On the Practical Feasibility of Secure Distributed Computing: a Case Study

Gregory Neven, Frank Piessens*, Bart De Decker
Dept. of Computer Science, K.U.Leuven
Celestijnenlaan 200A, B-3001 Heverlee, Belgium
Fax: ++32(0)16 327996
E-mail:{gneven,frank,bart}@cs.kuleuven.ac.be

Abstract

Secure Distributed Computing addresses the problem of performing a computation with a number of mutually distrustful participants, in such a way that each of the participants has only limited access to the information needed for doing the computation. Over the past decade, a number of solutions for this problem have been developed. The various proposed solutions differ in the cryptographic primitives that are used, and in the class of computations that can be performed. However, all sufficiently general solutions have one thing in common: the communication overhead between the involved parties seems to be prohibitive.

In this paper, we consider a concrete instance (with considerable practical interest) of the general problem of secure distributed computing, and we investigate how bad the communication overhead really is. This involves tailoring the different general solutions to the specific problem at hand, optimizing them for minimal communication overhead, and evaluating the resulting communication overhead.

Our conclusion is that the communication overhead is still large, but can be substantially reduced using specific techniques. Moreover, agent-techology makes Secure Distributed Computing more appealing, since the mobile agents can position themselves close to each other.

*Postdoctoral Fellow of the Belgian National Fund for Scientific Research (F.W.O.)

1 Introduction

Secure distributed computing addresses the problem of distributed computing where some of the algorithms and data that are used in the computation must remain private. Usually, the problem is stated as follows, emphasizing privacy of data. Let f be a publicly known function taking n inputs, and suppose there are n parties (named $p_i, i = 1 \dots n$), each holding one private input x_i. The n parties want to compute the value $f(x_1, \dots, x_n)$ without leaking any information about their private inputs (except of course the information about x_i that is implicitly present in the function result) to the other parties. An example is voting: the function f is addition, and the private inputs represent yes ($x_i = 1$) or no ($x_i = 0$) votes. In case you want to keep an algorithm private, instead of just data, you can make f an interpreter for some (simple) programming language, and you let one of the x_i be an encoding of a program.

In descriptions of solutions to the secure distributed computing problem, the function f is usually encoded as a boolean circuit, and therefore secure distributed computing is also often referred to as *secure circuit evaluation*.

It is easy to see that an efficient solution to the secure distributed computing problem would be an enabling technology for a large number of interesting distributed applications across the Internet. Some example applications are: auctions without a trusted third party ([9]), charging for the use of algorithms on the basis of a usage count ([10, 11]), various kinds of weighted voting, protecting mobile code integrity and privacy ([8]), ...

Over the past decade, a fairly large variety of solutions to the problem has been proposed. An overview is given by Franklin [4]. These solutions differ from each other in the cryptographic primitives that are used, and in the class of computations that can be performed (some of the solutions only allow for specific kinds of functions to be computed). However, these ingenious, powerful solutions have not yet been used in practical distributed applications, mainly because they incur a very large communication overhead.

Given the fact that communication bandwidth increases steadily, and research on more efficient solutions to the secure distributed computing problem slowly decreases the communication overhead, it is clear that practical feasibility of secure distributed computing protocols is only a matter of time. Moreover, agent-technology makes secure distributed computing more appealing, since the mobile agents can position themselves close to each other, thus avoiding large transfers across the Internet. Of course, the site on which the agent is executing must be trusted by at least one of the parties. A host that is unconditionally trusted by all parties obviously reduces secure distributed computing to a trivial problem requiring hardly any protocol at all. However, the level of trust needed here exceeds the one we need by far. In contrast with protocols that use unconditionally trusted third parties, the trusted site is not involved directly. It simply offers a secure execution platform: i.e. it executes the agents correctly, does not spy on them and does not leak information to other agents. The trusted host does *not* have to know the protocol used between the agents. In other words, the combination of agent technology and secure distributed computing protocols makes it possible to use a *generic* trusted third party that, by offering a secure execution platform, can act as trusted third party for a wide variety of protocols in a uniform way.

The goal of this paper is to investigate how close we are to practical applicability of secure distributed computing technology. We investigate this in the following way. In section 2, we describe a concrete problem of practical interest that requires this technology. Next, we try to solve the problem in the most efficient way, using various techniques for secure distributed computing proposed in the literature. For each of the investigated techniques, we tailor and optimize the technique for the problem at hand. Finally, we conclude by assessing the acceptability of the communication overhead.

Our conclusions show that practical feasibility of secure distributed computing technology is for the very near future, if not for today.

2 An example application of secure distributed computing technology

In this section, an interesting application that requires the use of secure distributed computing technology is introduced, and a number of assumptions about this application (like estimations of the size of the inputs to the application) are made explicit. This application could be implemented based on conventional cryptography and a trusted third party, but if it must be implemented without trusted third party, the use of secure distributed computing techniques is essential.

Suppose Alice is human resource manager of a company looking for new employees and Bob makes a living gathering CVs in a database and selling these to interested companies. Bob's database contains a wide variety of records but Alice is not willing to pay for a CV she's not interested in: she only wants records that satisfy a certain query. Of course, Bob doesn't want to reveal his entire database if Alice is only going to buy a few records from it, but on the other hand Alice might want to keep her query secret from Bob (maybe she doesn't want her competitors to know about her selection criteria). What they need is a so-called Secret Query Database, that works as follows.

Let's denote Alice's query as q and one of Bob's records as x. This Secret Query Database must allow them to cooperate in such a way that they can compute $q(x)$ while Alice preserves the secrecy of q and Bob preserves that of x. The result of $q(x)$ is a single bit: a one if x satisfies q or a zero if it doesn't. In case the outcome of $q(x)$ is a one, Bob sends the record to Alice and Alice pays for it. In case it is a zero, Bob keeps the record for himself.

What exactly is meant by *secrecy of the query* is a fine matter. In this paper, we try to achieve the following level of secrecy: in the assumption that the query is completely random, that E is the set of already evaluated records (so Bob knows $q(x)$ for every $x \in E$) and that $y \notin E$, Bob should not be able to predict $q(y)$ non-negligibly better than random guessing. This definition allows Bob to know *which* records Alice wants, but not *why* she wants those records. If we look at q as a huge truth table with on every line a possible record as input and the result bit of the query for that record as output, we can say that Bob learns only the lines that correspond to records he evaluated together with Alice, but he can't tell anything about the other lines. A stronger definition would be for Bob to learn only *how many* records are selected (for payment purposes) but not the exact identity of the records. This would lead us to a generalization of Private Information Retrieval (PIR), introduced in [3], which is a conceptually different problem.

No matter which protocol is used, there must always be some minimal interaction. Neither of both parties should be able to evaluate a query without any help from the other. More specifically, after evaluating $q(x)$ together, Bob should not be able to learn $q(x')$ without help from Alice and in the same way Alice should not be able to calculate $q'(x)$ all by herself. If the first condition doesn't hold, it is trivial to see that our definition of secrecy of Alice's query is violated. If the second doesn't hold, Alice could subsequently evaluate the queries "is the ith bit of x a one?" for every bit in x and thus obtain the entire record x without paying for it.

Most protocols for secure distributed computing represent the function to be evaluated as a logical circuit. An important issue if we want to estimate the communication complexity is the number of gates and inputs needed in those circuits. With the concrete example of Bob's database containing CVs in mind, we use the following notation: the size of a single record is i_s bits or $i_s/8$ bytes. A circuit representing a query takes C gates on the average. Realistic values for the example of the Secret Query Database are: $i_s = 4000, C = 1000$. Some protocols allow one party to hide the circuit from the other parties, but some don't. If we want to use the latter to implement a Secret Query Database, some kind of universal circuit is needed where Bob's secret input is still one of his records but Alice's secret input is an encoding of her query (in other words, as discussed in the introduction, we need an 'interpreter circuit' that can interpret a simple query language). The number of gates of such a universal circuit is denoted as C_u. It is feasible to construct such a universal circuit allowing most common queries using $C_u = 4000$ gates. The encoding of a query would take about $i_{cu} = 1000$ bits.

Protocols for secure distributed computing typically require a lot of network resources. Sometimes, we can reduce the communication overhead if Bob publishes reusable data on CD-ROM. By *reusable*, we mean that it must be possible to use this data over and over again in multiple evaluations without leaking any additional information about the records or the query, even if these evaluations are performed with gossiping clients or — which is essentially the same thing — with one and the same client. This method is most useful for slowly evolving databases like the one containing CVs. Bob can then publish this CD-ROM on a regular but not too frequent basis in relatively large quantities.

In the remaining text of this paper, Bob is referred to as the *server*, while Alice is called the *client*.

3 Possible Solutions

Various kinds of solutions for the secure distributed computing problem have been proposed in the literature (often using different terminology than the one used in this paper). We distinguish four kinds of solutions, and discuss how the Secret Query Database could be implemented using them.

3.1 Protocols based on symmetric cryptography and oblivious transfer

In [6], Goldreich, Micali and Wigderson present a two-party protocol for the problem of *combined oblivious transfer* which is equivalent to the problem of secure circuit evaluation. The two players A and B proceed as follows. B assigns two random bit strings r_i^0 and r_i^1 to every wire i in the circuit, which represent an encoded 0 and 1 on that wire. This defines a mapping $\phi_i : r_i^b \mapsto b$ for every wire i. B also chooses a random bit string R that will allow A to check if a decryption key is correct. The general idea of the protocol is that, if b is the bit on wire i in the evaluation of the circuit for A's and B's secret inputs, A will only find out about

r_i^b and will never get any information about $\phi_i(r_i^b)$ or $r_i^{\overline{b}}$. In other words, A evaluates the circuit with encoded data.

We use the notation $E(M, r)$ for a symmetric encryption function of the message M with secret key r. To encrypt a NOT-gate with input wire i and output wire o, B constructs a random permutation of the tuple

$$< E(R \cdot r_o^1, r_i^0), E(R \cdot r_o^0, r_i^1) >$$

where $\cdot$ denotes the concatenation of bit strings. To encrypt an AND-gate with input wires l and r and output wire o, B constructs a random permutation of the tuple

$$< E(R \cdot r_o^0, r_l^0 \oplus r_r^0), E(R \cdot r_o^0, r_l^0 \oplus r_r^1),$$
$$E(R \cdot r_o^0, r_l^1 \oplus r_r^0), E(R \cdot r_o^1, r_l^1 \oplus r_r^1) >$$

with $\oplus$ the bit-wise XOR. Any other binary port can be encrypted in an analogous way.

B sends the encryption of every gate in the circuit together with R, the encoding of his own input bits and the mapping ϕ_m of the output wire m to A. To perform the evaluation of the circuit on encoded data, A first needs encodings of all the input bits. For B's input bits, the encoding was sent to her, but since B doesn't know A's inputs, B can't send an encoding of them. Note that B can't send the encoding of both a 1 and a 0 on A's input wires either, because that would allow A to find out more than just the result of the circuit. The technique that is used to get the encoding of A's input to A is called *one-out-of-two oblivious transfer* ([7]). This is a protocol that allows A to retrieve one of two data items from B in such a way that (1) A gets exactly the one of two items she chose and (2) B doesn't know which item A has got.

Thus, A and B execute a one-out-of-two oblivious bit string transfer (often referred to as $\binom{2}{1}$-OT^k) for each of A's input bits. This guarantees that A only obtains the encoding of her own input bits without releasing any information about her bits to B. A evaluates each gate by trying to decrypt every element of the tuple using the encoding of the bit on the input wire (or the XOR of two input bit encodings) as a key; she will only decrypt one of the elements successfully, thereby obtaining the encoded bit on the output wire. Note that she can know if a decryption was correct by comparing the first bits of the decrypted string with R. Proceeding this way through the entire circuit, A obtains the encoding of the final output and applies ϕ_m to reveal the plain output bit.

How can this protocol fit in an implementation of a Secret Query Database? The circuit is visible to both parties (it is not an ambition of the protocol to hide the circuit), so we need a universal circuit. The server plays the role of B, the circuit scrambler, and the client takes A's part, the circuit evaluator. Evaluating multiple records on the same encrypted circuit should be prevented: the client would learn a lot of information about the records. For every record evaluation, a new encryption of the circuit has to be constructed and transmitted. To save network resources, the server can publish the encoding of his input bits (i.e. the records) on CD-ROM without jeopardizing the secrecy of the records: in fact it's just a CD-ROM filled with random bits. The circuits the server constructs afterwards to evaluate these records must of course use the same mappings for his input bits as used on the CD-ROM.

The encoding of the client's query, which is her secret input, remains the same for all records — there's no reason to change it, any record looks the same to her. The server can choose to construct the different instances of the circuit for evaluation of the same query with identical mappings for the client's inputs. The advantage of this technique is that the $\binom{2}{1}$-OT^k only has to be executed once instead of once for every record.

If we use a symmetric 64-bit block cipher as encryption function, we can estimate the data complexity of the entire protocol as follows. Each encoding of a record takes $8i_s$ bytes on CD-ROM. An encrypted NOT-gate takes 32 bytes and an AND-gate takes 64 bytes, so on the average an encrypted gate takes 48 bytes, which means a scrambled (universal) circuit takes about $48C_u$ bytes for each record. Using the implementation described in [7] with $|p| = 512$, the oblivious transfers take an additional $384i_{cu}$ bytes (independent of the number of records) to be sent over the network.

3.2 Using probabilistic encryption

Another technique to compute with encrypted data is based on *homomorphic probabilistic encryption*. An encryption technique is *probabilistic* if the same cleartext can encrypt to many different ciphertexts. To work with encrypted bits, probabilistic encryption is essential, otherwise only two ciphertexts (the encryption of a zero and the encryption of a one) would be possible, and cryptanalysis would be fairly simple. An encryption technique is *homomorphic* if it satisfies equations of the form $E(x \quad \mathbf{op} \quad y) = E(x) \quad \mathbf{op}' \quad E(y)$ for some operations $\mathbf{op}$ and $\mathbf{op}'$. A homomorphic encryption scheme allows operations to be performed on encrypted data, and hence can be used for secure circuit evaluation.

Abadi and Feigenbaum present a protocol for two-player secure circuit evaluation using a homomorphic probabilistic encryption scheme based on the Quadratic Residuosity Assumption (QRA) in [1]. This protocol allows A who has a secret function f and B who has secret data x to calculate $f(x)$ without jeopardizing their secrets. More precisely, the protocol allows A to hide her circuit unconditionally except for the number of AND-gates, while B hides his data under the QRA.

Let k be the product of two primes p and q, each congruent to 3 mod 4. An integer $a \in Z_k^*[+1]$ — the integers relatively prime to k with Jacobi symbol 1 — is a quadratic residue mod k if there exists an $x \in Z_k^*[+1]$ such that $a = x^2 \bmod k$. The QRA states that determining if an integer a is a quadratic residue mod k is a hard problem if the factorization of k is unknown but is easy to solve if p and q are given.

If we encrypt a zero by a quadratic residue and a one by a quadratic nonresidue mod k, we can define the encryption of a bit b as

$$E_k(b) = (-1)^b \cdot r^2 \bmod k$$

with $r \in_R Z_k^*[+1]$ chosen at random. This probabilistic encryption scheme has two homomorphic properties that will come in very handy in the protocol:

$$E_k(\overline{b}) = (-1) \cdot E_k(b) \bmod k$$

$$E_k(b_1 \oplus b_2) = E_k(b_1) \cdot E_k(b_2) \bmod k$$

B starts the protocol by choosing p and q and multiplying them to produce k. B sends k and the encryption of his data bits $E_k(x_1), \ldots, E_k(x_n)$ to A. B keeps the factorization of k secret. A then starts evaluating her secret circuit. If she has to evaluate a NOT gate with input $E_k(b)$, she simply calculates $-E_k(b) \bmod k$. An XOR with inputs $E_k(b_1)$ and $E_k(b_2)$ is also easy to evaluate: A just takes $E_k(b_1) \cdot E_k(b_2) \bmod k$ as the output of the gate. To evaluate the AND of inputs $E_k(b_1)$ and $E_k(b_2)$, she needs B's help. A chooses two bits c_1 and c_2 at random and sends $E_k(b_1 \oplus c_1)$ and $E_k(b_2 \oplus c_2)$ to B. B decrypts the bits A just sent him as d_1 and d_2 (he can do so because he knows p and q) and sends the tuple

$$< E_k(d_1 \wedge d_2), E_k(d_1 \wedge \overline{d_2}), E_k(\overline{d_1} \wedge d_2), E_k(\overline{d_1} \wedge \overline{d_2}) >$$

to A. A takes the first element of this tuple as the output of the AND gate if she chose $c_1 = c_2 = 0$, the second if she chose $c_1 = 0$ and $c_2 = 1$, the third if she chose $c_1 = 1$ and $c_2 = 0$ and the last one if she chose $c_1 = c_2 = 1$. Proceeding this way from gate to gate, A ends with the encrypted result $E_k(f(x))$ and sends it for decryption to B.

The evaluation of an AND-gate requires six encrypted bits to be sent over the network. We can reduce this number to three by applying a little trick Franklin and Haber used in [5]. Just like the original protocol, A chooses random bits c_1 and c_2 and sends $E_k(b_1 \oplus c_1)$ and $E_k(b_2 \oplus c_2)$ to B. B again decrypts these bits as d_1 and d_2 but sends only $E_k(d_1 \wedge d_2)$ back to A. Now, it is easy to prove that

$$\begin{aligned} d_1 \wedge d_2 &= (b_1 \oplus c_1) \wedge (b_2 \oplus c_2) \\ &= (b_1 \wedge b_2) \oplus (b_1 \wedge c_2) \oplus \\ &\quad\ (b_2 \wedge c_1) \oplus (c_1 \wedge c_2) \end{aligned}$$

or, applying the homomorphism between XOR and multiplication mod k,

$$\begin{aligned} E_k(b_1 \wedge b_2) &= E_k(d_1 \wedge d_2) \cdot E_k(b_1 \wedge c_2) \cdot \\ &\quad\ E_k(b_2 \wedge c_1) \cdot E_k(c_1 \wedge c_2) \end{aligned}$$

A just got $E_k(d_1 \wedge d_2)$ from B and she can easily make an encryption of $c_1 \wedge c_2$. Computing $E_k(b_1 \wedge c_2)$ is a little more subtle. If she chose c_2 to be 0, she can take any encryption of a zero to be that factor. If she chose c_2 to be 1, she uses $E_k(b_1)$ itself. The last factor, $E_k(b_2 \wedge c_1)$, is calculated in an analogous way.

This protocol can do a great job in a Secret Query Database. Unlike the previous protocol, we don't need a universal circuit if we let the client play the role of A and let the secret function be her query q itself. The server plays the role of B, supplying encrypted records as his secret data.

The length of k is an important security parameter: if a client is able to compute the factorization of k, she can decrypt the record. At this moment, it is possible to factor 512-bit numbers if you are willing to spend a few thousands of dollars and wait a few months. The plaintext of the entire database could be worth this effort, but one single record probably isn't. If we use a new 512-bit k for every record, an attacker has to factor a 512-bit number for every record he wants. For most databases it will be cheaper simply to buy the records from the server.

In spite of the relatively small key length, the encryption of a record still causes a huge data blow: it's size is multiplied by a factor of 512. Our records of $i_s/8$ bytes take $64i_s$ bytes in encrypted form. Of course, it is intolerable to send this kind of data over a network. Yet the factors of k are not revealed during the protocol, allowing the data to be reused in multiple evaluations. Again, the server can publish encrypted records and the public key k used for the encryption on CD-ROM to drastically reduce network load.

As we stated above, the evaluation of an AND-gate needs three encrypted bits to be sent over the network. If the client's circuit consists of C gates, half of which are NOT-gates or XOR-gates that require no interaction and half of which are AND-gates that do require interaction, on the average $96C$ bytes are to be sent over the network for each record evaluation.

3.3 Autonomous protocols

The protocols discussed in the two previous subsections require more communication rounds than strictly necessary. The oblivious transfer based protocol requires one communication round for performing the oblivious transfer of the input, and another for sending the encrypted circuit. The probabilistic encryption based protocol requires one communication round per AND-gate in the circuit.

Although it is impossible (as discussed in section 2) to construct a completely non-interactive solution, it turns out to be possible to construct solutions using only one communication round: the client sends (in one message) an encrypted function f, and it receives from the server an encrypted result $f(x)$ in such a way that f remains private to the client and x remains private to the server. A protocol that achieves this result in only one communication round is called an *autonomous* protocol.

Two kinds of autonomous protocols have been proposed in the literature. The first kind, introduced by Sander and Tschudin ([10, 11]), allows for a fairly efficient evaluation of polynomials in a ring of integers modulo n using a homomorphic encryption scheme. However, it is not clear if this technique can be extended to general boolean circuits, and hence it does not seem applicable to the Secret Query Database.

The second kind, introduced by Loureiro and Molva ([8]), uses a public key encryption system based on Goppa codes, and allows for the evaluation of functions describable by a matrix multiplication. Loureiro and Molva also show how any boolean circuit evaluation can be done by a matrix multiplication. However, the representation of a boolean circuit requires a *huge* matrix (for a circuit with l inputs, one of the dimensions of the matrix is 2^l), which renders the technique hopelessly inefficient for application in the Secret Query Database. It remains an open problem whether more efficient representations of boolean circuits as matrices can be achieved.

Hence, for the current state of the art of autonomous protocols, it seems that these protocols cannot compete with the protocols discussed in the sections above.

3.4 Multiparty protocols

Obviously, protocols that solve the problem of multiparty secure computation can also be applied to the two-party case. Chaum, Damgård and van de Graaf present a multiparty protocol in [2] that starts with the truth table of every gate in the circuit. Each player in turn receives a "scrambled" version of the truth tables from the previous player, transforms the truth tables by adding his own encryptions and permutations, commits to his encryptions and sends these transformed truth tables on to the next player. When the last player finished his transformation, all players evaluate the scrambled circuit by selecting the appropriate row from the truth tables.

Franklin and Haber present an elegant multiparty protocol based on group-oriented cryptography in [5]. All parties send each other an El-Gamal alike joint encryption of their input bits and evaluate the entire circuit together. The evaluation of a NOT-gate can be done without interaction while the evaluation of an AND-gate requires broadcasting encrypted bits and "decryption witnesses". Finally, each party sends a decryption witness for the output bit.

Although these protocols are in many ways stronger and more general than the two-party protocols discussed in previous sections, they prove to be less suitable for our concrete application of a Secret Query Database. Compared with genuine two-party protocols, they both use far more network resources. Hence, we will not consider them in our assessment.

4 Assessment

We compare the two most promising solutions of all the solutions discussed in section 3. A distinction is made between (1) the amount of data that can be published once and for all (e.g. on a CD-ROM), (2) the amount of data that must be exchanged once per query (but can be reused when evaluating the query on successive records) and (3) the amount of data that must be exchanged for each individual query evaluation. We call the first kind of data *CD-ROM data*, the second kind *query setup data* and the third kind *query evaluation data*. We consider the (realistic) numerical values discussed in section 2, and assume a database of $n_r = 1024$ records. Table 1 summarizes the communication overhead.

For the oblivious transfer based solution, the total network communication overhead to query the entire database becomes 375K $+1024 \cdot 190$K or about 190 megabytes. For the probabilistic encryption based solution, this total overhead becomes $1024 \cdot 94$K or 94 megabytes.

Although this communication overhead is indeed high, it is important to note that:

1. Communication overhead can be reduced if the required security level can be lowered. For example, we assumed a 512 bit crypto-system for the second solution. This number could be reduced if the price of the data in each record is rather small. Similarly, we achieved very high secrecy of the query, revealing only the number of gates in the circuit. The communication overhead can be further reduced if more information of the structure of the query is allowed to leak to the server.

2. Communication overhead can be traded off against query complexity. For example, in solution 1, if a simpler query language is used, the size of the universal circuit used in the protocol can be reduced, strongly reducing the network overhead.

3. By using agent technology, communication can be made more efficient: the agents can position themselves close to one another. Of course, this assumes that there are trusted hosts that the agents can migrate to, and since the Secret Query Database can be solved much more efficiently using a trusted third party, one might think at first sight that the use of agents doesn't have any advantages. Note however that, in the agent scenario, the trusted host doesn't have to know the Secret Query Database protocol: the host just has to offer a secure execution platform. Hence it can be reused for many similar applications. In contrast, a dedicated trusted third party for the Secret Query Database problem can be made more efficient, but is only usable for this particular application. To support new applications, another server has to be started, or the existing server has to be reprogrammed.

Given all these considerations, we believe it is fair to state that secure distributed computing technology is ready to be considered for practical applications.

5 Conclusion

This paper has made an evaluation of the practical feasibility of secure distributed computing. We presented a concrete application of this advanced cryptographic technology, the Secret Query Database, that is a representative instance of a whole class of interesting applications using secure distributed computing techniques.

Next, we gave a survey of solutions for the secure distributed computing problem. This survey was strongly biased towards our concrete application: only techniques applicable to the Secret

	CD-ROM	query setup	query evaluation
OT-based solution	$8n_r i_s$ bytes = 32 Mbytes	$384 i_{cu}$ bytes = 375 Kbytes	$48C_u$ bytes/record = 190 Kbytes/record
PE-based solution	$64 n_r i_s$ bytes = 250 Mbytes	0 bytes	$96C$ bytes/record = 94 Kbytes/record

Table 1: Summary of communication overhead

Query Database were elaborated in detail, and were optimized for the case study. Techniques that were clearly too inefficient for our application were only shortly mentioned and references to more detailed treatments were given.

Finally, we assessed the resulting communication overhead for the two most suitable solutions, leading to the conclusion that, by trading off security level and expressiveness of possible queries against acceptable communication overhead, a practically implementable solution can be achieved, especially when agent-technology is involved.

References

[1] M. Abadi and J. Feigenbaum, "Secure circuit evaluation, a protocol based on hiding information from an oracle," Journal of Cryptology, 2(1), p. 1–12, 1990

[2] D. Chaum, I. Damgård and J. van de Graaf, "Multiparty computations ensuring privacy of each party's input and correctness of the result," in *Advances in Cryptology—CRYPTO '87 Proceedings* (Lecture Notes in Computer Science, Vol. 293), ed. C. Pomerance, p.87–119 , Springer-Verlag, New York, 1988

[3] B. Chor, O. Goldreich, E. Kushilevitz and M. Sudan, "Private information retrieval," Proc. of 36th IEEE Conference on the Foundations of Computer Science (FOCS), p. 41–50, 1995

[4] M. Franklin, "Complexity and security of distributed protocols," Ph. D. thesis, Computer Science Department of Columbia University, New York, 1993

[5] M. Franklin and S. Haber, "Joint encryption and message-efficient secure computation," Journal of Cryptology, 9(4), p. 217–232, Autumn 1996

[6] O. Goldreich, S. Micali and A. Wigderson, "How to play any mental game," Proc. of 19th ACM Symposium on Theory of Computing (STOC), p. 218–229, 1987

[7] B. Litow, "Introduction to cryptography," 1999, available at URL: `http://www.cs.jcu.edu.au/ftp/web/teaching/Subjects/cp5030/1999/`

[8] S. Loureiro and R. Molva, "Privacy for Mobile Code", Proceedings of the workshop on *Distributed Object Security*, OOPSLA '99, p. 37–42.

[9] N. Nisan, "Algorithms for selfish agents", *Proceedings of the 16th Annual Symposium on Theoretical Aspects of Computer Science*, Trier, Germany, March 1999, p. 1–15.

[10] T. Sander and C. Tschudin, "On software protection via function hiding", *Proceedings of the second workshop on Information Hiding*, Portland, Oregon, USA, April 1998.

[11] T. Sander and C. Tschudin, "Towards mobile cryptography", *Proceedings of the 1998 IEEE Symposium on Security and Privacy*, Oakland, California, May 1998.