Secure Mediation of Join Queries by Processing Ciphertexts

Joachim Biskup, Christian Tsatedem* and Lena Wiese
Universität Dortmund
Fachbereich Informatik
44221 Dortmund
Germany
E-mail: {biskup,tsatedem,wiese}@ls6.cs.uni-dortmund.de

Abstract

In a secure mediated information system, confidentiality is one of the main concerns when transmitting data from datasources to clients via a mediator. We present three approaches that allow a mediator to compute a JOIN operation on encrypted relations; each approach uses a different encryption scheme. We adapt these schemes to the mediator architecture and provide a comprehensive description of the resulting protocols. A first security analysis is also given.

Key words: secure mediation, confidentiality, credential based access control, encrypted data processing, database as a service, commutative encryption, homomorphic encryption

1. Introduction

One crucial task in collaborative environments is finding, combining and filtering information from different (possibly heterogeneous) datasources. Clients in search of information may be unsure which datasources hold the relevant data and it is inconvenient for them to query several datasources separately; in a dynamic system with changing number or location of datasources this situation gets even worse. To facilitate the search process for clients, a new entity called *mediator* was introduced in such environments (see e.g. [23]). In a mediated system, the client issues a global query to the mediator, the mediator passes partial queries to adequate datasources, retrieves their partial results and returns a global result combined from the partial results to the client (see Figure 1 for a basic mediated system). Moreover, a hierarchy of mediators is possible: a mediator can also serve as a datasource for other mediators.

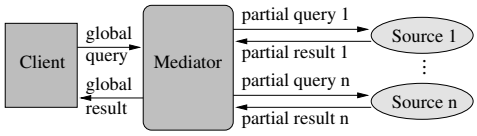

Figure 1. A basic mediated information system

A mediated system offers several benefits for clients as well as datasources. The mediator takes the burden of

- soliciting sufficient access control information (e.g. credentials) from the clients and passing relevant subsets of it to the datasources
- splitting the global query into partial queries
- choosing and contacting the relevant datasources
- generating the global result and returning it to the client

In an intra-enterprise setting a mediator might be one trusted, centralized entity. However, mediation offers a lot more in an inter-enterprise setting: In a dynamic environment with several loosely coupled participants (clients, datasources and mediators) that do not trust each other, contract based confederations can be built. That is, clients can dynamically sign up for a mediation service and mediators can flexibly contract several datasources to supply their data.
Primarily in this dynamic inter-enterprise setting (but possibly also in the intra-enterprise setting), there are some serious security concerns like for example confidentiality and integrity of data, reliability and availability of services or anonymity of participants. This led to systems for *secure mediation* like the multimedia mediator (MMM) system (see [3]); other secure mediation systems are for example Hermes [5] and Chaos [17].

*C.T.'s current affiliation is Fraunhofer SCAI, 53754 Sankt-Augustin.

In this paper we focus on the *confidentiality* of the data sent from the datasources via the mediator to the client in the MMM system. Obviously we face the problem that the untrusted mediator should generate the global result from the partial results without learning anything about the data it processes. A previous solution to this problem was mobile code (see [4]): The mediator sends the client an executable that computes the global result from the partial results (after decryption of the partial results). In this approach, the client is left with the task of executing the mobile code and it gives rise to additional security threats like for example malicious code. Thus, the optimal solution would be to let the mediator compute an *encrypted global result* from the *encrypted partial results*. In this paper we present a first step toward this computation on encrypted data: We extend the MMM with a mechanism to compute a JOIN operation over encrypted relations that yields an appropriately encrypted joined relation as the result.

This paper is organized as follows: First of all, we briefly describe the MMM system in Section 2. In the following Sections 3 to 5 we present three different encryption schemes, incorporate them into the MMM system and describe a JOIN operation on encrypted relations for each scheme; a first, rudimentary comparison of the schemes is given in Section 6. In Section 7 we relate our work to general approaches for computation on encrypted data. We conclude the paper with suggestions for future work.

2. The Multimedia Mediator

In the MMM system datasources execute access control based on a set of credentials. Therefore, before sending a query, each client has to have an appropriate set of credentials issued by a trusted certification authority. Each credential links properties of the client to one of his public encryption keys but in general does not contain details on his identity; the client keeps other certificates linking his identity to each public key in a safe place to enable identification in case it is needed – for example in a legal dispute. This acquisition of credentials is part of the *preparatory phase* of the MMM protocol as thoroughly described in [3].

Now, the client can start the *request phase*: When issuing a global query to a mediator, the client attaches a set of credentials to the query. The mediator splits the global query into partial queries; SQL queries for instance can be transformed into a so-called "algebra tree" (with relational operators in the inner nodes of the tree and partial queries at the leaves) by using the "SQL2Algebra" library; see [4] for a description. With each partial query, the mediator forwards a subset of the credentials to an appropriate datasource. Datasources base their access control decisions only on the properties presented in the credentials. If the presented credentials suffice to grant data access, the datasources evaluate the partial queries. In case the credentials do not allow full data access, the partial results might be filtered in order to return only those records for which access permissions exist. See Figure 2 for a schematic credential-based MMM system (for simplicity with just one credential shown).

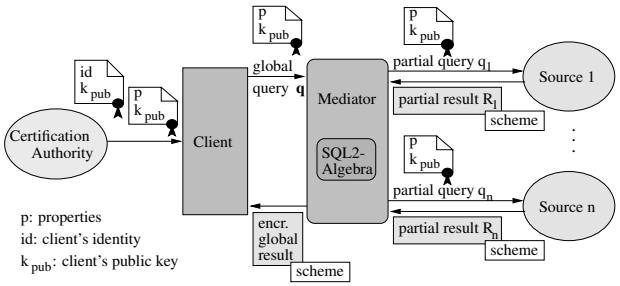

Figure 2. A credential-based MMM system

Finally, the *delivery phase* begins: With one of the three encryption schemes we present in the following, each datasource encrypts its partial result in such a way that the mediator can compute the JOIN over the encrypted partial results and thus the generation of an encrypted global result is possible. The public keys in the credentials can be used by the datasources to send information (basically encrypted partial results and additional data necessary for the decryption step at client side) securely via the mediator to the client. This information is best encrypted with a hybrid encryption scheme; that is, the information is encrypted with a newly generated symmetric session key and the session key is encrypted with the public keys of the client. We denote as $encrypt(\ldots)$ and $decrypt(\ldots)$ the according hybrid encryption and decryption functions.

In this paper we confine ourselves to queries **q** that can be split into one JOIN operation and two partial queries q_1 and q_2 over two relations R_1 and R_2 managed by datasources S_1 and S_2, respectively. To keep things simple, we assume that the partial queries are just "`select *`"-queries although more complex queries could be executed by the datasources.

In the MMM system, the mediator combines the heterogeneous database schemas of the datasources into one homogeneous global schema (via a so-called embedding; see [2]). That is why the mediator can identify the sets $\mathcal{A}_1$ and $\mathcal{A}_2$ of attributes (of relations R_1 and R_2, respectively) that have to be considered in the JOIN operation. In this paper, we assume that there is just one join attribute A_{join} common to R_1 and R_2; that is, $\mathcal{A}_1 = \mathcal{A}_2 = \{A_{join}\}$. If a distinction is necessary, we qualify the join attribute with the relation names: $R_1.A_{join}$ and $R_2.A_{join}$. Listing 1 shows the basic request phase for a JOIN query. The delivery phases of our new protocols vary according to the encryption scheme

1. The client sends query **q** requiring the JOIN of the relations R_1 and R_2 with a set of credentials CR to the mediator.
2. The mediator localizes the appropriate datasources S_1 and S_2 and decomposes **q** into partial queries q_1 ="`select * from` R_1" and q_2 ="`select * from` R_2". The mediator also selects appropriate subsets CR_1 and CR_2 of credentials, which are required to execute the queries q_1 and q_2.
3. For $i \in \{1, 2\}$, let $\mathcal{A}_i$ be the set of join attributes of R_i (for simplicity, $\mathcal{A}_i = \{A_{join}\}$). The mediator sends the triple $\langle q_i, CR_i, \mathcal{A}_i \rangle$ to S_i.
4. S_i checks the credentials CR_i. If authorization is granted, query q_i is executed with R_i as the result.

Listing 1. Basic MMM request phase

used; they are presented in the following sections.

We finally assume that all parties in our protocols are *semi-honest* (see for instance [6]; also called *honest-but-curious*, see e.g. [1]). That is, each party exactly acts as specified in the protocol, but possibly tries to gain information about the other parties' inputs.

3. The Database-as-a-Service Model

The first approach for computation with encrypted data we consider is the work of Hacıgümüş et al. [13]. Their work is based on the fact that in the database-as-a-service model (DAS model) – where data is stored at a service provider site – the data owner does not trust the provider.
In their approach, the data owner outsources his data in encrypted form to the service provider. For a relation of the schema $R(A_1, A_2, ..., A_n)$, the encrypted relation has the schema $R^S(Etuple, A_1^S, A_2^S, ..., A_n^S)$. The attribute $Etuple$ stores encrypted representations of tuples $t = \langle a_1, ..., a_n \rangle$ of R, where a_i is an element of the domain $dom(A_i)$ of attribute A_i; encryption is done by the data owner with a standard encryption scheme. Each attribute A_i^S in R^S is called the *index* of the attribute A_i in R; these values are used for processing the encrypted data. The index values for an attribute A_i are defined by first dividing the active domain $dom_{active}(A_i)$ into partitions and then assigning a unique identifier to each partition; these identifiers can for example be computed with a collision free hash function that uses properties of the partition. The identifiers are used as the index values. When encrypting a relation, for a tuple with value a_i of attribute A_i, in the encrypted tuple the attribute A_i^S contains the index value of the partition that contains a_i.
A decryption function is also defined; we denote it as $decrypt_{DAS}(\ldots)$. For an encrypted tuple $t^S = \langle etuple, a_1^S, ..., a_n^S \rangle$, the function decrypts $etuple$ (according to the standard encryption scheme) and drops all index values a_i^S with $i = 1...n$.

When querying the data, the service provider is able to evaluate the query at least partially and thus returns a superset of the result to the client. That is why in the DAS model there is a *query translator*, which splits a query **q** into

1. a server query $\mathbf{q}^S$ over encrypted data using the index values to be run at the service provider site
2. a client query $\mathbf{q}^C$ for post-processing results of the server query at the data owner site

For details refer to the article by Hacıgümüş et al. [13].

3.1. Secure Mediation with the DAS Model

We will now adapt the DAS approach to the MMM system for secure mediation of JOIN queries; that is, we combine the basic protocol for secure mediation with the DAS encryption scheme.

First of all, there are a number of differences between the DAS model and the mediation system:

- There is one more layer (the mediator) in the mediation system; the mediator executes the server query $\mathbf{q}^S$ on the encrypted partial results of the datasources.
- The datasources are the data owners and encrypt their partial results according to the DAS approach.
- The client is not the data owner; he is merely querying the data and has to have a means of decrypting the client server result and executing the client query $\mathbf{q}^C$ on it to retrieve the global result.

With the DAS approach, a total computation of the natural join over encrypted relations cannot be achieved, as the client still has to execute the client query. However, this approach allows the mediator to partially evaluate the global query over encrypted relations. This leads to the question which participant should split the global query into server query and client query. In principle, it is possible to place the DAS query translator in any layer of the mediation system. We call the resulting settings *mediator setting* (query translator with the mediator), *source setting* (query translator with one of the datasources) and *client setting* (query translator with the client). In this article we only describe the *client setting*.

We now briefly describe the delivery phase of the MMM protocol with DAS encryption; in Listing 2 this phase is listed step by step.

1. S_i partitions the active domain $dom_{active}(A_{join})$ of the join attribute and maps each partition to an index value in $ITable_{R_i.A_{join}}$.
2. S_i encrypts R_i according to the DAS approach using the public keys from CR_i and the index table $ITable_{R_i.A_{join}}$ with R_i^S as the resulting encrypted relation. S_i encrypts $ITable_{R_i.A_{join}}$ using the public keys from CR_i; we call the resulting encrypted table $encrypt(ITable_{R_i.A_{join}})$.
3. S_i sends $\langle R_i^S, encrypt(ITable_{R_i.A_{join}})\rangle$ to the mediator.
4. The mediator sends $encrypt(ITable_{R_1.A_{join}})$ and $encrypt(ITable_{R_2.A_{join}})$ to the client.
5. The client decrypts $encrypt(ITable_{R_1.A_{join}})$ and $encrypt(ITable_{R_2.A_{join}})$. He translates $\mathbf{q}$ into $\mathbf{q}^S$ and $\mathbf{q}^C$ according to the DAS approach using $ITable_{R_1.A_{join}}$ and $ITable_{R_2.A_{join}}$. The client sends $\mathbf{q}^S$ to the mediator.
6. The mediator computes $\mathbf{q}^S$ on R_1^S and R_2^S with R^C as the result. The mediator sends R^C to the client.
7. The client decrypts R^C using his private keys according to the DAS approach and executes $\mathbf{q}^C$ on the decrypted relation.

Listing 2. DAS delivery phase (client setting)

A datasource S_i encrypts its partial result on tuple level (that is, row-wise) using the public keys of the client with a hybrid encryption scheme; in other words, each tuple t is encrypted to $encrypt(t)$. Index values only have to be computed for the join attribute; that is, only the domain $dom_{active}(R_i.A_{join})$ has to be partitioned. The mapping from partitions to index values is represented in a so-called "index table" $ITable_{R_i.A_{join}}$. From the encrypted tuples and the index values, datasource S_i builds a partial result R_i^S encrypted in DAS-like manner (that is, consisting of tuples of the form $t^S = \langle etuple, a_{join}^S\rangle$ where $etuple = encrypt(t)$ and a_{join}^S is the appropriate index value). Additionally, S_i encrypts the index table such that only the client can decrypt it (that is, preferably with the same session key as used for the partial result). Finally, S_i sends its encrypted partial result R_i^S and its encrypted index table $encrypt(ITable_{R_i.A_{join}})$ to the mediator.

The query translator needs both index tables to generate the server query $\mathbf{q}^S$ and the client query $\mathbf{q}^C$ from a client's global query $\mathbf{q}$. That is why in the client setting – with the query translator at client side – the mediator forwards the encrypted index tables of both datasources to the client. The client decrypts the index tables and the query translator generates the server query and the client query based on the index values (we denote R^C the result of the server query):

$$R^C := \mathbf{q}^S(R_1^S, R_2^S) = \sigma_{Cond_S}(R_1^S \times R_2^S)$$

where (for $i \in \{1,2\}$) for all partitions p_i and their index values $index(p_i)$ in $ITable_{R_i.A_{join}}$ such that $p_1 \cap p_2 \neq \emptyset$ $Cond_S =$

$$\bigvee_{p_1,p_2} (R_1^S.A_{join} = index(p_1) \wedge R_2^S.A_{join} = index(p_2)).$$

and

$$\mathbf{q}^C(decrypt_{DAS}(R^C)) = \sigma_{Cond_C}(decrypt_{DAS}(R^C))$$

where

$$Cond_C = (R_1.A_{join}^S = R_2.A_{join}^S)$$

That is, the server query generates a superset of the global result by combining encrypted tuples of R_1^S and R_2^S whose values of A_{join} belong to overlapping partitions in the two index tables.

Afterwards, the client sends the server query $\mathbf{q}^S$ to the mediator; the mediator applies it to the encrypted partial results R_1^S and R_2^S and sends the encrypted result R^C to the client. Since the mediator computes the server query on the encrypted partial results, the plaintexts are kept secret. Finally, the client decrypts R^C and applies the client query to it in order to get the global result.

4. Commutative Encryption

In this section we describe the approach for encrypted data processing by Agrawal et al. (see [1]). The authors consider two participants in the computation: a *sender* and a *receiver*. Both participants have an input relation and the receiver is to execute a relational operation on the input relations in such a way that he learns only those data from the sender's input relation that form part of the resulting relation – but without learning other data from the sender's input relation. The authors use a commutative encryption scheme and present interactive protocols for the operations intersection and join on encrypted relations. In these protocols, as additional information both receiver and sender learn the cardinality of the other participant's input relation.

In the following we define the commutative encryption function according to [1]. A commutative encryption function is a polynomial-time computable function

$$f_e : dom_f \longrightarrow dom_f$$

(where e is an adequate key and dom_f an adequate domain) with the following properties (the notation $\in_r$ means "chosen uniformly at random from"):

1. **[Commutativity]** For all keys e_1 and e_2

$$f_{e_1} \circ f_{e_2} = f_{e_2} \circ f_{e_1}.$$

2. **[Bijectivity]** Each f_e is a bijection.

3. **[Invertibility]** The inverse f_e^{-1} is polynomial-time computable given e.

4. **[Secrecy]** The distribution of $\langle x, f_e(x), y, f_e(y)\rangle$ is indistinguishable from the distribution of $\langle x, f_e(x), y, z\rangle$, where $x,\ y,\ z \in_r\ dom_f$ and e is a random key.

The secrecy property means that given a plaintext x and its corresponding ciphertext $f_e(x)$, for a new value y, it is not possible to distinguish $f_e(y)$ and a random value z in polynomial time. In [1], this property is used to prove the security of the protocols. To guarantee this property, the input parameters x and y of f have to be random values. For this purpose, the encryption function f is not executed on the original data in the input relation but on hash values of the same. The authors assume that the hash function is ideal, that is, computed by a random oracle. The hash function (written as h) maps values of the join attribute to values in an adequate domain dom_f. As an example domain, Agrawal et al. mention the set of quadratic residues modulo a safe prime and show that exponentiation can be used as a commutative encryption function. For details refer to [1].

4.1. Secure Mediation with Commutative Encryption

We now extend the basic MMM protocol with a mechanism for computing the JOIN operation on encrypted relations with the help of commutative encryption. Again we can see some differences between the two architectures. In the MMM system with commutative encryption

- there is not one distinguished receiver; instead the receiver's responsibilities are distributed between client, mediator and datasources
- unlike the original approach, datasources do not encrypt their partial results with a newly generated key (see [1]) before sending them to the mediator; instead they use our hybrid encryption scheme $encrypt$ so that only the client can decrypt the data
- the mediator now identifies the exact set of those tuples of the partial results that form the global result

For the relations R_1 and R_2 we define $Tup_i(a)$ to be the set of those tuples t in which the join attribute has value a:

$$Tup_i(a) := \{t \in R_i \mid t[A_{join}] = a\}$$

1. S_i chooses a secret key e_i of a commutative encryption scheme; for each $a \in dom_{active}(R_i.A_{join})$, it encrypts a's hash value with e_i: $f_{e_i}(h(a))$.
2. For each $a \in dom_{active}(R_i.A_{join})$, S_i encrypts the set $Tup_i(a)$ of tuples with the appropriate public keys of the client. The resulting ciphertexts are called $encrypt(Tup_i(a))$.
3. S_i sends the (arbitrarily ordered) set of messages $M_i := \{\langle f_{e_i}(h(a)), encrypt(Tup_i(a))\rangle | a \in dom_{active}(R_i.A_{join})\}$ to the mediator.
4. The mediator sends the set of messages M_1 to S_2 and the set of messages M_2 to S_1.
5. For each message $\langle f_{e_2}(h(a)), encrypt(Tup_2(a))\rangle$, S_1 computes $f_{e_1}(f_{e_2}(h(a)))$ and sends the set of all messages $\langle f_{e_1}(f_{e_2}(h(a))), encrypt(Tup_2(a))\rangle$ to the mediator.
6. For each message $\langle f_{e_1}(h(a)), encrypt(Tup_1(a))\rangle$, S_2 computes $f_{e_2}(f_{e_1}(h(a)))$ and sends the set of all messages $\langle f_{e_2}(f_{e_1}(h(a))), encrypt(Tup_1(a))\rangle$ to the mediator.
7. The mediator now checks for messages that have an identical first component, that is, messages where $f_{e_1}(f_{e_2}(h(a))) = f_{e_2}(f_{e_1}(h(a)))$; if this is the case, the mediator combines the second components of all such messages to result messages $\langle encrypt(Tup_1(a)), encrypt(Tup_2(a))\rangle$ and sends the set of all result messages as the encrypted global result to the client.
8. The client decrypts all messages $\langle encrypt(Tup_1(a)), encrypt(Tup_2(a))\rangle$ with his private keys; he then constructs tuples from the sets $Tup_1(a)$ and $Tup_2(a)$. These tuples form the global result.

Listing 3. Commutative encryption delivery phase

When starting the delivery phase, each datasource first of all generates a new secret key e_i for the commutative encryption function f. With the ideal hash function, it computes the hash values $h(a)$ of all elements of its active domain of the join attribute – that is, all elements $a \in dom_{active}(R_i.A_{join})$. Then, it computes the commutative encryption $f_{e_i}(h(a))$ of each hash value. Next, for all values a each datasource encrypts its set $Tup_i(a)$ with the

appropriate public keys presented in the client's credentials (that is, with our hybrid encryption function $encrypt$). Via the mediator, the messages $\langle f_{e_i}(h(a)), encrypt(Tup_i(a))\rangle$ are exchanged between the datasources.[1] Each datasource commutatively encrypts the hash values again – so that in the end the hash values are encrypted with both keys e_1 and e_2. These messages are returned to the mediator.
We assume that both datasources use the same ideal hash function h and thus identical inputs (from both datasources) yield identical hash values, whereas distinct inputs yield distinct hash values. From the commutativity and bijectivity properties of the commutative encryption function f we conclude that if a value a is included in both active domains, the twofold application of f on $h(a)$ returns identical ciphertexts independent of the order of application of the keys e_1 and e_2. Therefore the mediator can identify the messages that belong to identical values of the join attribute. It combines the corresponding encrypted sets of tuples in a set of result messages of the form $\langle encrypt(Tup_1(a)), encrypt(Tup_2(a))\rangle$ and returns it to the client. The client can now decrypt the tuple sets with his private keys; he then just has to construct tuples from the sets (that is, executing a crossproduct operation on each pair of corresponding tuple sets) and combine them in a result relation. In Listing 3 we present our protocol of the delivery phase with commutative encryption.

5. Efficient Private Matching

We now describe the approach called *private matching* (PM) presented by Freedman et al. in [12]. In the PM model, two parties – the sender and the chooser – each have a set of input values. The chooser is to compute the intersection of these sets (that is, their private matching) without learning values of the sender's input set that are not contained in the intersection. The authors employ homomorphic encryption to ensure confidentiality of these data.

Homomorphic encryption schemes allow for efficiently performing operations like addition on encrypted data. In the following, we denote E an additively homomorphic encryption function. More precisely, E is a semantically secure public key encryption function with the following two properties:

- Given two ciphertexts $E(a)$ and $E(b)$, there is a way to efficiently compute the encrypted sum $E(a+b)$.
- Given a constant γ and a ciphertext $E(a)$, there is a way to efficiently compute $E(\gamma \cdot a)$.

[1] In real life implementations, the mediator should refrain from sending the encrypted tuples to the opposite datasource for performance as well as security reasons. Instead, the mediator could use ID values of fixed length that replace an encrypted tuple set in the messages and is later on used to map the encrypted hash value to the corresponding tuple set. For sake of simplicity, we abstract from that for now.

The elliptic curve variant of ElGamal (see [10]) and the Paillier cryptosystem (see [20]) satisfy these demands; see also [21] for a detailed analysis of homomorphic cryptosystems.
These properties allow for obtaining the encrypted result of an evaluation of a polynomial at an unencrypted point with only the encryptions of its coefficients given. In other words, for a polynomial $P(x) = \sum_{k=0}^{n} c_k x^k$ and an unencrypted input value a such that $b = P(a)$ one can efficiently compute

$$E(b) = E(P(a)) = E(\sum_{k=0}^{n} c_k a^k)$$

– even if only the encryptions $E(c_k)$ of the coefficients are known. Furthermore, for a constant value γ one can efficiently compute $E(\gamma \cdot \sum_{k=0}^{n} c_k a^k + a)$.

Evaluation of an encrypted polynomial is employed in the PM approach in the following way. The chooser – having input set $A = \{a_1, \ldots, a_n\}$ – generates a polynomial:

$$P(x) := (a_1 - x) \cdot (a_2 - x) \cdot \ldots \cdot (a_n - x) = \sum_{k=0}^{n} c_k x^k$$

That is, the roots of P are the chooser's input values: $P(a_i) = 0$ for $i = 1 \ldots n$. Coefficients c_k of P are computed to achieve a sum-representation of the polynomial. The chooser then generates a public homomorphic key and encrypts each coefficient c_k. The encrypted coefficients $E(c_k)$ are sent to the sender.
Let the sender's input set be $A' = \{a'_1, \ldots, a'_m\}$; for $l = 1 \ldots m$, the sender generates a random value r_l and computes (based on the homomorphic properties as described above):

$$E(r_l \cdot P(a'_l) + a'_l) \tag{1}$$

These m values are returned to the chooser who decrypts them with his private key. For the values contained in the intersection $a \in A \cap B$ (that is, a subset of the roots of the polynomial), the decryption step yields a itself, because for those values $E(r_l \cdot P(a) + a) = E(a)$; for values not contained in the intersection, the decryption step yields a random value. Thus, by picking those decrypted values that are also contained in A, the chooser can identify the intersection set.
The sender can also concatenate his a'_l-value with *payload data* (in [12] denoted p_y) and send it to the chooser: Instead of computing Equation (1), the sender computes $E(r_l \cdot P(a'_l) + (a'_l || p_y))$; the chooser can only retrieve p_y if the corresponding a'_l-value is in the intersection.

5.1. Secure Mediation with the PM Model

We now adapt the PM model to the MMM system. First of all, we note that homomorphic encryption is a form of

public key encryption. That is why we decided that the client (of the mediator system) should be the only one to generate a public-private homomorphic key pair. The public homomorphic key is distributed in the MMM system with the client's credentials as described in Section 2; the private key is kept secret by the client.

The datasources each build a polynomial with their input values as the roots; that is

- for all $a_k \in dom_{active}(R_1.A_{join})$ (with $k = 1 \ldots n$), S_1 builds the polynomial

$$P_1(x) := (a_1 - x) \cdot (a_2 - x) \cdot \ldots \cdot (a_n - x) = \sum_{k=0}^{n} c_k x^k$$

  and computes the coefficients c_k

- for all $a'_l \in dom_{active}(R_2.A_{join})$ (with $l = 1 \ldots m$), S_2 builds the polynomial

$$P_2(x) := (a'_1 - x) \cdot (a'_2 - x) \cdot \ldots \cdot (a'_m - x) = \sum_{l=0}^{m} d_l x^l$$

  and computes the coefficients d_l

Using the client's public homomorphic key, the datasources can encrypt their coefficients with encryption scheme E and send them to the mediator; that is, S_1 sends all $E(c_k)$ and S_2 sends all $E(d_l)$.

The mediator sends the encrypted coefficients to the opposite datasource. The datasources evaluate the encrypted polynomial of the opposite datasource for each of their input values: as described above they multiply the polynomial with a fresh random number and add their current input value concatenated with payload data; the payload data are those tuples of the input relation that have the current input value as the value of the join attribute. As in Section 4.1 we denote $Tup_i(a)$ the set of all tuples of relation R_i that have value a in the join attribute. That is,

- for all $a_k \in dom_{active}(R_1.A_{join})$ (with $k = 1 \ldots n$), S_1 computes

$$e_k := E(r_k \cdot P_2(a_k) + (a_k || Tup_1(a_k)))$$

- for all $a'_l \in dom_{active}(R_2.A_{join})$ (with $l = 1 \ldots m$), S_2 computes

$$e'_l := E(r'_l \cdot P_1(a'_l) + (a'_l || Tup_2(a'_l)))$$

The encrypted[2] values e_k and e'_l are sent to the client via the mediator. The client decrypts all e_k to either a random value or a value of the form $(a_k || Tup_1(a_k))$; analogously, the client decrypts all e'_l to either a random value or a value of the form $(a'_l || Tup_2(a'_l))$. The client now identifies those value pairs where $a_k = a'_l$ and – as in Section 4.1 – computes the crossproduct of corresponding tuple sets $Tup_1(a_k)$ and $Tup_2(a'_l)$; the resulting tuples form the global result. Listing 4 shows our protocol with homomorphic encryption.

1. Alteration to preparatory and query phase: we assume that the client has one public key for the homomorphic encryption scheme E; this key is distributed with the client's credentials.
2. Let $P_1(x) = \sum_{k=0}^{n} c_k x^k$ be a polynomial, whose roots are all elements in $dom_{active}(R_1.A_{join})$. S_1 computes coefficients c_k, encrypts them with E – using the client's public key – and sends $E(c_k)$ to the mediator.
3. Let $P_2(x) = \sum_{l=0}^{m} d_l x^l$ be a polynomial, whose roots are all elements in $dom_{active}(R_2.A_{join})$. S_2 computes coefficients d_l, encrypts them with E and sends $E(d_l)$ to the mediator.
4. The mediator forwards the encrypted coefficients to the opposite datasource.
5. For each $a_k \in dom_{active}(R_1.A_{join})$, S_1 generates a new random number r_k and computes $e_k := E(r_k \cdot P_2(a_k) + (a_k || Tup_1(a_k)))$. S_1 returns all e_k-values to the mediator.
6. For each $a'_l \in dom_{active}(R_2.A_{join})$, S_2 generates a new random number r'_l and computes $e'_l := E(r'_l \cdot P_1(a'_l) + (a'_l || Tup_2(a'_l)))$. S_2 returns all e'_l-values to the mediator.
7. The mediator sends the $n + m$ encrypted values to the client.
8. The client decrypts the values with his private key. He then checks for decrypted values of the form $(a_k || Tup_1(a_k))$ and $(a'_l || Tup_2(a'_l))$ where $a_k = a'_l$. The corresponding tuple sets $Tup_1(a_k)$ and $Tup_2(a'_l)$ are then combined into the final result.

Listing 4. Homomorphic encryption delivery phase

[2] As tuple sets can be of large size, we could face length restrictions when using asymmetric encryption. To avoid this, instead of encrypting the tuple sets as payload data in the polynomial, the hybrid encryption approach can be taken further. For each tuple set, the datasources generate a separate session key; the session key and an ID value are encrypted in the polynomial whereas each tuple set is encrypted with its corresponding session key and mapped to the ID value in a table. This table is sent separately to the mediator and forwarded to the client; the client can only decrypt the session keys of those tuple sets that form part of the global result.

6. Analysis and Comparison

In this section, we analyze some security aspects of the three different protocols for the delivery phase and compare them based on some key points. However, we do not give a comprehensive cryptanalytic comparison of the protocols; as for the cryptographic strength of the presented technologies we rely on the security proofs as stated by the authors in the respective articles. We also assume that their cryptographic assumptions (for example random oracle model or large domains) are respected in our protocols.

Our main concern is confidentiality of the transmitted data. First of all, we note again that only those data records are included in the datasources' partial results for which access permissions could be established based on the client's credentials. That is, even if the client receives a superset of the global result (as in the DAS approach), he never receives data he is not allowed to read. In the commutative approach, the client only receives the exact global result; whereas in the PM approach the client receives encrypted values of both partial results but he is only able to decipher and combine those values that form the exact global result. See Table 1 for an overview of the disclosed information.

In all three approaches the partial results are encrypted in such a way that only the client can decrypt them and read the actual data records: in the DAS approach we use our hybrid encryption function ($encrypt$) to encrypt the partial results tuple-wise, in the commutative approach we encrypt sets of tuples with $encrypt$, and in the PM approach tuple sets are included in the result of a polynomial evaluation encrypted with the client's public homomorphic key.
However, although the mediator cannot decrypt the partial results, in some approaches he is able to infer some extra information about the partial results as well as the global result. We look at this in detail for each approach (see also Table 1):

- In the DAS approach, it is crucial to encrypt the index table and let the query translator reside on client side. Otherwise the mediator would know the partition ranges and thus be able to approximate the join attribute value for each tuple. While in our protocol the mediator does not know the partition ranges, he still learns the sizes of the partial results (counted in number of tuples) as they are encrypted tuple-wise. After execution of the server query, the mediator also knows the size of the server query result R^C which is an upper bound of the size of the global result.
  One important point in the DAS approach is how the attribute domains are partitioned and indexed. Small partitions with only a few values are more efficient (less post-processing is necessary) but can leak confidential information (see [15] and [8] for an analysis). This is even worse when the domain of the attribute is small (for example just the two values *yes* and *no*).
- In the commutative approach, the client receives an encrypted hash value from each datasource for each value in the active domain of the join attribute; that is, the mediator learns $|dom_{active}(R_i.A_{join})|$. As he is able to identify how many values are common to both datasources, he also learns $|dom_{active}(R_1.A_{join}) \cap dom_{active}(R_2.A_{join})|$ which is a lower bound of the size of global result.
- In the PM approach, the mediator knows the degree of the polynomial (and thus the number of roots) from the number of encrypted coefficients he receives; thus he knows $|dom_{active}(R_i.A_{join})|$.

Table 1. Extra information disclosed to client and mediator

	Client	Mediator
Database-as-a-Service	superset of global result, index tables	$\|R_i\|$ and $\|R^C\|$
Commutative Encryption	(only exact global result)	$\|dom_{active}(R_i.A_{join})\|$ and size of intersection
Private Matching	$\|dom_{active}(R_i.A_{join})\|$	

We also note that in the commutative approach and in the PM approach the datasources each learn the size of the active domain of the join attribute of the opposite datasource: In the commutative approach the number of encrypted hash values equals the number of distinct values in the active domain while in the PM approach this is the case for the degree of the polynomial.

In addition to credentials and hybrid encryption already used in the MMM system, the following cryptographic primitives have to be applied in the three protocols: in the DAS approach we employ a collision-free hash function to compute index values for the partitions, in the commutative approach we need an ideal hash function (computed by a random oracle) known to both datasources and secret keys for both datasources in order to encrypt hash values commutatively, and in the PM approach we rely on secure evaluations of a homomorphically encrypted polynomial that is masked by different random numbers. See Table 2 for an overview.

Whereas confidentiality of data is guaranteed in all three approaches, from the computational point of view there are some differences:

- In the DAS approach, the client has to interact twice with the mediator: In a first step, he sends the global

Table 2. Applied cryptographic primitives

Database-as-a-Service	hashfunction
Commutative Encryption	hashfunction and commutative encryption
Private Matching	homomorphic encryption and random numbers

query and in a second step he retrieves the index tables and generates the client and the server query. He receives more data records than necessary and has to execute the client query to retrieve the global result. For the datasources, the DAS approach is the most convenient one, as they only have to send data once.

- In the commutative approach, the client receives the exact tuple sets of both datasources that form the global result; he has to decrypt them and combine them to full tuples of the global result. The datasources only do a small extra computation to encrypt their hash values and the hash values of the other datasource; however, they have to interact twice with the mediator.
- In the PM approach, the client retrieves all the tuples of the encrypted partial results. The datasources have to interact twice with the mediator and have to evaluate the encrypted polynomial for all their input values (this is quite expensive, although Freedman et al. show in [12] how the polynomial can be evaluated efficiently).

Based on these performance considerations, the commutative approach seems to be the most efficient one to be employed in a secure mediation system; a prototypical web based system for commutative encryption has thus been implemented at our department.

7. Related Work

Several other approaches cover the topics of secure multi-party computation and querying encrypted databases and seem to be promising for secure mediation. Özsoyoglu et al. (see [19]) for example study different encryption techniques (such as order preserving or multiple encryption) for databases and their respective query transformations. Another approach we have not investigated so far is "encryption with subkeys" by Davida et al. (see [11]).

While in this article we only cover join queries, Yang et al. (see [24]) analyze selection queries over a table in an outsourced database. Unlike the DAS model, they encrypt each attribute value (that is, each table cell) separately. Each encrypted value also has a "checksum" that is necessary for query execution on the encrypted table. Furthermore, the server returns the exact set of encrypted values that satisfy the condition of the query. Therefore the client does not need to post-process the results of the server as in the DAS model. The authors also propose a solution using metadata that enhance the efficiency of query evaluation.

As another form of queries, aggregation queries over encrypted data are treated in [14] and [9]. However, the encryption scheme used by Hacıgümüş et al. in [14] was shown to be insecure by Mykletun et al. in [18]; as an alternative, they propose another approach for processing aggregation queries without using this encryption scheme. Additionally they propose a variant of the DAS model (called "mixed DAS model") where only sensitive attributes are encrypted and the other attributes are not encrypted.

Approaches for secure multi-party computation include for example the work of Kissner et al. (see [16]); they use homomorphic encryption in a multi-party setting to compute the intersection or the union of multisets.
In [22] the authors consider a two-party computation of the scalar product and analyze the level of information inference using information theory; they show that in order to solve the scalar product problem without a third party at least half of the private information must be disclosed.

Finally we mention that database query processing of encrypted data has also been studied in more restricted architectures. For example, Carminati et al. (see [7]) employ the DAS approach to treat XPath expressions in a third-party architecture that only distinguishes between the owner and the publisher of information.

8. Conclusion and Future Work

In this paper, we presented three protocols that extend the basic delivery phase of the Multimedia Mediator with mechanisms to execute join queries over encrypted data records. All three protocols combine the notion of secure mediation with querying encrypted databases and secure multi-party computation. In contrast to pure multi-party computation approaches, responsibilities (like for example, ownership of data) are distributed between the participants. However, we showed that all three considered approaches can be employed in the secure mediation setting.

So far we only considered one join attribute; it would be interesting to analyze whether our three protocols can be easily adapted to work with more than just one join attribute. Moreover, in a mediator hierarchy one mediator can act as a datasource for other mediators. Therefore, the case in which several join queries are executed successively has to be considered. Inclusion of other relational operations is a demanding field of further research as well.

9. Acknowledgments

We thank Frank Müller for valuable discussions.

References

[1] R. Agrawal, A. V. Evfimievski, and R. Srikant. Information sharing across private databases. In A. Y. Halevy, Z. G. Ives, and A. Doan, editors, *ACM SIGMOD International Conference on Management of Data, Proceedings*, pages 86–97. ACM, 2003.

[2] C. Altenschmidt and J. Biskup. Explicit representation of constrained schema mappings for mediated data integration. In S. Bhalla, editor, *Databases in Networked Information Systems, Proceedings*, volume 2544 of *Lecture Notes in Computer Science*, pages 103–132. Springer, 2002.

[3] C. Altenschmidt, J. Biskup, U. Flegel, and Y. Karabulut. Secure mediation: Requirements, design, and architecture. *Journal of Computer Security*, 11(3):365–398, June 2003.

[4] J. Biskup, B. Sprick, and L. Wiese. Secure mediation with mobile code. In S. Jajodia and D. Wijesekera, editors, *Data and Applications Security XIX, Proceedings*, volume 3654 of *Lecture Notes in Computer Science*, pages 267–280. Springer, 2005.

[5] K. S. Candan, S. Jajodia, and V. S. Subrahmanian. Secure mediated databases. In S. Y. W. Su, editor, *12th International Conference on Data Engineering, Proceedings*, pages 28–37. IEEE Computer Society, 1996.

[6] R. Canetti, U. Feige, O. Goldreich, and M. Naor. Adaptively secure multi-party computation. In *Symposium on the Theory of Computing, Proceedings*, pages 639–648. ACM, 1996.

[7] B. Carminati, E. Ferrari, and E. Bertino. Securing XML data in third-party distribution systems. In O. Herzog, H.-J. Schek, N. Fuhr, A. Chowdhury, and W. Teiken, editors, *Conference on Information and Knowledge Management, Proceedings*, pages 99–106. ACM, 2005.

[8] A. Ceselli, E. Damiani, S. D. C. di Vimercati, S. Jajodia, S. Paraboschi, and P. Samarati. Modeling and assessing inference exposure in encrypted databases. *ACM Transactions on Information and System Security*, 8(1):119–152, 2005.

[9] S. S. Chung and G. Özsoyoglu. Anti-tamper databases: Processing aggregate queries over encrypted databases. In R. S. Barga and X. Zhou, editors, *International Conference on Data Engineering – Workshops, Proceedings*, page 98. IEEE Computer Society, 2006.

[10] R. Cramer, R. Gennaro, and B. Schoenmakers. A secure and optimally efficient multi-authority election scheme. In W. Fumy, editor, *EUROCRYPT, Proceedings*, volume 1233 of *Lecture Notes in Computer Science*, pages 103–118. Springer, 1997.

[11] G. I. Davida, D. L. Wells, and J. B. Kam. A database encryption system with subkeys. *ACM Transactions on Database Systems*, 6(2):312–328, 1981.

[12] M. J. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In C. Cachin and J. Camenisch, editors, *Advances in Cryptology – EUROCRYPT, Proceedings*, volume 3027 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2004.

[13] H. Hacıgümüş, B. R. Iyer, C. Li, and S. Mehrotra. Executing SQL over encrypted data in the database-service-provider model. In M. J. Franklin, B. Moon, and A. Ailamaki, editors, *ACM SIGMOD International Conference on Management of Data, Proceedings*, pages 216–227. ACM, 2002.

[14] H. Hacıgümüş, B. R. Iyer, and S. Mehrotra. Efficient execution of aggregation queries over encrypted relational databases. In Y.-J. Lee, J. Li, K.-Y. Whang, and D. Lee, editors, *Database Systems for Advances Applications, Proceedings*, volume 2973 of *Lecture Notes in Computer Science*, pages 125–136. Springer, 2004.

[15] B. Hore, S. Mehrotra, and G. Tsudik. A privacy-preserving index for range queries. In M. A. Nascimento, M. T. Özsu, D. Kossmann, R. J. Miller, J. A. Blakeley, and K. B. Schiefer, editors, *International Conference on Very Large Data Bases, Proceedings*, pages 720–731. Morgan Kaufmann, 2004.

[16] L. Kissner and D. X. Song. Privacy-preserving set operations. In V. Shoup, editor, *Advances in Cryptology – CRYPTO, Proceedings*, volume 3621 of *Lecture Notes in Computer Science*, pages 241–257. Springer, 2005.

[17] D. Liu, K. H. Law, and G. Wiederhold. Chaos: An active security mediation system. In B. Wangler and L. Bergman, editors, *Conference on Advanced Information Systems Engineering, Proceedings*, volume 1789 of *Lecture Notes in Computer Science*, pages 232–246. Springer, 2000.

[18] E. Mykletun and G. Tsudik. Aggregation queries in the database-as-a-service model. In E. Damiani and P. Liu, editors, *Data and Applications Security XX, Proceedings*, volume 4127 of *Lecture Notes in Computer Science*, pages 89–103. Springer, 2006.

[19] G. Özsoyoglu, D. A. Singer, and S. S. Chung. Anti-tamper databases: Querying encrypted databases. In S. D. C. di Vimercati, I. Ray, and I. Ray, editors, *Data and Applications Security XVII, Proceedings*, pages 133–146. Kluwer, 2003.

[20] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In J. Stern, editor, *Advances in Cryptology – EUROCRYPT, Proceedings*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer, 1999.

[21] D. K. Rappe. *Homomorphic Cryptosystems and their Applications*. PhD dissertation, University of Dortmund, Department of Mathematics, Aug. 2004. Cryptology ePrint Archive, Report 2006/001, `http://eprint.iacr.org/2006/001`.

[22] D.-W. Wang, C.-J. Liau, Y.-T. Chiang, and T.-S. Hsu. Information theoretical analysis of two-party secret computation. In E. Damiani and P. Liu, editors, *Data and Applications Security XX, Proceedings*, volume 4127 of *Lecture Notes in Computer Science*, pages 310–317. Springer, 2006.

[23] G. Wiederhold and M. R. Genesereth. The conceptual basis for mediation services. *IEEE Expert Intelligent Systems and their Applications*, 12(5):38–47, Sept./Oct. 1997.

[24] Z. Yang, S. Zhong, and R. N. Wright. Privacy-preserving queries on encrypted data. In D. Gollmann, J. Meier, and A. Sabelfeld, editors, *European Symposium on Research in Computer Security, Proceedings*, volume 4189 of *Lecture Notes in Computer Science*, pages 479–495. Springer, 2006.