   d. If $v_{ti} > v$, the tuple t is selected.

In step 2.c, we can check if $(v_{ti} - v + 1) G$ corresponds to a point of type $(j + 1) G$, $0 \leq j < r$, computing $(j + 1) G$, for $j = 0, 1, ..., r - 1$; and comparing if $(v_{ti} - v + 1) G = (j + 1) G$ for some j. In this case, the points $(j + 1) G$, for $j = 0, 1, ..., r - 1$, can be computed and stored previously in a table T. So, to compare if $(v_{ti} - v + 1) G = (j + 1) G$ for some j, it's necessary only to search the table T to find the value $(v_{ti} - v + 1) G$.

The select operations $\sigma_{pred}(I_s)$ where the predicate $pred$ includes operations $<$, $\leq$ and $>$ are analogous. Let, for instance, $\sigma_{Ai > v}(I_s)$. In this case, we only have to change the interval $0 \leq j < r$ to $1 \leq j < r$, in step 2.c of the protocol. If the $pred$ includes logical connectives and, or, not, we can evaluate single relative clauses as discussed above and apply intersection, union and different operations according to the logical connectives.

4.6.4. The Cartesian Product Operation

The cartesian product operation over two relation instances I_{s1} and I_{s2}, denoted by $(I_{s1} \times I_{s2})$, can be evaluated by joining the encrypted versions EI_{s1} and EI_{s2}. Each tuple in EI_{s1} is joined by all tuples in EI_{s2}.

5. Conclusions

In this paper, we proposed a multi-user key and data exchange protocol to manipulate stored encrypted data without decrypting. The system's security is based on the following properties. All the stored data is encrypted. The server is secure enough to answer queries without decrypting any data or metadata in the main or secondary memory. In this sense our protocol is an extension of the Kerberos protocol. The proposed protocol uses data in ciphered form the whole time. In the proposed protocol ciphered data in a certain user's view is transformed into ciphered data in the server view and vice-versa, without decrypting the original data. The new mechanism, proposed here, solves both the multi-user data exchange and protects the server leaving it free to answer queries without deciphering any data.

We applied the proposed protocol to DBMS. The DBMS uses cryptographic protocols to maintain all encrypted data and does not require any external security mechanisms or any tools like firewalls. The security data system is equivalent to the cryptographic security system it utilizes. In this case, the cryptographic system is based on elliptic curves.

The user and server must trust one another bilaterally. This proposed system falls apart under a known-plaintext attack by trusted users against the server. The user knows the transmitted plaintext, and then it can learn the secret key used to encrypt the cell data. In this context the user must avoid a known-plaintext attack by an intruder. We are presently working on this problem, adding some extensions to the protocol, as well as higher efficiency and updating issues.

References

[1] N. Ahituv, Y. Lapid, and S. Neumann, Processing Encrypted Data, *Communications of the ACM*, 30(9):777-780, 1987.

[2] I. Blake, G. Seroussi, and N. Smart, *Elliptic Curves in Cryptography*, Cambridge University Press, 1999.

[3] M. Blaze and M. Strauss, *Atomic Proxy Cryptography*, EuroCrypt'98, 1997.

[4] S. Castano, M. Fugini, G. Martella, and P. Samarati, *Database Security*, Addison-Wesley, 1994.

[5] R. Elmasri and S. B. Navathe, *Fundamentals of Database Systems*, Addison-Wesley, 3rd edition, 2000.

[6] J. Feigenbaum, M. Y. Liberman, E. Grosse, and J. A. Reeds, Cryptography Protection of Membership Lists, *Newsletter of the International Association of Cryptography Research*, 9(1):16-20, 1992.

[7] J. Feigenbaum, M. Y. Liberman, and R. N. Wright, Cryptography Protection of Database and Software, *American Mathematical Society of Distributed Computing and Cryptography*, 0(1):161-172, 1991.

[8] N. Koblitz, *Algebraic Aspects of Cryptography*, Springer Verlag, 1999.

[9] A. J. Menezes, P. C. van Oorshot, and S. A. Vastone, *Handbook of Applied Cryptography*, CRC Press, 1997.

[10] M. Rosing, *Implementing Elliptic Curve Cryptography*, Manning, 1999.

[11] B. Schneier, *Applied Cryptography*, John Willey, 1996.

[12] A. Silberschatz, H. F. Korth, and S. Sudarshan, *Database Systems*, McGraw-Hill, 1997.

[13] W. Stallings, *Cryptography and Network Security: Principles and Practice*, Prentice Hall, 1999.

[14] D. R. Stinson, *Cryptography, Theory and Practice*, CRC Press, 1995.

[15] R. E. Smith, *Internet Cryptography*, Addison-Wesley, 1997.

Appendix – Example of Didactical Application

The Table 7 shows part of an elliptic group with 240 points defined as follow.

j	*E(j)*	*j*	*E(j)*	*j*	*E(j)*	*j*	*E(j)*	*j*	*E(j)*
0	**(2, 2)**	3	(159, 114)	6	(179, 199)	9	(75, 90)	66	(131, 127)
1	(5, 200)	4	(153, 108)	7	(174, 163)	...	...	...	...
2	(129, 56)	5	(125, 152)	8	(111, 145)	65	(79, 136)	239	(2, 209)

Table 7: Elliptic Group generated by $\boldsymbol{G}$ = (2, 2).

In this example, we consider the curve ($y^2 = x^3 + a\,x + b$) presented in page 197 of Stallings [13], where $\boldsymbol{p = 211}$, $\boldsymbol{a = 0}$, and $\boldsymbol{b = 207}$. Notice that: $\boldsymbol{a < p}$, $\boldsymbol{b < p}$, and $\boldsymbol{(4a^3 + 27b^2) \bmod p \neq 0}$. We take the generator point $\boldsymbol{G = (2, 2)}$ as the representation of the number $\boldsymbol{0}$ denoted by $\boldsymbol{E(0) = G}$. We consider $\boldsymbol{E(1) = G + G = 2\,G}$, etc. In this case we have $\boldsymbol{n = 241}$ and $\boldsymbol{r = 120}$. Thus we can represent natural numbers from 0 to 119. The Table 7 shows the representation of points $\boldsymbol{j}$ such that $\boldsymbol{0 \leq j \leq 239}$. Let us suppose a relational table $\boldsymbol{I_s}$ to be ciphered and the corresponding ASCII code:

I_s	
A	B
9	4
1	3
8	7

Using ASCII table for attributes names we have:

I_s ASCII	
65	66
9	4
1	3
8	7

The Table 8 summarizes the chosen and calculated variables we need to cipher $\boldsymbol{I_s}$. In this example we don't show the use of $\boldsymbol{E_2}$, $\boldsymbol{D_2}$, $\boldsymbol{E_3}$ and $\boldsymbol{D_3}$ operations, because they are using classical ECC cryptography and to understand the proposed system, this isn't necessary.

<table>
<tr>
<td>Chosen by user A:
$\boldsymbol{k_A}$ = 3

Calculated by user A:
$\boldsymbol{P_A} = \boldsymbol{k_A}\,\boldsymbol{G}$ = 3 (2, 2) = (129, 56)
$\boldsymbol{S_A} = \boldsymbol{k_A}\,\boldsymbol{P_S}$ = 3 (159, 114) = (155, 96)</td>
<td>Chosen by server:
$\boldsymbol{k_S}$ = 4
$\boldsymbol{n_A}$ = 2
$\boldsymbol{n_{Sti}}$
<table>
<tr><td>$\boldsymbol{n_{S1}}$ = 1</td><td>$\boldsymbol{n_{S2}}$ = 2</td></tr>
<tr><td>$\boldsymbol{n_{S11}}$ = 5</td><td>$\boldsymbol{n_{S12}}$ = 8</td></tr>
<tr><td>$\boldsymbol{n_{S21}}$ = 6</td><td>$\boldsymbol{n_{S22}}$ = 9</td></tr>
<tr><td>$\boldsymbol{n_{S31}}$ = 3</td><td>$\boldsymbol{n_{S32}}$ = 4</td></tr>
</table>
$\boldsymbol{G}$ = (2, 2) $\Rightarrow$ $\boldsymbol{n}$ = 241 $\Rightarrow$ $\boldsymbol{r}$ = 120
Calculated by server:
$\boldsymbol{P_S} = \boldsymbol{k_S}\,\boldsymbol{G}$ = 4 (2, 2) = (159, 114)
$\boldsymbol{S_A} = \boldsymbol{k_S}\,\boldsymbol{P_A}$ = 4 (129, 56) = (155, 96)
$\boldsymbol{W_A} = \boldsymbol{n_A}\,\boldsymbol{P_A}$ = 2 (129, 56) = (125, 152)
$\boldsymbol{X_A} = \boldsymbol{n_A}\,\boldsymbol{G}$ = 2 (2, 2) = (5, 200)
<table>
<tr><td rowspan="4">$\boldsymbol{T_{ASti}}$ and $\boldsymbol{T_{SAti}}$</td><td>(75, 90), (75, 121)</td><td>(201, 85), (201, 126)</td></tr>
<tr><td>(51, 136), (51, 75)</td><td>(130, 8), (130, 203)</td></tr>
<tr><td>(70, 200), (70, 11)</td><td>(45, 179), (45, 32)</td></tr>
<tr><td>(198, 139), (198, 72)</td><td>(84, 210), (84, 1)</td></tr>
</table>
</td>
</tr>
</table>

Table 8: Example of chosen and calculated variables.

We compute $\boldsymbol{T_{ASti} = n_{Sti}\,P_S - n_A\,P_A + S_A}$ and $\boldsymbol{T_{SAti} = n_A\,P_A - n_{Sti}\,P_S - S_A}$.

$$T_{ASti} = n_{Sti}\,P_S - n_A\,P_A + S_A = n_{Sti}\,P_S + S_A - n_A\,P_A = n_{Sti}\,P_S + (155, 96) - 2\,(129, 56)$$
$$= n_{Sti}\,P_S + (155, 96) - (125, 152) = n_{Sti}\,P_S + (125, 152)$$

$T_{SAti} = n_A P_A - n_{Sti} P_S - S_A = n_A P_A - S_A - n_{Sti} P_S = 2\ (129, 56) - (155, 96) - n_{Sti} P_S$
$= (125, 152) - (155, 96) - n_{Sti} P_S = (125, 59) - n_{Sti} P_S$

The Table 9 shows the resulting calculation $\boldsymbol{n_{Si} P_S}$ and $\boldsymbol{n_{Sti} P_S}$.

(159, 114)	(174, 163)
(54, 138)	(136, 11)
(192, 201)	(96, 3)
(155, 96)	(181, 209)

Table 9: $\boldsymbol{n_{Si} P_S}$ and $\boldsymbol{n_{Sti} P_S}$.

In the Table 10 we show the computation of $\boldsymbol{Y_{ti} = E(v_{ti}) + n_{Sti} P_S}$, made by the server.

$\boldsymbol{I_s}$		Computing $\boldsymbol{EI_s}$		Tuple	$\boldsymbol{EI_s}$	
65	66	(79, 136) + (159, 114)	(131, 127) + (174, 163)	0	(209, 58)	(32, 108)
9	4	(75, 90) + (54, 138)	(153, 108) + (136, 11)	1	(70, 200)	(186, 14)
1	3	(5, 200) + (192, 201)	(159, 114) + (96, 3)	2	(51, 136)	(128, 21)
8	7	(111, 145) + (155, 96)	(174, 163) + (181, 209)	3	(27, 30)	(192, 201)

Table 10: Computation witch transforms $\boldsymbol{I_s}$ in $\boldsymbol{EI_s}$.

Now, let us consider the following query $\boldsymbol{\sigma_{A \geq 3}(I_s)}$. We have to encrypt the attribute name "A" and the number "3" in the user A view. We can do that computing $\boldsymbol{Y_{Av} = E(v) + W_A - S_A}$.

- For the attribute name A we have: $\boldsymbol{Y_{AA}} = (79, 136) + (125, 152) - (155, 96) = (116, 114)$.
- For the number 3 we have: $\boldsymbol{Y_{Av}} = (159, 114) + (125, 152) - (155, 96) = (5, 11)$.

Now we can apply the selection operation algorithm.

1. To identify the attribute name A in the tuple 0 of $\boldsymbol{EI_s}$, the server receives the point ***(116, 114)***, ciphered in the user A view. The server has to convert to your view computing $\boldsymbol{Y_1 = Y_{AA} + T_{AS1}} = (116, 114) + (75, 90) = \boldsymbol{(209, 58)}$. This point corresponds to the first attribute in the first tuple of $\boldsymbol{EI_s}$.
2. The point $\boldsymbol{Y_{Av}} = \boldsymbol{(5, 11)}$ corresponds to the number 3 in the select operation $\boldsymbol{\sigma_{A \geq 3}(I_s)}$, and is ciphered in the user A view. The server has to do:
   For the tuple number 1:
   a. Compute $\boldsymbol{Y_v = Y_{Av} + T_{AS11}} = (5, 11) + (51, 136) = (192, 201)$.
   b. Compute $\boldsymbol{Y_{11} - Y_v + G} = (70, 200) - (192, 201) + (2, 2) = (179, 199)$.
   c. The point (179, 199) corresponds a point of type $\boldsymbol{(j + 1)\ G}$ for some $\boldsymbol{j}$, $\boldsymbol{1 \leq j \leq 120}$? Yes, it corresponds to the point 6 (2, 2). Notice that 6 = 9 (the stored value) – 3 (the inputted value).
   d. This tuple is selected.

   For the tuple number 2:
   a. Compute $\boldsymbol{Y_v = Y_{Av} + T_{AS21}} = (5, 11) + (70, 200) = (64, 194)$.
   b. Compute $\boldsymbol{Y_{21} - Y_v + G} = (51, 136) - (64, 194) + (2, 2) = (2, 209)$.
   c. The point (2, 209) corresponds a point of type $\boldsymbol{(j + 1)\ G}$ for some $\boldsymbol{j}$, $\boldsymbol{1 \leq j \leq 120}$? We have (2, 209) = 240 (2, 2) and $\boldsymbol{240 > 120}$ when $\boldsymbol{r = 120}$.
   d. This tuple is not selected.

   For the tuple number 3:
   a. Compute $\boldsymbol{Y_v = Y_{Av} + T_{AS31}} = (5, 11) + (198, 139) = (181, 209)$.
   b. Compute $\boldsymbol{Y_{31} - Y_v + G} = (27, 30) - (181, 209) + (2, 2) = (125, 152)$.

c. The point (125, 152) corresponds a point of type $(j + 1)\,G$ for some j, $1 \leq j \leq 120$? Yes, 5 (2, 2) = (125, 152).
d. This tuple is selected.

In this example we select the tuples 1 and 3. In summary:

	Tuple	Selected over EI_s in the server view	
Attributes Names ⇒	0	(209, 58)	(32, 108)
	1	(70, 200)	(186, 14)
	3	(27, 30)	(192, 201)

This answer has to be sent to the user A. In this case, the server has to revert Y_{ti} to Y_{Aa} computing: $Y_{Aa} = Y_{ti} + T_{SAti}$, resulting:

Tuple	Selected tuples over EI_s in the user A view	
0	(209, 58) + (75, 121) = (116, 114)	(32, 108) + (201, 126) = (60, 115)
1	(70, 200) + (51, 75) = (159, 114)	(186, 14) + (130, 203) = (2, 209)
3	(27, 30) + (198, 72) = (129, 56)	(192, 201) + (84, 1) = (5, 200)

The user A decrypts the query answer received from the server computing $Y_{Aa} - k_A X_A + S_A$ for each element of query answer. Notice that:

$$Y_{Aa} - k_A X_A + S_A = Y_{Aa} + S_A - k_A X_A = Y_{Aa} + (155, 96) - 3\,(5, 200)$$
$$= Y_{Aa} + (155, 96) - (125, 152) = Y_{Aa} + (125, 152)$$

Then we have:

User A deciphering query answer	
(116, 114) + (125, 152) = (79, 136) ⇒ 65	(60, 115) + (125, 152) = (131, 127) ⇒ 66
(159, 114) + (125, 152) = (75, 90) ⇒ 9	(2, 209) + (125, 152) = (153, 108) ⇒ 4
(129, 56) + (125, 152) = (111, 145) ⇒ 8	(5, 200) + (125, 152) = (174, 163) ⇒ 7

Using ASCII table to revert the attributes names we have:

Numeric Query Answer		⇒	Query Answer	
65	66		A	B
9	4		9	4
8	7		8	7

Notice that the query answer is the correct result of selection $\sigma_{A \geq 3}(I_s)$.