On the trade-off between communication and trust in secure computations

Frank Piessens[*], Bart De Decker, Erik Van Hoeymissen, Gregory Neven
Dept. of Computer Science, K.U.Leuven
Celestijnenlaan 200A, B-3001 Heverlee, Belgium
E-mail:{frank,bart,erikv,gneven}@cs.kuleuven.ac.be

Note: regular submission for DISC2000, eligible for best student paper.

Abstract

In this paper, we show that mobile code technology may prove to be a useful tool in advanced cryptographic protocols for secure distributed computing.

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. In the presence of a third party completely trusted by all participants, the problem is trivially solvable. However, this assumption is in many applications non-realistic. 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.

We will show in this paper how the use of mobile agents employing these cryptographic techniques can provide for a trade-off between communication overhead and trust. The communication overhead problem would be solved if the communicating parties were brought close enough together. Our solution is to use mobile agents to execute the cryptographic protocols. Of course, a mobile agent needs to trust his execution platform, but we show that the level of trust needed in this case is much less than the level of trust needed for a classical trusted third party.

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

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 cryptographic 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 ([12]), charging for the use of algorithms on the basis of a usage count ([13, 14]), various kinds of weighted voting, protecting mobile code integrity and privacy ([14, 10]), etc...

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 cryptographic solutions has been proposed. An overview is given by Franklin [6], and more recently by Cramer [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). 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 [11].

Mobile agents employing these cryptographic techniques can provide for a trade-off between communication overhead and trust. The communication overhead problem would be solved if the communicating parties were brought close enough together. Our solution is to use mobile agents to execute the cryptographic protocols. Of course, a mobile agent needs to trust his execution platform, but we show that the level of trust needed in this case is much less than the level of trust needed with a classical TTP. 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 code 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 ([13, 14]). 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. First, we present a short survey on cryptographic protocols for the SDC problem, since these protocols are not widely known in the distributed systems community. Moreover, the survey gives an indication of the amount of communication these protocols require. Then, we discuss how code mobility can offer a middleground between a TTP-based solution and a purely cryptographic solution. Finally, we conclude by summarizing the advantages of this middleground solution.

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). Since these solutions are not well-known in the distributed systems community, we present a short survey.

2.1 Using probabilistic encryption

One class of techniques for SDC 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 **op** and **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 (i.e. circuit) 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 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 [11].

2.2 Protocols based on oblivious transfer

In [8], 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

$$\begin{gathered} < E(R \cdot r_o^0, r_l^0 \oplus r_r^0), E(R \cdot r_o^0, r_l^0 \oplus r_r^1), \\ E(R \cdot r_o^0, r_l^1 \oplus r_r^0), E(R \cdot r_o^1, r_l^1 \oplus r_r^1) > \end{gathered}$$

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* ([5]). 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 2-party secure computation based on oblivious transfer is presented in [9]. 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 ([14]). 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 ([13, 14]) 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. They consider the example of implementing a digital signing primitive for mobile agents. Even if the real signature routine can be kept secret, the whole (encrypted but operational) routine might be abused to sign arbitrary documents. Thus, it is necessary 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 [14].

Loureiro and Molva ([10]) use a public key encryption system based on Goppa codes that allows for the non-interactive 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, in a very recent paper ([15]), Sander, Young and Yung propose an autonomous protocol based on a new homomorphic encryption scheme.

2.4 Multiparty protocols

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.

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 has 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 [7]. All parties send each other an ElGamal 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.

Even information-theoretically secure multiparty computation can be achieved (as opposed to only computationally secure). A possible realisation is discussed in [4].

The communication overhead for multiparty protocols is even more serious than that for the 2-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 tradeoff 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 trusted third party 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.

3.1 A Trusted Third Party

The first, perhaps most straightforward option, is to use a trusted third party (TTP). 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 $p_i, i = 1..n$.

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 computation. However, every participant should uncondionally trust the TTP. What does that trust involve? First of all, every p_i trusts:

- that the TTP will correctly compute $f(x_1, ..., x_n)$;
- that the TTP will **not** conspire with one of the other parties and leak private inputs x_i.

In other words, every partner trusts the code of the TTP. Although not trivial, this could be accomplished through code-inspection, and the use of a trusted (chain of) compiler(s).

However, this is not sufficient: the execution platform on which the TTP is executing, should also be considered. Hence, every participant p_i needs to be convinced that:

- the TTP is present on that execution platform;
- the execution platform excecutes the code of the TTP correctly;
- the execution platform does not spy on the processes, (e.g. it does not copy the private data x_i of the TTP for direct or later misuse);
- the execution platform destroys the TTP after execution (there are no traces of the data on the TTP left).

It is not clear whether n distrustful participants will easily agree on a trustworthy execution site, and what procedures should be used.

Finally, 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.

To sum up, this approach has a very low communication overhead, but requires extensive trust relations.

3.2 Cryptographic Secure Distributed Computing

The second option is the use of cryptographic techniques that make the use of a trusted third party superfluous.

The trust requirements are really minimal: every participant p_i trusts its own execution site, 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 [11] that the communication overhead is far too high to be practically useful in a general networked environment.

3.3 A Virtual Trusted Third Party

The last option tries to combine the two previous options: the communication overhead of SDC-techniques are remedied by introducing semi-trusted execution sites.

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 different participants send their agents to different sites, the only restriction being that the sites should be located closely to each other.

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.

Let us examine the trust requirements for this approach.

Every p_i sends his own a_i to a remote execution site E_j. Hence, p_i can trust its agent. Furthermore, p_i needs to be convinced that the excecution base E_j on which its agent a_i runs, is trustworthy:

- E_j correctly executes the code of a_i;
- E_j does not spy on a_i;
- E_j destroys all traces of a_i after its execution.

These are the same requirements as those for the trusted site of the first option. The difference, however, is that not all the participants have to endorse the same site. Moreover, in contrast with protocols that use unconditionally trusted third parties, the trusted site is not involved directly. It simply offers a secure execution platform for the mobile agents: 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* 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.

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

Note that a p_i does not need to trust the other execution sites. The agents that participate in the secure computation are protected against malicious behaviour of the other execution sites by the SDC-protocols. That is sufficient to make this approach work.

4 Conclusion

We briefly summarize the advantages of the combination of mobile code and cryptographic SDC-techniques.

- It is not necessary to find one site trusted by all participants. Every participant only trusts the execution site of his agent.
- There is no need for computation specific code that has to be endorsed by all participants. The trusted execution sites are generic, and can be reused for a wide variety of applications.
- Since the execution sites are located closely to each other, the communication overhead of SDC-protocols are no longer prohibitive for their use.

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] R. Cramer. "An introduction to secure computation", in LNCS 1561, pp 16–62, 1999.

[5] S. Even, O. Goldreich, A. Lempel. "A randomized protocol for signing contracts." Communications of the ACM, vol. 28, 1985, pp. 637–647.

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

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

[8] 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

[9] 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

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

[11] G. Neven, F. Piessens, B. De Decker, "On the Practical Feasibility of Secure Distributed Computing: a Case Study", submitted to WCC2000.

[12] 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.

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

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

[15] T. Sander, A. Young, M. Yung, "Non-Interactive CryptoComputing for NC^1", preprint.