Implementing Secure Distributed Computing with Mobile Agents

Gregory Neven[1] — **Erik Van Hoeymissen** — **Bart De Decker** — **Frank Piessens**[2]

Dept. of Computer Science, K.U.Leuven
Celestijnenlaan 200A, B-3001 Leuven, Belgium

E-mail:{gregory, erikv, bart, frank}@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 two decades, a number of solutions requiring no trusted third party have been developed using cryptographic techniques. The disadvantage of these cryptographic solutions is the excessive communication overhead they incur.*
In this paper, we use mobile agents employing these cryptographic techniques to provide for a trade-off between communication overhead and trust. We solve the communication overhead problem by using mobile agents to execute the cryptographic protocols and running these agents on hosts that are close to each other. Of course, a mobile agent needs to trust his execution platform, but we show that the trust requirements in this case are much lower than for a classical trusted third party. As a practical case study, we look at the implementation of a second price auction.

RÉSUMÉ. *Le problème des calculs secrets distribués est le suivant: donné un certain nombre de participants méfiants, comment peuvent-ils évaluer une fonction de telle manière que chacun des participants n'ait qu'un accès limité aux données d'entrée des autres? Plusieurs solutions pour ce problème en utilisant des techniques cryptographiques ont déjà été publiées. Le désavantage de ces solutions est la quantité excessive de dates transmises.*
Dans cet article-ci, nous utilisons les agents mobiles pour trouver un équilibre entre la complexité de communication et la confiance. L'évaluation des protocoles cryptographiques est faite par des agents mobiles qui peuvent exécuter sur des serveurs qui sont près l'un de l'autre. Bien sur, un agent mobile doit faire confiance à son plateforme d'exécution, mais nous montrons que cette confiance est inférieure à celle d'une partie centrale fiable classique. Comme étude de cas pratique, nous discutons l'implémentation d'une enchère à deuxième prix.

KEYWORDS: *secure distributed computing, mobile agents, second price auction*

MOTS-CLÉS : *calculs secrets distribués, agents mobiles, enchère à deuxième prix*

1. Research Assistant of the Fund for Scientific Research — Flanders, Belgium (F.W.O.)
2. Postdoctoral Fellow of the Belgian National Fund for Scientific Research (F.W.O.)

1re soumission à *Networking and Information Systems Journal*, le October 30, 2000.

1. Introduction

Secure distributed computing (SDC) 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 $p_i, i = 1 \ldots n$), each holding one private input x_i. The n parties want to compute the value $f(x_1, \ldots, x_n)$ 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 ([NIS 99]), charging for the use of algorithms on the basis of a usage count ([SAN 98, SAN 98b]), various kinds of weighted voting, protecting mobile code integrity and privacy ([SAN 98b, LOU 99]), ...

Secure distributed computing is trivial in the presence of a globally trusted third party(TTP): all participants send their data and code to the TTP (over a secure channel), the TTP performs the computation and broadcasts the results. The main drawback of this approach is the large amount of trust needed in the TTP.

However, solutions without a TTP are also possible. Over the past two decades, a fairly large variety of solutions to the problem has been proposed. An overview is given by Franklin [FRA 93] and more recently by Cramer [CRA 99]. 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). The main drawback of these solutions is the heavy communication overhead that they incur. For a case-study investigating the communication overhead in a concrete example application, we refer the reader to [NEV 00].

Mobile agents employing these cryptographic techniques can provide for a trade-off between communication overhead and trust. The communication overhead is alleviated if the communicating parties are brought close enough together. In our approach, every participant sends its representative agent to a trusted execution site. The agent contains a copy of the private data x_i and is capable of running a SDC-protocol. Different participants may send their agents to different sites, as long as these sites are

located closely to each other. Of course, a mobile agent needs to trust his execution platform, but we show that the trust requirements in this case are much lower than for a classical TTP. Also, in contrast with protocols that use unconditionally TTPs, the trusted site is not involved directly. It simply offers a secure execution platform: i.e. it executes the mobile code correctly, does not spy on it and does not leak information to other mobile agents. Moreover, the trusted host does not have to know the protocol used between the agents. In other words, the combination of mobile agent technology and secure distributed computing protocols makes it possible to use a *generic* TTP that, by offering a secure execution platform, can act as TTP for a wide variety of protocols in a uniform way. A detailed discussion of the use of mobile agent technology for advanced cryptographic protocols is given in section 3.

The combination of cryptographic techniques for secure computing and mobile code has been investigated from another point of view by Sander and Tschudin ([SAN 98, SAN 98b]). In their paper on mobile cryptography, they deal with the protection of mobile agents from possibly malicious hosts. Hence, the focus in their work is on the use of cryptographic techniques for securing mobile code. The security concerns posed by the mobile agent protection problem are code privacy (Can a mobile agent conceal the program it wants to have executed?), code and execution integrity (Can a mobile agent protect itself against tampering by a malicious host?) and computing with secrets in public (Can a mobile agent remotely sign a document without disclosing the user's private key?). To address some of these concerns, cryptographic secure computation techniques can be used. We discuss this in more detail in section 2.3, which is part of our survey on secure distributed computing protocols.

The structure of this paper is as follows. In the next section, we start of with a survey of existing cryptographic solutions to the secure computing problem. In section 3, we introduce and compare three possible ways to implement secure distributed computing, making use of both cryptographic techniques, and trusted parties. The comparison is done based on a simple model of the trust and communication requirements for each of the solutions. In section 4, we focus on an example application of secure distributed computing. More precisely, we will show how multi-party secure computations can be used to perform second price auctions and we will assess the incurred communication overhead. Finally, in section 5, we summarize the main outcomes of this article.

2. Survey of SDC protocols

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).

2.1. *Using probabilistic encryption*

One class of techniques 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 [ABA 90]. This protocol allows A who has a secret function f and B who has secret data x to calculate $f(x)$ without revealing their secrets.

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 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.

Note the large amount of communication overhead in the protocol: for each AND gate to be evaluated, a large amount of communication is necessary. Concrete estimates of the communication overhead in a realistic example can be found in [NEV 00]

2.2. *Protocols based on oblivious transfer*

In [GOL 87], 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 setting is slightly different than in the previous protocol. Here, two parties A and B want to evaluate a publicly known boolean circuit. This circuit takes input from both A and B, but each party wants to keep his part of the data private. In contrast, in the previous protocol, the circuit was private to A, and the data was private to B. Recall from the introduction that these two settings are essentially equivalent: by making the publicly known circuit a universal circuit, it is still possible to hide functions instead of data.

The basic idea of the protocol we are about to describe is the following: A will evaluate the circuit, not on the actual bits, but on encodings of those bits. The encoding of the bits is known only to B. So A evaluates the circuit, but can not make sense of intermediary results because she doesn't know the encoding. B knows the encoding but never gets to see the intermediary results. When the final result is announced by A (in encoded form), B will announce a decoding for this final result.

We give a more detailed description of the protocol. 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* ([EVE 85]). 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 verify 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.

Another protocol for two-party secure computation based on oblivious transfer is presented in [GOL 87b]. The basic idea in this protocol is to have the participants compute the circuit on data that is shared by the two parties using a technique known as *secret sharing*.

2.3. *Autonomous protocols*

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

For protecting mobile code privacy and integrity, non-interactive (or autonomous) protocols are necessary ([SAN 98b]). The idea here is to realize a system where a host

can execute an encrypted function without having to decrypt it. Thus, functions would be encrypted such that the resulting transformation can be implemented as a mobile program that will be executed on a remote host. The executing computer will be able to execute the program's instructions but will not be able to understand the function that the program implements. Having function and execution privacy immediately yields execution integrity: an adversary can not modify a program in a goal-oriented way. Modifying single bits of the encrypted program would disturb its correct execution, but it is very hard to produce a desired outcome.

It turns out to be possible to construct such autonomous solutions where 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.

Various autonomous protocols have been proposed in the literature. Sander and Tschudin ([SAN 98, SAN 98b]) introduce a technique that allows for a fairly efficient evaluation of polynomials in a ring of integers modulo n using a homomorphic encryption scheme. They also show how an autonomous protocol could be realized using compositions of rational functions.

Sander and Tschudin emphasize in their paper that securing single functions is not sufficient. Consider for example the problem of implementing a digital signing primitive for mobile agents. Even if the real signature routine can be kept secret, still the whole (encrypted but operational) routine might be abused to sign arbitrary documents. Thus, the second task is to guarantee that cryptographic primitives are unremovably attached to the data to which they are supposed to be applied (the linking problem). The general idea behind the solution here is to compose the signature generating function s with the function f of which the output is to be signed. Crucial for the security of this scheme is the difficulty of an adversary to decompose the final function into its elements s and f. An outline of how this could be implemented using rational functions is given in [SAN 98b].

Loureiro and Molva ([LOU 99]) use a public key encryption system based on Goppa codes. Their protocol 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). It remains an open problem whether more efficient representations of boolean circuits as matrices can be achieved.

Finally, two very recent papers also focus on autonomous protocols: Sander, Young and Yung ([YUN 00]) propose an autonomous protocol based on a new homomorphic encryption scheme, and Cachin, Camenisch, Kilian and Müller ([CAC 00]) start from an OT-based SDC protocol as in section 2.2, and succeed in merging the two phases of this protocol into one.

It is worth emphasizing that, even though autonomous protocols use the minimal number of messages, they do not solve the communication overhead problem: even though there are only two messages exchanged, these messages are extremely large.

2.4. *Using group-oriented cryptography*

All the previous protocols concentrate on the two-party case: only two parties are involved in the secure computation process. It is clear that the multi-party case is even more interesting from an application-oriented point of view. The multi-party case has also received considerable interest in the literature.

In [FRA 96], Franklin and Haber propose a protocol that is somewhat similar to the protocol by Abadi and Feigenbaum ([ABA 90]), in the sense that this protocol too evaluates a boolean circuit on data encrypted with a homomorphic probabilistic encryption scheme. The major difference between the two protocols, however, is that this protocol allows any number of parties to participate in the secure computation, while Abadi and Feigenbaum's protocol is restricted to two parties.

To extend the idea of [ABA 90] to the multi-party case, we need an encryption scheme that allows anyone to encrypt, but needs the cooperation of all participants to decrypt. In a joint encryption scheme, all participants know the public key K_{pub} while each participant $P_1, \ldots, P_n$ has his own private key $K_1, \ldots, K_n$. Using the public key, anyone can create an encryption $E_S(m)$ of some message m, where $S \subseteq \{P_1, \ldots, P_n\}$, such that the private key of each participant in S is needed to decrypt. More formally, if D_i denotes the decryption with P_i's private key, the relation between encryption and decryption is given by

$$D_i(E_S(m)) = E_{S \setminus \{P_i\}}(m)$$

The plaintext m should be easily recoverable from $E_\emptyset(m)$.

In the joint encryption scheme used by Franklin and Haber, a bit b is encrypted as

$$E_S(b) = \left[g^r \mod N,\ (-1)^b \left(\prod_{j \in S} g^{K_j} \right)^r \mod N \right]$$

where $N = pq$, p and q are two primes such that $p \equiv q \mod 4$, and $r \in_R Z_N$. The public key is given by $[N, g, g^{K_1} \mod N, \ldots, g^{K_n} \mod N]$. This scheme has some additional properties that are used in the protocol:

– *XOR-Homomorphic.* Anyone can compute a joint encryption of the XOR of two jointly encrypted bits. Indeed, if $E_S(b) = [\alpha, \beta]$ and $E_S(b') = [\alpha', \beta']$, then $E_S(b \oplus b') = [\alpha\alpha' \mod N, \beta\beta' \mod N]$.

– *Blindable.* Given an encrypted bit, anyone can create a random ciphertext that decrypts to the same bit. Indeed, if $E_S(b) = [\alpha, \beta]$ and $r \in_R Z_N$, then

$$\left[\alpha g^r \mod N,\ \beta \left(\prod_{j \in S} g^{K_j} \right)^r \mod N \right]$$

is a joint encryption of the same bit.

– *Witnessable.* Any participant can withdraw from a joint encryption by providing the other participants with a single value. Indeed, if $E_S(b) = [\alpha, \beta]$, it is easy to compute $D_i(E_S(b))$ from

$$W_i([\alpha,\ \beta]) = \alpha^{-K_i} \mod N$$

First of all, the participants must agree on a value for N and g, choose a secret key K_i and broadcast $g^{K_i} \mod N$ to form the public key. To start the actual protocol, each participant broadcasts a joint encryption of his input bits. For an XOR-gate, everyone simply applies the XOR-homomorphism. The encrypted output of a NOT-gate can be found by applying the XOR-homomorphism with a default encryption of a one, e.g. $[1, -1]$.

Again, it is the AND-gate that causes some trouble. Suppose the encrypted input bits for the AND-gate are $\hat{u} = E(u)$ and $\hat{v} = E(v)$. To compute a joint encryption $\hat{w} = E(w) = E(u \wedge v)$, they proceed as follows:

1. Each participant P_i chooses random bits b_i and c_i and broadcasts $\hat{b}_i = E(b_i)$ and $\hat{c}_i = E(c_i)$.

2. Each participant repeatedly applies the XOR-homomorphism to calculate $\hat{u}' = E(u') = E(u \oplus b_1 \oplus \ldots \oplus b_n)$ and $\hat{v}' = E(v') = E(v \oplus c_1 \oplus \ldots \oplus c_n)$. Each participant broadcasts decryption witnesses $W_i(\hat{u}')$ and $W_i(\hat{v}')$.

3. Everyone can now decrypt $\hat{u}'$ and $\hat{v}'$. We have the following relation between $w' = u' \wedge v'$ and $w = u \wedge v$:

$$\begin{aligned}
w' &= u' \wedge v' \\
&= (u \oplus b_1 \oplus \cdots \oplus b_n) \wedge (v \oplus c_1 \oplus \cdots \oplus c_n) \\
&= (u \wedge v) \ \oplus\ (u \wedge c_1) \ \oplus \cdots \oplus\ (u \wedge c_n) \\
&\qquad \oplus\ (b_1 \wedge c_1) \ \oplus \cdots \oplus\ (b_n \wedge c_1) \\
&\qquad \vdots \qquad\quad \vdots \qquad\qquad\ \vdots \\
&\qquad \oplus\ (b_1 \wedge c_n) \ \oplus \cdots \oplus\ (b_n \wedge c_n) \\
&\qquad \oplus\ \underbrace{(b_1 \wedge v)}_{w_1} \ \oplus \cdots \oplus\ \underbrace{(b_n \wedge v)}_{w_n} \\
&= (u \wedge v) \oplus w_1 \oplus \cdots \oplus w_n
\end{aligned}$$

Each participant is able to compute a joint encryption of w_i: he knows b_i and c_i (he chose them himself) and he received encryptions $\hat{c}_j$ from the other participants, so he can compute $E(b_i \wedge c_j)$ as follows:

– If $b_i = 0$, then $b_i \wedge c_j = 0$, so any default encryption for a zero will do, e.g. $[1, 1]$.

– If $b_i = 1$, then $b_i \wedge c_j = c_j$, so $\hat{c}_j$ is a valid substitution for $E(b_i \wedge c_j)$.

$E(u \wedge c_i)$ and $E(v \wedge b_i)$ can be computed in an analogous way. He uses the XOR-homomorphism to combine all these terms, blinds the result and broadcasts this as $\hat{w}_i$.

4. Each participant combines $\hat{w}'$ and $\hat{w}_j$ ($j = 1 \ldots n$), again using the XOR-homomorphism, to form $\hat{w} = E(w)$.

When all gates in the circuit have been evaluated, every participant has a joint encryption of the output bits. Finally, all participants broadcast decryption witnesses to reveal the output.

2.5. *Other multi-party protocols*

Chaum, Damgård and van de Graaf present a multi-party protocol in [CHA 87] 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 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.

Even information-theoretically secure multi-party computation can be achieved (as opposed to only computationally secure). A possible realisation is discussed in [CRA 99].

The communication overhead for multi-party protocols is even more serious than that for the two-party protocols.

3. Trust versus communication overhead

In this section, the different options for implementing secure distributed computation are discussed. It will be shown that there is a trade-off between trust and communication overhead in secure computations. If all participants are distrustful of each other, the secure computation can be performed using protocols surveyed in the previous section with a prohibitive huge amount of communication. However, if a TTP is involved, the communication overhead can be made minimal.

Recall from section 1 that f is a publicly known function taking n inputs. Assume that there are n distrustful participants $p_1, ..., p_n$, each holding one private input x_i. The n participants want to compute the value of $f(x_1, ..., x_n)$ without leaking information of their private inputs to the other participants.

To compare the trust requirements of the different approaches, we use the following simple trust model. We say a participant *trusts* an execution site if it believes that:

– the execution site will correctly execute any code sent to it by the participant;

– the execution site will correctly (i.e. as expected by the participant) handle any data sent to it by the participant.

It also implies that the execution site will maintain the privacy of the data or the code if this is expected by the participant. If p trusts E, we denote this as shown in figure 1.

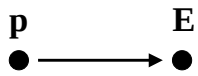

Figure 1. *Notation for "p trusts E"*

To compare bandwidth requirements (for communication overhead), we make the following simple distinction. *High* bandwidth is required to execute a SDC protocol. *Low* bandwidth suffices to transmit data or agent code. We assume low bandwidth communication is available between any two sites. If high bandwidth communication is possible between E_i and E_j, we denote this as shown in figure 2.

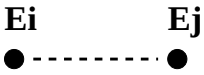

Figure 2. *Notation for high bandwidth connection between E_i and E_j*

To see that this simple two-valued model of bandwidth requirements is sufficient for our case, we refer the reader to [NEV 00]. In that paper, a case-study investigating the communication overhead for a so-called *Secret Query Database* is given. In this application, A has a query q and B owns a database with records x. The Secret Query Database allows them to cooperate in such a way that they can compute $q(x)$ while A preserves the secrecy of q and B preserves that of x. The communication overhead to solve this concrete case with SDC protocols is in the order of magnitude of 100 megabytes. On the other hand, sending just the query data, or sending an agent containing the query requires only a few kilobytes of communication. The large difference in amount of communication shows that our simplified model of high and low bandwidth requirements is realistic.

Based on these simple models of communication and trust, we compare the three options for implementing secure distributed computations.

3.1. *A Trusted Third Party*

The first, perhaps most straightforward option, is to use a globally trusted third party. Every p_i sends its private input x_i to the TTP who will compute $f(x_1, ..., x_n)$ and disseminate the result to the participants $p_i, i = 1..n$.

Of course, before sending its private data to the TTP, every p_i must first authenticate the TTP, and then send x_i through a safe channel. This can be accomplished via conventional cryptographic techniques.

It is clear that this approach has a very low communication overhead: the data is only sent once to the TTP; later, every participant receives the result of the computa-

tion. However, every participant should uncondionally trust the TTP. For the case of 4 participants, the situation is as shown in figure 3.

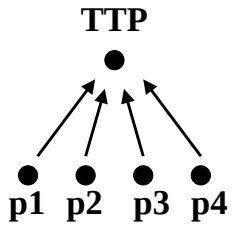

Figure 3. *Situation with 4 participants and a TTP.*

It is not clear whether n distrustful participants will easily agree on one single trustworthy execution site. This requirement of one single globally trusted execution site is the main disadvantage of this approach.

3.2. *Cryptographic Secure Distributed Computing*

The second option is the use of cryptographic techniques (as surveyed in section 2) that make the use of a TTP superfluous.

The trust requirements are really minimal: every participant p_i trusts its own execution site E_i, and expects that the other participants provide correct values for their own inputs.

Although this option is very attractive, it should be clear from the previous sections and from [NEV 00] that the communication overhead is far too high to be practically useful in a general networked environment. High bandwidth is required between all of the participants. For the case of 4 participants, the situation can be summarized as shown in figure 4.

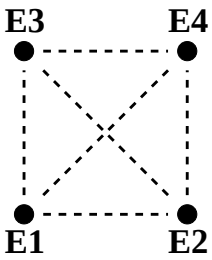

Figure 4. *Situation with 4 participants without a TTP.*

3.3. *A Virtual Trusted Third Party*

Finally, our solution tries to combine the two previous options: the communication overhead of SDC-techniques are remedied by introducing semi-trusted execution sites and mobile agents.

In this approach, every participant p_i sends its representative, agent a_i, to a trusted execution site E_j. The agent contains a copy of the private data x_i and is capable of running a SDC-protocol.

It is allowed that that different participants send their agents to different sites. The only restriction being that the sites should be located closely to each other, i.e. should have high bandwidth communication between them.

Of course, every execution site needs a mechanism to safely download an agent. However, that can be easily accomplished through convential cryptographic techniques.

The amount of large distance communication is moderate: every participant sends its agent to a remote site, and receives the result from its agent. The agents use a SDC-protocol, which unfortunately involves a high communication overhead. However, since the agents are executing on sites that are near each other, the overhead of the SDC-protocol is acceptable. For a situation with 4 participants, we could have the situation as depicted in figure 5.

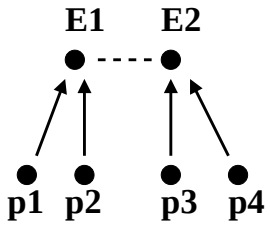

Figure 5. *Situation with 4 participants and a Virtual Trusted Third Party*

No high bandwidth communication between the participants is necessary, and there is no longer a need for one single trusted execution site. p_1 for example, does not need to trust site E_2. The agents that participate in the secure computation are protected against malicious behaviour of other (non-trusted) execution sites by the SDC-protocols. That is sufficient to make this approach work.

Moreover, in contrast with the approach where one uses an unconditionally trusted third party, the trusted sites are not involved directly. They simply offer a secure execution platform: the trusted hosts do *not* have to know the protocol used between the agents. In other words, the combination of mobile agent technology and secure distributed computing protocols makes it possible to use *generic* trusted third parties that, by offering a secure execution platform, can act as trusted third party for a wide variety of protocols in a uniform way.

Finally, the question remains whether it is realistic to assume that participants can find execution sites that are close enough to each other. Given the fact however that these execution sites can be *generic*, we believe that providing such execution sites could be a commercial occupation. Various deployment strategies are possible. Several service providers, each administering a set of geographically dispersed "secure hosts", can propose their subscribers an appropriate site for the secure computation. The site is chosen to be in the neighborhood of a secure site of the other service providers involved. Another approach is to have execution parks, offering high bandwidth communication facilities, were companies can install their proprietary "secure site". The park itself could be managed by a commercial or government agency.

4. Case study: second price auctions

In this example we consider the case of a second price auction, where there is one item for sale and there are n bidders. The item will only be sold if the bid of one participant is strictly higher than the other bids. In all other cases there is no winner. The clearing price is the second highest bid. The requirements for this type of auction are the following:

– if there is no winner, do not reveal anything;

– if there is a winner:

- reveal the identity of the highest bidder, but hide the highest bid;
- reveal the 2^{nd} highest bid, but hide the identity of the 2^{nd} highest bidder;
- do not reveal any other information.

Our goal is to estimate the communication overhead of an implementation of secure distributed second price auctions with the protocol proposed by Franklin and Haber (section 2.4). The auction is designed as a boolean circuit and the communication overhead for secure circuit evaluation is estimated. The communication overhead is determined by the following steps in the protocol:

– broadcast of the encrypted input bits of each participant;

– evaluation of an AND gate:

- broadcast of the encrypted bits $E(b_i)$, $E(c_i)$;
- broadcast of the decryption witnesses $W_i(\hat{u}')$, $W_i(\hat{v}')$;
- broadcast of the blinded $\hat{w}_i$;

– broadcast of the output decryption witnesses.

The associated communication overhead is:

– $2 \cdot |N| \cdot in_i \cdot n$ for the broadcast of the input bits;

– $8 \cdot |N| \cdot n$ for the evaluation of an AND gate;

– $|N| \cdot out \cdot n$ for the decryption broadcast.

n	4	16	32
Overhead (MB)	15	1000	8000

Table 1. *Network overhead of secure second price auctions*

where $|N|$ is the length of N in bits, which is the same as the number of bits needed to represent an element of Z_N^*, in_i is the number of input bits of participant i, n is the number of participants and out is the number of output bits of the circuit. In order to estimate the communication overhead, we need to be able to determine the number of AND gates in the boolean circuit (remember that each OR gate can be implemented with AND and NOT gates).

For three participants X, Y and Z, the boolean circuit is shown in figure 6. The inputs to the circuit are 32-bit bids. The output is the identity of the winner, represented by the bits $R1$ and $R0$ ($R1R0 = 00$ no winner, 01 winner is X, 10 winner is Y, 11 winner is Z), and the clearing price. If there is no winner, the clearing price is set to zero. To determine the winner, the circuit uses three comparators and a number of AND and OR gates. To determine the clearing price, four multiplexers are used. Consider the situation where X makes the highest bid. In this case $G1 \wedge G2 = 1$, $L1 \wedge G3 = 0$, $L2 \wedge L3 = 0$ and $R1R0 = 10$, so the second input to the final multiplexer will be chosen. The input on this line is determined by the bids made by Y and Z. If $Y > Z$ then $G3 = 1$ and Y will be selected as the clearing price. In the other cases ($Y < Z$ or $Y = Z$) Z will be the clearing price.

For $n > 3$ participants, the circuit changes as follows. The number of comparators needed is now $\binom{n}{2} = n \cdot (n-1)/2$ and each comparator has 434 AND gates. The final multiplexer will need to distinguish between $n+1$ different cases, i.e. n possible winners or no winner at all. The other n multiplexers are there to select the clearing price out of $n-1$ bids when there is a winner. The number of AND gates needed for each multiplexer as a function of the number of inputs m is shown in figure 7. Besides the comparators and the multiplexers, some additional AND and OR gates are needed. However, the number of these gates is negligible compared to the number of gates needed for the comparators and multiplexers. In summary, the circuit has a total gate complexity of $O(n^2)$.

The results of estimating the communication overhead for this circuit as a function of the number of participants n are summarized in table 1[1]. Franklin and Haber's protocol is linear in the number of broadcasts, so the total message complexity is $O(n^3)$. However, it must be noted that this only holds on a network with broadcast or multicast functionality, such that the communication overhead of sending a message to all participants is the same as that of sending a message to a single participant. In absence of such infrastructure, the total message complexity is $O(n^4)$.

1. We choose $|N|$ to be 1024 bits.

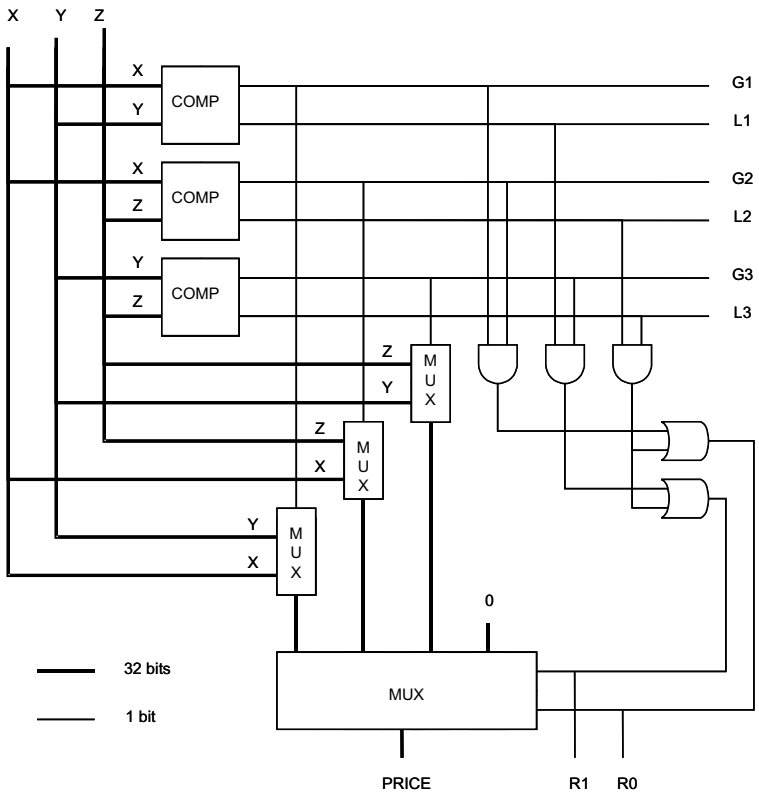

Figure 6. *Boolean circuit implementation of second price auctions.*

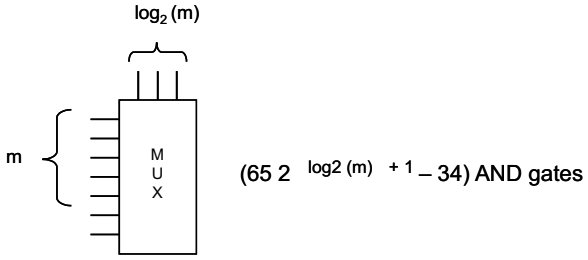

Figure 7. *Number of AND gates needed in a mulitplexer*

The figures in table 1 show that the mobile agent approach for implementing SDC protocols makes sense: the communication overhead is feasible on a LAN, but prohibitive over the Internet.

5. Conclusion

This paper shows how the use of semi-trusted hosts and mobile agents can provide for a trade-off between communication overhead and trust in secure distributed computing. There is no need for one generally trusted site, nor does the program code have to be endorsed by all participants. The trusted execution sites are generic and can be used for a wide variety of of applications. The communication overhead of secure distributed computing protocols is no longer prohibitive for their use since the execution sites are located closely to each other.

6. Bibliography

[ABA 90] M. ABADI, J. FEIGENBAUM, "Secure circuit evaluation, a protocol based on hiding information from an oracle," Journal of Cryptology, 2(1), p. 1–12, 1990

[CAC 00] C. CACHIN, J. CAMENISCH, J. KILIAN, J. MÜLLER, "One round Secure Computation and Secure Autonomous Mobile Agents", submitted to ICALP 2000.

[CHA 87] D. CHAUM, I. DAMGÅRD, 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

[CHO 95] B. CHOR, O. GOLDREICH, E. KUSHILEVITZ, M. SUDAN, "Private information retrieval," Proc. of 36th IEEE Conference on the Foundations of Computer Science (FOCS), p. 41–50, 1995

[CRA 99] R. CRAMER. "An introduction to secure computation", in LNCS 1561, pp 16–62, 1999.

[EVE 85] S. EVEN, O. GOLDREICH, A. LEMPEL. "A randomized protocol for signing contracts." Communications of the ACM, vol. 28, 1985, pp. 637–647.

[FRA 93] M. FRANKLIN, "Complexity and security of distributed protocols," Ph. D. thesis, Computer Science Department of Columbia University, New York, 1993

[FRA 96] M. FRANKLIN AND S. HABER, "Joint encryption and message-efficient secure computation," Journal of Cryptology, 9(4), p. 217–232, Autumn 1996

[GOL 87] O. GOLDREICH, S. MICALI, A. WIGDERSON, "How to play any mental game," Proc. of 19th ACM Symposium on Theory of Computing (STOC), p. 218–229, 1987

[GOL 87b] O. GOLDREICH, R. VAINISH. "How to solve any protocol problem: an efficiency improvement", Proceedings of Crypto'87, LNCS 293, pp. 73–86, Springer Verlag, 1987

[LOU 99] S. LOUREIRO, R. MOLVA, "Privacy for Mobile Code", Proceedings of the workshop on *Distributed Object Security*, OOPSLA '99, p. 37–42.

[NEV 00] G. NEVEN, F. PIESSENS, B. DE DECKER, "On the Practical Feasibility of Secure Distributed Computing: a Case Study", Information Security for Global Information Infras-

tructures (S. Qing and J. Eloff, eds.), Kluwer Academic Publishers, 2000, pp. 361-370.

[NIS 99] 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.

[SAN 98] T. SANDER, C. TSCHUDIN, "On software protection via function hiding", *Proceedings of the second workshop on Information Hiding*, Portland, Oregon, USA, April 1998.

[SAN 98b] T. SANDER, C. TSCHUDIN, "Towards mobile cryptography", *Proceedings of the 1998 IEEE Symposium on Security and Privacy*, Oakland, California, May 1998.

[YUN 00] T. SANDER, A. YOUNG, M. YUNG, "Non-Interactive CryptoComputing for NC^1", preprint.