Restricted Adaptive Oblivious Transfer

Javier Herranz

IIIA, Institut d'Investigació en Intel·ligència Artificial,
CSIC, Consejo Superior de Investigaciones Científicas
Campus UAB s/n, E-08193 Bellaterra, Spain
e-mail: jherranz@iiia.csic.es

Abstract

In this work we consider the following primitive, that we call *restricted adaptive oblivious transfer*. On the one hand, the owner of a database wants to restrict the access of users to this data according to some policy, in such a way that a user can only obtain information satisfying the restrictions imposed by the owner. On the other hand, a legitimate user wants to privately retrieve allowed parts of the data, in a sequential and adaptive way, without letting the owner know which part of the data is being obtained.

After having formally described the components and required properties of a protocol for restricted adaptive oblivious transfer, we propose two schemes to realize this primitive. The first one is only of theoretical interest at the current time, because it uses a cryptographic tool which has not been realized yet: cryptosystems which are both multiplicatively and additively homomorphic. The second scheme, fully implementable, is based on secret sharing schemes.

Keywords. Oblivious transfer, homomorphic encryption, secret sharing.

1 Introduction

With the growth of the Internet, many practical situations involving operations on digital confidential data appear constantly. On the one hand, the owner of the data wants it to remain private against those users who do not have the right to access it. This can be achieved by keeping the data in a secure device, or by encrypting it. Many examples of such confidential databases can be found in the areas of medical analysis, electronic commerce, banking, or digital business in general. On the other hand, some users can have the right to access to some parts of this confidential data, because of their role (doctors) or because they pay for it. In some situations, these allowed users may want to keep private what part of the data they are retrieving. Imagine the following motivating example inspired by *pay per view* systems: a TV channel over the Internet broadcasts different programs (films, sport events...) which can be watched only by those clients who have paid for them. A client may pay a registration fee which gives him the right of watching five films and ten football matches in the current month, for example. A solution for the TV channel is to keep a private database containing one password for each program, and then to allow the download of the program only to those users having the corresponding password. If a client has paid only to watch films, he should have access only to the part of the database which contains the passwords

for the films. On the other hand, when asking for a password to watch a film, maybe the client does not want the TV channel to know which film he is going to watch.

Summing up, there are situations where both the owner of the data (from now on, the *server*) and the users who have access to some parts of the data (from now on, the *clients*) want to preserve some kind of privacy. The server wants to restrict the access of each client to his data, by means of some policy; these restrictions can be defined by (decreasingly) monotone families, containing the subsets of items that are allowed to be retrieved. The clients can ask for retrieval of different items, in a sequential and adaptive (i.e., possibly depending on the previously retrieved items) way, and should obtain these items, as long as they form a subset of allowed items, without letting the server know which items have been retrieved. We denote this problem as *restricted adaptive oblivious transfer*. The cryptographic primitive of standard oblivious transfer [16, 7] provides an interactive protocol between a server and a client: the client retrieves an item db_i, and nothing else, from a database $DB = \{db_1, \ldots, db_N\}$ of secret items maintained by the server, who does not obtain any information about the index i (chosen by the client) of the retrieved item. Of course, such a protocol does not solve, by itself, our problem of restricted adaptive oblivious transfer: since the server does not know the index of the currently queried item, he cannot decide if the client has (still) the right to obtain this item.

Related work. Ishai and Kushilevitz introduced in [13] a primitive that they called *generalized oblivious transfer* (GOT), which can be seen as a non-sequential (and, thus, non-adaptive) counterpart of the primitive that we consider here, restricted adaptive oblivious transfer. In a GOT protocol, the server defines a monotone family of restrictions, as well; however, the client must decide at the same time the whole subset of items (satisfying the restrictions) in the database he wants to retrieve. The retrieval protocol is executed only once, and the client receives all the queried items together. The solution proposed in [13] uses parallel instances of a standard oblivious transfer protocol (with $N = 2$). This is why the authors used the term 'generalized oblivious transfer' to define this primitive. A different solution to implement GOT has been recently proposed in [18], by combining secret sharing schemes and parallel instances of standard oblivious transfer. However, since the use of parallel instances of standard oblivious transfer is not mandatory to realize this primitive (as we will see in this paper), and since the name 'generalized oblivious transfer' has already been used for other different extensions of the concept of oblivious transfer (see [6, 5]), we have chosen a different name, 'restricted adaptive oblivious transfer', for the considered primitive, and not 'adaptive generalized oblivious transfer', for example.

We want to stress here that the main difference between restricted adaptive oblivious transfer and generalized oblivious transfer, i.e. the fact that queries can be made in a sequential and adaptive way, is of great importance in some applications. For example, in a pay-per-view system, a client may want to see/buy a film of a particular director before deciding if he wants to buy/see other films of this director.

Anyway, the concept of restricted adaptive oblivious transfer is not completely new. Aiello et al. [1] introduced and realized the primitive of *priced oblivious transfer*, which is a particular case of restricted adaptive oblivious transfer: each item of the database has a price, each client has a budget, and the client can privately retrieve/buy items, through the life of the system, as long as his money balance (which is updated by the server at the end of each execution) remains positive. Finally, an even more particular case is that of k-out-of-N oblivious transfer [14, 9], where each client can run the retrieval protocol at most k times, in a sequential and adaptive way.

Our contribution. In this paper we first describe in detail the general primitive of restricted adaptive oblivious transfer: how to model the restrictions of clients, by using (decreasingly) monotone families of subsets; which are the inputs and outputs of each of the protocols; and which are the required security properties for these protocols. Once this is done, we explain how priced oblivious transfer can be recovered as a particular case of this primitive.

Then we present two solutions to the general problem of restricted adaptive oblivious transfer. The first one is inspired by the solution presented in [1] to the problem of priced oblivious transfer. However, differently than in the priced case, the general solution that we propose makes use a very special cryptographic tool: public key encryption schemes which are at the same time additively and multiplicatively homomorphic. Since such schemes are not known to exist yet, the interest of this solution is only theoretical, right now. We propose therefore a second solution, which is fully implementable because it uses well-known (and existing) cryptographic tools; it is inspired by a conceptual relation between restricted adaptive oblivious transfer and secret sharing schemes.

Organization of the paper. In Section 2 we recall the concepts of oblivious transfer, homomorphic encryption, conditional disclosure of a value, and secret sharing schemes. In Section 3 we describe in detail the primitive that we want to implement, restricted adaptive oblivious transfer, and we briefly explain the particular case of priced oblivious transfer. We propose our first, theoretical, solution to the general problem in Section 4, and we propose our second solution, fully implementable, in Section 5. We conclude our work in Section 6.

2 Preliminaries

In this section we recall four cryptographic primitives which will appear in the solutions to the problem of restricted adaptive oblivious transfer that we review or propose.

2.1 1-out-of-N Oblivious Transfer

The problem of 1-out-of-N oblivious transfer can be (informally) defined as follows: a server S stores a database DB containing N items $\{db_i\}_{1 \leq i \leq N}$. By means of an interactive protocol, a client C wants to obtain the i-th item, db_i, for an index i of his choice. The protocol must satisfy a correctness property (the client obtains the required item if both parties behave correctly) and two privacy properties:

- **Privacy for the client.** At the end of the protocol, the server S obtains no information about the index i of the item db_i which has been retrieved by the client.
- **Privacy for the server.** At the end of the protocol, the client C obtains no information about the other items db_j of the database, where $j \neq i$.

1-out-of-N oblivious transfer was introduced in [6, 7] as a generalization of the original concept of oblivious transfer introduced in [16].

2.2 Homomorphic Encryption

A public key encryption scheme $PKE = (\mathcal{KG}, \mathcal{E}, \mathcal{D})$ consists of three probabilistic and polynomial time algorithms. The key generation algorithm $\mathcal{KG}$ takes as input a security parameter (for example, the desired length for the secret key) and outputs a pair (sk, pk) of secret and public keys. The encryption algorithm takes as input a plaintext m and a public key pk, and outputs a ciphertext $c = \mathcal{E}_{pk}(m)$. Finally, the decryption algorithm takes as input a ciphertext and a secret key, and gives a plaintext $m = \mathcal{D}_{sk}(c)$ as output.

Such a scheme has an *homomorphic* property if there exist two operations, defined on the set of ciphertexts and plaintexts, respectively, such that the result of operating two ciphertexts is an encryption of the result of operating the two corresponding plaintexts. For example, a public key cryptosystem is *additively* homomorphic if there exists an operation $\oplus$ defined on the set of ciphertexts, such that the message encrypted in $c_1 \oplus c_2$ is $m_1 + m_2$, where m_i is the message encrypted in c_i, for $i = 1, 2$. Formally, this property is written as

$$\mathcal{D}_{sk}(\mathcal{E}_{pk}(m_1) \oplus \mathcal{E}_{pk}(m_2)) \;=\; m_1 + m_2.$$

Analogously, a cryptosystem is *multiplicatively* homomorphic if there exists an operation $\otimes$ defined on the set of ciphertexts, such that $\mathcal{D}_{sk}(\mathcal{E}_{pk}(m_1) \otimes \mathcal{E}_{pk}(m_1)) = m_1 \cdot m_2$, for any pair of plaintexts (m_1, m_2). Of course, these definitions make sense only if the multiplication and addition operations are properly defined on the set of plaintexts.

ElGamal cryptosystem [11] is the classical example of multiplicatively homomorphic scheme, whereas Paillier's one [15] is an additively homomorphic encryption scheme. Boneh et al. [4] proposed a public key encryption scheme which is, at the same time, additively and multiplicatively homomorphic; but the multiplicatively homomorphic property is satisfied only if the product operation is applied once. Homomorphic cryptosystems have a lot of applications, including electronic auctions and electronic voting.

2.3 Conditional Disclosure of Secrets

The general *conditional disclosure* primitive was introduced in [12]. In this work we are interested in the following particular case: a server holds a secret value s, a message m and a public key pk corresponding to some (additively homomorphic) cryptosystem PKE. A user holds the matching secret key sk, and a message m'. By using the conditional disclosure primitive, the user gives $\mathcal{E}_{pk}(m')$ to the server, and he must obtain secret s from the server if and only if the condition $m' = m$ is satisfied. The server must obtain no information about m'.

This primitive can be realized in the following way: after receiving $\mathcal{E}_{pk}(m')$, the server takes at random an element α and, using the homomorphic property of PKE, computes an encryption c of $\beta = \alpha(m - m') + s$. The server sends c to the user, who can use sk to decrypt and obtain β, which is equal to the secret s if $m = m'$, and is a random value otherwise.

This simple protocol solves the conditional disclosure problem in the case of a single equality between a value known to the server and an encrypted value. The more general case where the condition is a monotone formula with equality leaves can be similarly solved, using the following recursive method, as proposed in [1]. On the one hand, to realize the conditional disclosure of secret s under condition $C_1 \vee C_2$, one can run two independent instances of the protocol, one under condition C_1 and the other under condition C_2. On the other hand, to realize the conditional

disclosure of secret s under condition $C_1 \wedge C_2$, one can choose r at random and run two independent instances of the conditional disclosure protocol, one for secret r under condition C_1, and the other for secret $s + r$ under condition C_2.

2.4 Secret Sharing Schemes

In a *secret sharing scheme*, shares of a secret value are distributed among a set $\mathcal{I} = \{1, \ldots, N\}$ of N players, according to some *access structure* $\Gamma \subset 2^{\mathcal{I}}$, which is increasingly monotone: if $A_1 \in \Gamma$ and $A_1 \subset A_2$, then $A_2 \in \Gamma$, for any $A_1, A_2 \subset \mathcal{I}$. If the scheme is *perfect*, only authorized subsets of players (those in Γ) can recover the secret value from their shares, whereas non-authorized subsets (those out of Γ) do not obtain any information about the secret.

Secret sharing schemes were introduced independently by Shamir [17] and Blakley [3] in 1979. Shamir considered *threshold access structures* $\Gamma = \{A \subset \mathcal{P} : |A| \geq t\}$ defined by a threshold t, and proposed a secret sharing scheme for these structures, based on polynomial interpolation. Other works have proposed schemes realizing more general access structures, such as *vector space secret sharing schemes* [8]. An access structure Γ is realizable by such a scheme, in a finite field $\mathbb{Z}_q$ for some prime q, if there exist a positive integer r and an assignment of vectors $\psi : \mathcal{I} \cup \{D\} \longrightarrow (\mathbb{Z}_q)^r$ (one for each participant) such that $A \in \Gamma$ if and only if $\psi(D)$ is a linear combination of the vectors in $\{\psi(i)\}_{i \in A}$, for any $A \subset \mathcal{I}$. Here D denotes a special entity (real or not), called the *dealer*, outside the set $\mathcal{I}$. If a real dealer D wants to share a secret $s \in \mathbb{Z}_q$ among the players in $\mathcal{I}$, he chooses a random vector $\vec{v} \in (\mathbb{Z}_q)^r$ such that $\vec{v} \cdot \psi(D) = s$. The share of each player $i \in \mathcal{I}$ is $s_i = \vec{v} \cdot \psi(i)$. If $A \in \Gamma$, there exist values $\{\lambda_i^A\}_{i \in A}$ such that $\psi(D) = \sum_{i \in A} \lambda_i^A \psi(i)$. Then, it is easy to see that $s = \sum_{i \in A} \lambda_i^A s_i \bmod q$. Using simple linear algebra, one can also see that subsets out of Γ obtain no information at all about the secret value s. In other words, any possible secret $s' \in \mathbb{Z}_q$ is equally likely from the point of view of a non-authorized subset.

Simmons, Jackson and Martin [19] introduced *linear secret sharing schemes*, which can be seen as a generalization of vector space secret sharing schemes where each player can be assigned more than one vector (and therefore, the length of each share can be larger than the secret). They proved that any access structure can be realized by a linear secret sharing scheme.

3 Restricted Adaptive Oblivious Transfer

In this section we explain in more detail the general functionality of restricted adaptive oblivious transfer (phases, inputs/outputs of the protocols, desired security properties). We then review some particular cases of this functionality, which can be realized quite directly by using existing results.

3.1 Definitions: Functionality and Properties

Let us remember the functionality we want to implement: a server S maintains a secret database $DB = \{db_1, \ldots, db_N\}$ with N entries/items, and a policy defining which subsets of entries of the database can be available, on request, to the different clients. A client wants his requests to be private, i.e. the server should not obtain any information about the item(s) that the client is requesting. The server wants to be sure that a client C will not obtain any information about items

of the database which are not allowed to C. In general, a solution fulfilling this functionality will consist of two protocols:

- **1st protocol: defining rights.** This phase should be run off-line, maybe before the specific values of the entries $\{db_i\}_{1\leq i\leq N}$ of the database are defined. Let $\mathcal{I} = \{1, 2, \ldots, N\}$ denote the set of indices of the items in the database. For a particular client C, the server S specifies the family $\mathcal{B}_C \subset 2^{\mathcal{I}}$ of subsets of items that client C is allowed to obtain. Of course, $\mathcal{B}_C$ must be a decreasingly monotone family: if $B_1 \in \mathcal{B}_C$ is allowed, and $B_2 \subset B_1$, then $B_2 \in \mathcal{B}_C$ is allowed, as well. The family $\mathcal{B}_C$ is known by both S and C. The server S stores an information $\texttt{info}_C$ related to C, which initially contains $\mathcal{B}_C$ and which is updated by S each time C requests an item of the database. Possibly, the client C receives some additional information α_C from S, to be used in the future requests.

- **2nd protocol: request and retrieval.** The input for the client C includes α_C and the index i corresponding to the entry db_i he wants to retrieve from the database. The input for the server consists of the database DB and $\texttt{info}_C$. At the end of the protocol, S must update his information $\texttt{info}_C$, and C obtains a value $\texttt{out}_i$. Assume that this is the t-th time that C executes this protocol with S, and that previous executions had inputs $i_1, \ldots, i_{t-1}$. Let us define the subset of indices $B = \{i_1, \ldots, i_{t-1}, i\}$. Then C obtains the desired value, i.e. $\texttt{out}_i = db_i$, if and only if $B \in \mathcal{B}_C$.

A scheme for this functionality of restricted adaptive oblivious transfer will be considered valid if it satisfies some requirements. Assume that the t-th execution of the protocol has input (i_t, α_C) for C, where $i_t \in \{1, \ldots, N\}$. The first requirement is a typical correctness one: if the client and the server behave honestly during the t-th execution and if $\{i_1, \ldots, i_t\} \in \mathcal{B}_C$, then $\texttt{out}_{i_t} = db_{i_t}$ is the secret output of C. Additionally, two privacy properties are required.

- **Privacy for the client.** In any execution of the protocol for request and retrieval of an item, the server S does not obtain any information about the index i in the input of the client.

- **Privacy for the server.** In the t-th execution of the 'request and retrieval' protocol, with input (i_t, α_C), the client C

  - does not obtain any information about items db_ℓ, for $\ell \neq i_t$; and
  - does not obtain any information about item db_{i_t}, if $\{i_1, \ldots, i_t\} \notin \mathcal{B}_C$.

Of course, a malicious server can always refuse to execute his part of the protocol. There is nothing we can do about it. However, there are quite standard ways to avoid other possible misbehaviours of the server. For example, a dishonest server could arbitrarily modify his database, and run the request and retrieval protocol with entries db'_j different than the correct ones, db_j. In this way, clients would obtain invalid items as answers to their queries. To avoid this problem, we can use a trusted third party (TTP) which engages a *commitment* protocol with the server: the server sends to the TTP the correct items db_j; the TTP chooses random values r_j and computes $h_j = H(db_j, r_j)$, for all $j = 1, \ldots, N$, where H is some hash function. The pairs (r_j, h_j) are published by the TTP. Later, when a client retrieves some item db_i, he can verify that the server has behaved honestly, by checking if $H(db_i, r_i) = h_i$. In the rest of the paper, for simplicity, we will assume that the server behaves honestly, i.e. he does not refuse any query of a client, and he considers as inputs the correct values db_j of the items.

3.2 A Particular Case: Priced Oblivious Transfer

When the family $\mathcal{B}_C \subset 2^{\mathcal{I}}$ is a weighted threshold family, which means that there exist a threshold β and an assignment of positive weights $\omega : \mathcal{I} \to \mathbb{Z}^+$ such that

$$\mathcal{B}_C = \{B \subset \mathcal{I} \text{ s.t. } \sum_{i \in B} \omega(i) \leq \beta\},$$

then we recover *priced oblivious transfer* [1], where elements of the database are supposed to be objects (or goods) with a price, and users have different budgets to privately buy goods. In the solution given in [1], the prices of the objects are the same for all the users/buyers. But their solution can be applied in general to the restricted adaptive oblivious transfer problem where each user can have different restrictions (not only the budget), as long as all the restriction families $\mathcal{B}_C$ are weighted threshold.

Namely, assume that the restrictions of a client are described by a weighted threshold family $\mathcal{B}_C$, which is realized by a threshold β and an assignment of weights $\omega : \mathcal{I} \to \mathbb{Z}^+$. All this information is known by both the client and the server. We present a brief sketch of how the basic solution proposed in [1] would work.

- All the weights are assumed to be different (if necessary, by scaling them and the threshold with a large enough factor): $\omega(i) \neq \omega(j)$, if $i \neq j$.
- The client generates a pair of secret/public keys (sk, pk) for an additively homomorphic cryptosystem; encryption is denoted as $c = \mathcal{E}_{pk}(m)$.
- The server encrypts $c = \mathcal{E}_{pk}(\beta)$.
- If the client wants to retrieve element db_i, he sends $\tilde{c} = \mathcal{E}_{pk}(\omega(i))$ to the server, along with a (cryptographic) proof that the value encrypted in c is greater or equal than the value encrypted in $\tilde{c}$. The client adds a standard oblivious transfer query for the index i.
- The server verifies that the validity proof is correct. If so, he answers the standard oblivious transfer query, and updates $c = c \ominus \tilde{c} = \mathcal{E}_{pk}(\beta - \omega(i))$, for possible future executions of the protocol with the same client.
- The client recovers db_i using the last step of the oblivious transfer protocol, and updates $\beta = \beta - \omega(i)$ and $c = c \ominus \tilde{c}$.

This solution forces the server to store a lot of different information (the threshold β, the assignment of weights ω, the ciphertext c) for each client. This drawback, as well as some of the techniques used in the priced oblivious transfer protocols of [1], will appear in the generic solution to the problem of restricted adaptive oblivious transfer that we present in next section.

4 A First Generic Solution (of Theoretical Interest)

The particular case of priced oblivious transfer does not cover all the possible cases of restricted adaptive oblivious transfer. For example, suppose that a database $DB = \{db_1, db_2, db_3, db_4\}$ contains four items, and that a client C is allowed to retrieve items in one of the subsets of indices

defined by the family $\mathcal{B}_C = \{\{1,2\},\{2,3\},\{3,4\}\}$. It is easy to see that this family cannot be weighted threshold (actually, this is an example of a non weighted threshold family proposed in [2]). Therefore, the ideas of priced oblivious transfer cannot be used here. This case cannot be solved, either, by using predicate encryption techniques.

We propose in this section a first generic solution which works for any possible family $\mathcal{B}_C$ of restrictions. It can be seen as a generalization of the protocols in [1] for priced oblivious transfer. Unfortunately, the solution is, at the current time, of theoretical interest only, because it employs a cryptographic tool which has not been realized yet: cryptosystems which are both multiplicatively and additively homomorphic. Let us assume, anyway, the existence of such a cryptosystem, $PKE = (\mathcal{KG}, \mathcal{E}, \mathcal{D})$, with operations $\otimes$ and $\oplus$ defined on the set of ciphertexts, satisfying for any two plaintexts m_1, m_2:

$$\begin{aligned} \mathcal{D}_{sk}(\mathcal{E}_{pk}(m_1) \otimes \mathcal{E}_{pk}(m_2)) &= m_1 \cdot m_2, \text{ and} \\ \mathcal{D}_{sk}(\mathcal{E}_{pk}(m_1) \oplus \mathcal{E}_{pk}(m_2)) &= m_1 + m_2. \end{aligned}$$

The two protocols of the generic solution that we propose work as follows.

1st protocol: defining rights. Let $\mathcal{B}_C = \{B_1, B_2, \ldots, B_s\} \subset 2^{\mathcal{I}}$ be the family of subsets of indices expressing the collections of items that client C is allowed to query, where $\mathcal{I} = \{1, 2, \ldots, N\}$. In general, we will have $B_j = \{i_{j,1}, i_{j,2}, \ldots, i_{j,n_j}\} \subset \mathcal{I}$. We represent each of these subsets B_j with its *incidence vector*, a length N binary vector $\vec{b}_j = (b_1^{(j)}, b_2^{(j)}, \ldots, b_N^{(j)}) \in \{0,1\}^N$, such that $b_i^{(j)} = 1$ if and only if $i \in B_j$, for $i = 1, 2, \ldots, N$. For simplicity of notation, we will sometimes skip the super-index $^{(j)}$, when it is clear from the context.

The client generates a pair of keys $(pk, sk) \leftarrow \mathcal{KG}(1^k)$ for the homomorphic cryptosystem PKE, and publishes pk. The server computes an encryption of $\mathcal{B}_C$, by encrypting all the vectors $\vec{b}_j$ coordinate-wise. In other words, he computes $\vec{c}_j = (\mathcal{E}_{pk}(b_1^{(j)}), \ldots, \mathcal{E}_{pk}(b_N^{(j)}))$, for $j = 1, \ldots, s$. He also chooses a value u_0. Initially, he sets $\texttt{info}_C = (pk, u_0, \{\vec{c}_j\}_{j=1,\ldots,s})$.

2nd protocol: request and retrieval. If the client wants to retrieve the item db_i, then he must encrypt the vector $\vec{e} = (e_1, \ldots, e_N) \in \{0,1\}^N$, which has a 1 in the i-th position, $e_i = 1$, and a 0 everywhere else, $e_\ell = 0$, $\forall \ell \neq i$. The result is a vector of ciphertexts $\vec{c} = (c_1, c_2, \ldots, c_N) = (\ \mathcal{E}_{pk}(e_1), \ldots, \mathcal{E}_{pk}(e_N)\ )$. If the client has behaved honestly, then the value

$$\tilde{c} = \bigotimes_{\ell=1}^{N} c_\ell^{db_\ell}$$

is an encryption of db_i, due to the (multiplicatively) homomorphic properties of the scheme PKE. Now the server has to update the access family $\mathcal{B}_C$, because the i-th item has been already queried and will be released. For each subset $B_j \in \mathcal{B}_C$, the server updates its representation vector $\vec{b}_j$, by replacing a 1 with a 0 in the i-th position, if $b_i^{(j)} = 1$ (which means that index i belonged to B_j) and by setting all the vector $\vec{b}_j$ to 0, if $b_i^{(j)} = 0$ (which means that the client has asked for an item which was not in B_j, and so the subset B_j must not be considered any more). This is done via the formula

$$\vec{b'}_j = [\vec{b}_j \cdot \vec{e}](\vec{b}_j\ - \vec{e}),$$

where $\vec{b'}_j$ denotes the updated version of the vector associated to subset B_j. The server must perform this operation over encrypted data, $\vec{c}'_j = [\vec{c}_j \odot \vec{c}](\vec{c}_j \ - \vec{c})$; this is the point where we need PKE to be both multiplicatively and additively homomorphic. Furthermore, we will have to perform two levels of products of ciphertexts, one for the 'scalar' product $\vec{c}_j \odot \vec{c}$, and another one to multiply the resulting ciphertext with each component of the vector of ciphertexts $(\vec{c}_j \ - \vec{c})$. For this reason, the scheme in [4], which allows only one level of products of ciphertexts, cannot be used here.

Of course, we have to consider the possibility of dishonest clients. The server will release to the client this value $\tilde{c}$ (an encryption of db_i), conditioned to the fact that the query is *correct*, which means that:

1. the client has behaved honestly in the previous executions of the protocol; and
2. the vector of ciphertexts $\vec{c} = (c_1, \ldots, c_N)$ contains exactly one encryption of 1, and $N-1$ encryptions of 0; and
3. the index i such that c_i is an encryption of 1 still belongs to some subset $B_j \in \mathcal{B}_C$.

The first condition is ensured with a chaining technique [1]: in the t-th execution of the protocol, the client receives an encryption of some value u_t, if and only if the t-th query is correct. Then, in the $(t+1)$-th execution, the client must provide an encryption of u_t.

The second condition ensures that the client will obtain at most one item, if any, and that the client cannot try to "increase" his rights $\mathcal{B}_C$ for the following interaction, by defining $e_\ell = -1$ for some $\ell \in \{1, \ldots, N\}$, for example. The server will not know which entry of $\vec{c}$ encrypts the value 1, but he will be convinced that $\vec{c}$ is well formed if the corresponding vector $\vec{e} = (e_1, \ldots, e_N)$ of plaintexts satisfies

$$\bigvee_{i=1}^{N} (e_i = 1 \ \wedge \ e_\ell = 0, \ \forall \ell \neq i).$$

Finally, the third condition is equivalent to verify that the value encrypted in the scalar product $\vec{b}_j \cdot \vec{c}$ is 1, for some $j \in \{1, \ldots, s\}$. In effect, assuming that $\vec{e}$ is well formed, the scalar product of the plaintext vectors, $\vec{b}_j \cdot \vec{e}$, will be 1 if and only if $i \in B_j$.

Putting all these pieces together, the t-th execution of this 'request and retrieval' protocol works as follows, for $t \geq 1$.

1. To retrieve item db_i, the client computes the vector $\vec{e} = (e_1, \ldots, e_N)$ such that $e_i = 1$ and $e_\ell = 0$, $\forall \ell \neq i$, and encrypts it coordinate-wise, to obtain $\vec{c} = (c_1, \ldots, c_N)$, where $c_\ell = \mathcal{E}_{pk}(e_\ell)$, for $\ell = 1, \ldots, N$.
2. The client sends $\vec{c}$ and $\mathcal{E}_{pk}(u)$ to the server, where $u = u_{t-1}$.
3. The server chooses at random u_t, and encrypts it. He also computes the value $\tilde{c} = \bigotimes_{\ell=1}^{N} c_\ell^{db_\ell}$.
4. By using the protocols/ideas explained in Section 2.3, the server performs a conditioned disclosure of the pair $(\mathcal{E}_{pk}(u_t), \tilde{c})$, under the condition

$$\left(\bigvee_{i=1}^{N} \left[e_i = 1 \ \wedge \ \bigwedge_{\ell=1, \ell \neq i}^{N} e_\ell = 0 \right] \right) \ \bigwedge \ \left(\bigvee_{j=1}^{s} \vec{b}_j \cdot \vec{e} = 1 \right) \ \bigwedge \ (u = u_{t-1}).$$

5. The server updates the value of the encryptions of the vectors $\vec{b}_j$, for $j = 1, \ldots, s$. Remember that the correct update is $\vec{b}_j := [\vec{b}_j \cdot \vec{e}](\vec{b}_j \ -\vec{e})$. Therefore, the server updates the ciphertexts

$$\vec{c}_j := [\vec{c}_j \odot \vec{c}](\vec{c}_j \ -\vec{c}),$$

for $j = 1, \ldots, s$, where $(c_1, \ldots, c_N) \odot (c'_1, \ldots, c'_N) = (c_1 \otimes c'_1) \oplus \ldots \oplus (c_N \otimes c'_N)$. He replaces the old values of $\vec{c}_j$ with the new ones, in $\texttt{info}_C$, where he also replaces u_{t-1} with u_t.

6. If the client has always behaved honestly, he can recover the pair $(\mathcal{E}_{pk}(u_t), \tilde{c})$ and decrypt both ciphertexts with sk, obtaining in this way u_t and the desired item db_i.

Because of the security properties of the encryption scheme and of the protocol for conditional disclosure of secrets, it is easy to see that this protocol enjoys the required privacy properties for restricted adaptive oblivious transfer. The server cannot obtain any information about the index i, and the client obtains item db_i only if he is allowed to do it.

5 A Second Generic Solution

In this section we propose a second solution to the general problem of restricted adaptive oblivious transfer. Although it is not very efficient (due to the amount of computational and communication effort which is required in each execution of the protocol), it is fully implementable in practice, since it uses existing cryptographic tools: additively homomorphic cryptosystems, standard oblivious transfer, and secret sharing schemes.

It can seem quite natural to relate the concept of restricted adaptive oblivious transfer (where a decreasingly monotone family $\mathcal{B}_C$ appears) and the concept of secret sharing (where an increasingly monotone family Γ appears). Actually, the proposal in [18] for the non-sequential case of 'generalized oblivious transfer' already uses secret sharing as an ingredient. Let us sketch the basic ideas of our second solution. Given a family $\mathcal{B}_C \subset 2^{\mathcal{I}}$ defining the restrictions of a client C, the server considers the increasingly monotone family

$$\Gamma = (\mathcal{B}_C)^c = \{A \subset \mathcal{I} \mid A \notin \mathcal{B}_C\}.$$

For simplicity, we assume that Γ admits a vector space secret sharing scheme, defined by $\psi : \mathcal{I} \cup \{D\} \to (\mathbb{Z}_q)^r$. The server can choose a vector $\vec{v} \in (\mathbb{Z}_q)^r$ at random, such that all the shares $s_i = \vec{v} \cdot \psi(i)$ are different. The real secret would be $s = \vec{v} \cdot \psi(D)$. Now, if we consider a different random value $s' \neq s$, the key point is that the system of equations $\Big\{\vec{x} \cdot \psi(D) = s',\ \{\vec{x} \cdot \psi(j) = s_j\}_{j \in B}\Big\}$, with vector of unknowns $\vec{x}$, will have solution if and only if $B \notin \Gamma$ (otherwise, if $B \in \Gamma$, the real shares of elements in B would completely determine the secret to be s, different from s'). In other words, the system has solution if and only if $B \in \mathcal{B}_C$.

During each execution of the 'request and retrieval' protocol, the client C should prove that he knows a solution $\vec{x}$ for such a system of equations, being B the subset of all the indices that C has queried to the database (including the current one). If this subset B is not allowed, then it will be in Γ and so C will not be able to provide such a solution to the system of equations. Of course, care must be taken in order to preserve the privacy of the queries, and also to prevent a possible dishonest behaviour of a client who tries to cheat the server. The details of the protocols which compose our second solution are as follows.

1st protocol: defining rights. Let $\mathcal{B}_C \subset 2^{\mathcal{I}}$ be the family containing the allowed subsets of items for client C, where $\mathcal{I} = \{1, 2, \ldots, N\}$. The server considers the increasingly monotone family (or access structure) $\Gamma = (\mathcal{B}_C)^c = \{A \subset \mathcal{I} \mid A \notin \mathcal{B}_C\}$, which admits (for simplicity) a vector space access secret sharing scheme, defined by $\psi : \mathcal{I} \cup \{D\} \to (\mathbb{Z}_q)^r$. Without loss of generality (for example by multiplying the vectors with the appropriate scalar values), we can assume that all the vectors $\psi(i)$ are different.

1. The client generates a pair of keys $(pk, sk) \leftarrow \mathcal{KG}(1^k)$ for an additively homomorphic cryptosystem $PKE = (\mathcal{KG}, \mathcal{E}, \mathcal{D})$, and publishes pk.
2. The server chooses at random a vector $\vec{v} \in (\mathbb{Z}_q)^r$ such that all the shares $s_i = \vec{v} \cdot \psi(i)$, for $i = 1, \ldots, N$, are different. This step can be done, if necessary, at the same time as the assignment of vectors ψ realizing Γ is chosen.
3. The server chooses at random a value s' different from the value $s = \vec{v} \cdot \psi(D)$.
4. The server makes public ψ, $\vec{v}$ and s'. He defines $\texttt{info}_C = (\psi, \vec{v}, s', pk)$.

2nd protocol: request and retrieval. In the t-th execution of this protocol, the client C wants to retrieve the item db_{i_t}. Assume that the subset of indices that C has queried (including this i_t) are allowed, i.e. $B = \{i_1, \ldots, i_t\} \in \mathcal{B}_C$.

1. The client solves the system of equations $\left\{\vec{x} \cdot \psi(D) = s',\ \{\vec{x} \cdot \psi(i_j) = s_{i_j}\}_{j=1,\ldots,t}\right\}$, obtaining a solution $\vec{x} = (x_1, \ldots, x_r) \in (\mathbb{Z}_q)^r$.
2. The client encrypts vector $\vec{x}$ coordinate-wise, to obtain $\vec{c} = (c_1, \ldots, c_r) = (\mathcal{E}_{pk}(x_1), \ldots, \mathcal{E}_{pk}(x_r))$. He also encrypts the share $\tilde{s}_{i_t} = s_{i_t}$, to obtain $\tilde{c}_{i_t} = \mathcal{E}_{pk}(\tilde{s}_{i_t})$.
3. The client sends $\vec{c}$, $\tilde{c}_{i_t}$ and a standard oblivious transfer (or private information retrieval, PIR) query for the index i_t to the server.
4. Using the homomorphic properties of PKE and the value $\tilde{c}_{i_t} = \mathcal{E}_{pk}(\tilde{s}_{i_t})$ received from the client, the server transforms the original database DB into $\widetilde{DB} = \{\widetilde{db}_j\}_{j=1,\ldots,N}$, being

$$\widetilde{db}_j = \mathcal{E}_{pk}(\gamma_j(s_j - \tilde{s}_{i_t}) + db_j),$$

   where γ_j are random elements. If the query is correct, i.e. $\tilde{s}_{i_t} = s_{i_t}$, then we have that $\widetilde{db}_{i_t} = \mathcal{E}_{pk}(db_{i_t})$, whereas $\widetilde{db}_j$ is an encryption of a totally random element, for $j \neq i_t$.
5. The server considers the oblivious transfer (or PIR) query from the client, over the transformed database $\widetilde{DB}$, and performs a conditioned disclosure (see Section 2.3) of the corresponding answer, under the condition

$$\bigwedge_{j=1}^{t} \left[\bigvee_{i=1}^{N} \left(\vec{x} \cdot \psi(i) = s_i \ \wedge\ \tilde{s}_{i_j} = s_i\right) \right] \bigwedge \left(\vec{x} \cdot \psi(D) = s'\right).$$

6. The server adds $\tilde{c}_{i_t} = \mathcal{E}_{pk}(\tilde{s}_{i_t})$ to $\texttt{info}_C$.

7. If the client has behaved honestly, he obtains the correct answer $\widetilde{db}_{i_t}$ to the oblivious transfer (or PIR) query, he decrypts it by using sk and recovers the desired item db_i.

Again, as it happened in our first (theoretical) solution, the security of this second protocol for restricted adaptive oblivious transfer directly infers from the security of the employed cryptographic primitives. On the one hand, the query of the client does not reveal anything about the index i_t to the server, because all the information is encrypted, and because of the privacy properties of the standard oblivious transfer (or PIR) protocol that has been used. On the other hand, due to the properties of the employed secret sharing scheme and protocol for conditional disclosure of secrets, the client can obtain only subsets of items of the database that are allowed to him. The formal proofs for the security of this proposal, which are done by reduction to the security of the employed primitives, are omitted due to their simplicity.

5.1 Some Remarks

- Note the importance of the fact that all the shares s_i must be different. If this is not the case, and $s_i = s_j$ for some $i \neq j$, then the client can send $\mathcal{E}_{pk}(s_i)$ along with an encryption of a correct solution $\vec{x}$ (if retrieving db_i is allowed at the current execution), whereas the attached oblivious transfer (or PIR) query corresponds to the (possibly not allowed) index j. The dishonest client would finally obtain the (possibly forbidden) item db_j.

- If the client knows *a priori* all the allowed queries that he is going to make, he can solve the whole system of equations, obtain a solution $\vec{x}$ and use this same vector for all the executions. If the server is prevented of this fact, he can use a chaining technique, such as the one described in the theoretical solution of Section 4 (using random elements $u_0, u_1, \ldots$ for the chaining), to ensure that the client behaves correctly in each execution. In the first execution, the condition for the disclosure would be

$$\bigvee_{i=1}^{N} (\vec{x} \cdot \psi(i) = s_i \ \wedge \ \tilde{s}_{i_1} = s_i) \bigwedge (\vec{x} \cdot \psi(D) = s') \bigwedge (u = u_0).$$

  Whereas, in the t-th execution, for $t > 1$, the condition for the disclosure would be

$$\bigvee_{i=1}^{N} (\vec{x} \cdot \psi(i) = s_i \ \wedge \ \tilde{s}_{i_1} = s_i) \bigwedge (u = u_{t-1}).$$

  .

- Private Information Retrieval (PIR, for short), introduced in [10], is a very similar concept to that of 1-out-of-N oblivious transfer (recall Section 2.1), with the difference that the 'privacy for the server' requirement is removed. In other words, the client may eventually obtain information about other entries db_j different from the one he is querying for. In our protocol for restricted adaptive oblivious transfer, a standard PIR query can be used instead of oblivious transfer, because the other entries $\widetilde{db}_j$ that the client could obtain, do not give any information at all about db_j to the client, provided all the shares s_i are different.

- For simplicity, we have considered that the access structure $\Gamma = (\mathcal{B}_C)^c$ admits a vector space secret sharing scheme. If this is not the case, we know anyway that Γ admits a linear secret

sharing scheme, which can be seen as a generalized vector space one, where each player i can be associated to more than one vector: $\psi(i)^{(1)}, \ldots, \psi(i)^{(\ell_i)} \in (\mathbb{Z}_q)^r$, and so he will receive more than one share: $s_i^{(1)}, \ldots, s_i^{(\ell_i)} \in \mathbb{Z}_q$. Our protocol can be easily extended to this case. The only precaution to be taken is to ensure that all the indices i are associated to the same amount of vectors, so that the query of the client, which includes encryptions of the shares associated to the desired index, does not leak any information about this index. This can be achieved by adding, if necessary, "dummy" vectors to some indices; the dummy vectors added to an index i must be linear combinations of the real vectors already associated to i.

6 Conclusions

We have presented in this work the first solutions to the general problem of restricted adaptive oblivious transfer. The first one is currently of theoretical interest only; it cannot be implemented in practice, because it uses public key cryptosystems which are at the same time additively and multiplicatively homomorphic. Instances of such cryptosystems have not been found yet. Our second solution is fully implementable, and makes use of a conceptual relation between restricted adaptive oblivious transfer and secret sharing schemes.

As future research related to this work, we can mention two possibilities. The first one would be to improve the efficiency of our solutions, regarding the required efforts for computations and communications. The second one would be to design a public key cryptosystem with both additive and multiplicative homomorphic properties, which would make our first solution fully implementable, as well.

References

[1] B. Aiello, Y. Ishai and O. Reingold. Priced oblivious transfer: how to sell digital goods. *Proceedings of Eurocrypt'01*, LNCS **2045**, Springer-Verlag, pp. 119–135 (2001).

[2] J. Benaloh and J. Leichter. Generalized secret sharing and monotone functions. *Proceedings of Crypto'88*, LNCS **403**, Springer, pp. 27–35 (1990).

[3] G.R. Blakley. Safeguarding cryptographic keys. *Proceedings of the National Computer Conference, American Federation of Information*, Processing Societies Proceedings **48**, pp. 313–317 (1979).

[4] D. Boneh, E-J. Goh and K. Nissim. Evaluating 2-DNF formulas on ciphertexts. *Proceedings of TCC'05*, LNCS **3378**, Springer-Verlag, pp. 325–341 (2005).

[5] G. Brassard and C. Crépeau. Oblivious transfers and privacy amplification. *Proceedings of Eurocrypt'97*, LNCS **1233**, Springer-Verlag, pp. 334–347 (1997).

[6] G. Brassard, C. Crépeau and J.M. Robert. Information theoretic reductions among disclosure problems. *Proceedings of FOCS'86*, IEEE Computer Society, pp. 168–173 (1986).

[7] G. Brassard, C. Crépeau and J.M. Robert. All-or-nothing disclosure of secrets. *Proceedings of Crypto'86*, LNCS **263**, Springer, pp. 234–238 (1987).

[8] E.F. Brickell. Some ideal secret sharing schemes. *Journal of Combinatorial Mathematics and Combinatorial Computing*, **9**, pp. 105–113 (1989).

[9] J. Camenisch, G. Neven and A. Shelat. Simulatable adaptive oblivious transfer. *Proceedings of Eurocrypt'07*, LNCS **4515**, Springer-Verlag, pp. 573–590 (2007).

[10] B. Chor, O. Goldreich, E. Kushilevitz and M. Sudan. Private information retrieval. *Proceedings of FOCS'95*, pp. 41–51 (1995).

[11] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Information Theory*, **31**, pp. 469–472 (1985).

[12] Y. Gertner, Y. Ishai, E. Kushilevitz and T. Malkin. Protecting data privacy in private information retrieval schemes. *Journal of Computer and System Sciences*, **60** (3), pp. 592–629 (2000).

[13] Y. Ishai and E. Kushilevitz. Private simultaneous messages protocols with applications. *Proceedings of ISTCS'97*, IEEE Computer Society, pp. 174–184 (1997).

[14] M. Naor and B. Pinkas. Oblivious transfer with adaptive queries. *Proceedings of Crypto'99*, LNCS **1666**, Springer-Verlag, pp. 573–590 (1999).

[15] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. *Proceedings of Eurocrypt'99*, LNCS **1592**, Springer-Verlag, pp. 223–238 (1999).

[16] M. Rabin. How to exchange secrets by oblivious transfer. Technical report TR-81, Harvard Aiken Computation Laboratory (1981).

[17] A. Shamir. How to share a secret. *Communications of the ACM*, vol. **22**, pp. 612–613 (1979).

[18] B. Shankar, K. Srinathan and C. Pandu Rangan. Alternative protocols for generalized oblivious transfer. *Proceedings of ICDCN'08*, LNCS **4904**, Springer-Verlag, pp. 304–309 (2008).

[19] G.J. Simmons, W. Jackson and K. Martin. The geometry of secret sharing schemes. *Bulletin of the ICA* **1**, pp. 71–88 (1991).