On solving distributed CS(O)Ps with privacy[1]

Technical Report No. IC/2002/55

Marius Călin Silaghi

Laboratoire d'Intelligence Artificielle
Faculté Informatique et Communications
EPFL, Ecublens
CH-1015 Lausanne

E-mail : silaghi@lia.di.epfl.ch

Fax: ++41-21-693.52.25

July, 2002

Cryptographic protocols can enforce privacy in distributed computation of functions [GB96] and are a competitor of the distributed constructive search techniques. [GMW87, CCD88b, CCD88a, BOGW88] show how cryptographic protocols can be compiled from protocols/functions for honest agents. For some combinations of concepts of security and types of attacks, cryptographic protocols obtained this way can be safe. We discuss their application to constraint satisfaction (and optimization) problems.

[1] A first version of this report, with some notation problems, appears in the PhD thesis report [Sil02].

1 Introduction

Cryptographic protocols can enforce privacy in distributed computation of functions [GB96] and are a competitor of the distributed constructive search techniques. [GMW87, CCD88b, CCD88a, BOGW88] show how cryptographic protocols can be compiled from protocols/functions for honest agents. For some combinations of concepts of security and types of attacks, cryptographic protocols obtained this way can be safe.

Example 1.1 *Without intractability assumptions, when at most a minority of the participants (<1/3) can make coalition, there exist cryptographic protocols that are safe even against active attacks [CCD88a].*

Nevertheless, with most cryptographic protocols, if the assumptions do not hold, the attacker that does not belong to prevented types of attackers can discover everything easily.

Remark 1.1 *Typically, at the end of a computation, no agent can know whether anything of its privacy is saved.*

Many existing cryptographic protocols are targeted to the computation of functions based on addition and multiplication, but there exist other secure protocols for problems like elections and user identification. Constraint satisfaction and optimization problems (CSOPs) are efficiently approached with versions of backtracking, but these show to be difficult to implement securely.

2 Why DisCSPs and Secure Protocols?

You will surely ask:

1 *Why discussing cryptographic protocols for DisCSPs?*

The question is reasonable since usually cryptographic protocols are simple synchronous translations of centralized techniques. Actually an important motivation to work on DisCSPs is the belief that such frameworks are useful for privacy in negotiations [SSHF00, MJ00, SSHCF01].

3 Cryptographic Protocols for multi-party computation

Cryptographic protocols are fascinating through their original properties. Interactive proofs and zero knowledge protocols, are one type of such cryptographic protocols [GMR85, GMR89, Bab85]. The interaction proofs are more powerful than written proofs. The user can be asked to tell an *unexpected* half of a transformed proof and when he is able to answer, the chances that he is lying are reduced by 50%. Without revealing the proof, a series of such interactions reduces exponentially the chances of accepting a liar.

Setting	Adv. Type	t-privacy	Literature
cryptographic	passive	$t < n$	[GMW87]
cryptographic	active	$t < n/2$	[GMW87]
information-theoretic	passive	$t < n/2$	[CCD88b, CCD88a, BOGW88]
information-theoretic	active	$t < n/3$	[CCD88b, CCD88a, BOGW88]
i.-t. with broadcast	active	$t < n/2$	[RBO89, Bea91]

Figure 1: Some available results on secure computations.

Secure multi-party computations are introduced in [Yao82]. The main motivation behind them seems to be the fact that one system is easier to break than a set of systems. Imagine that a cryptographic key is stored on a computer system. Once an attacker breaks the system, the key is lost.

4 Secure multi-party computation

The idea in multi-party computations is to break the key into n pieces and to distribute these pieces to a set of n distinct systems (agents). The key cannot be reconstructed except if a subset of k agents, $k \leq n$, put their pieces together. The new version of the system is robust against the attackers that cannot corrupt at least k agents.

Two simple ways of splitting a number are:

- n points of a k degree polynomial that intersects the y-axis at an ordinate equal to the secret number [Sha79].
- decomposition based on probabilistically transferred bits [CGMA85].

The second method allows for verifying the fact that the share of an agent has not been corrupted (modified) with less than O(log n) colluders. One of the first methods seems to be the one proposed in [Bla79].

Once numbers are split and distributed, the idea [Yao82] is that distributed computations of an arbitrary agreed function can be performed over the distributed shares, in such a way that all results remain shared secrets with the same security properties (the number of supported colluders, $k-1$). For [Sha79]'s secret sharing, the computation of the sum function is efficiently implemented as a sum of the shares for corresponding samples. Multiplication is more complex and recent research has focused on improving its efficiency [HMP00].

Functions can be *compiled* unto secure cryptographic protocols. When a certain function has to be computed on shared secrets, all its steps can be performed using secure protocols for the corresponding steps. This can be done in such a way that any attacker that cannot find the initial secrets, cannot find anything else than the *official output* of the protocol [GB96].

Techniques for compiling protocols for honest parties when less than n/2 players cheat, and that are based on intractability assumptions, are proposed

in [GMW86, GMW87]. [CCD88b, CCD88a, BOGW88] show compiling techniques for cases where less than n/3 players cheat and that are secure without intractability assumptions. [BOGW88] details a quite efficient technique for computing polynoms with less than n/2 cheaters, when all participants comply with the protocol. A compilation of available results on multi-party computation by Thomas Duebendorfer is shown in Figure 1.

One can see cryptographic protocols as a way of replacing one trusted party with a community performing the same operations but in a secure manner.

5 Cryptographic definitions for Privacy

A clear distinction has to be made between cryptographic security and information-theoretic security [CCD88a, BOGW88].

Definition 5.1 ([CCD88a]) *An input value, x_i, is* cryptographically secure *if gaining information about it, other than that obtained from the wished outcome, z, is thought to be computationally hard.*

The information-theoretic approach (non-Cryptographic) does not limit the computational power of the processors [BOGW88]:

Definition 5.2 ([CCD88a]) *An input value, x_i, is* unconditionally secure *if gaining information about x_i is impossible beyond that available from the wished outcome, z.*

Most multi-party secure protocols impose some kind of restrictions to the security. In the previously mentioned approaches, these restrictions are either computational-based, or a limit on the number of supported colluders, or both.

Some nice formal definitions of privacy that illustrate the restrictions on the number of supported colluders appear in [BOGW88]. When Q_i is the set of states and F the set of possible messages, $I_i : Q_i \rightarrow F$ is the input function for player i. [BOGW88] denote $\prod_{i \in C} Q_i$ by Q_C. A vector $\langle a_0, a_1, ..., a_{n-1} \rangle$ is denoted by $\langle a_i \rangle$. By u_C we denote the sub-vector of u, $u = \langle u_0, u_1, ..., u_{n-1} \rangle$ that contains $u_i, i \in C$. Consider that protocols consist of rounds $j = 1, 2, ..., T$ and that A is an arbitrary set. $q_i^{(j)}$ denotes the state of player i after round j.

Definition 5.3 ([BOGW88]) *A set C is* ignorant *in a protocol δ (for computing f), if for every set of initial states $\langle q_i^{(0)} \rangle$, every protocol δ_C that looks like δ and every function $g' : Q_C \rightarrow A$ there exists a function $g : Q_C \times F^{|C|} \rightarrow A$ satisfying*

$$g'(q_C^{(T)}) = g(q_C^{(0)}, f(\langle I_i(q_i^{(0)}) \rangle)_C).$$

Definition 5.4 ([BOGW88]) *A protocol (for computing f) is* t-private *if every coalition C with $|C| \leq t$ is ignorant.*

The main results in the field are summarized in Figure 1, where t specifies the guaranteed t-privacy property.

Another property of secure protocols is their correctness as parametrized function on the possible Byzantine behavior of the participants. A coalition is *harmless* if any of its incorrect actions can only lead to their elimination from the problem.

Definition 5.5 ([BOGW88]) *A protocol is* t-resilient *if every coalition C with $|C| \leq t$ is harmless.*

[BOGW88] states that for every probabilistic function and every $t<n/3$ there exists a protocol that is both t-resilient and t-private while there are functions for which there is no $n/3$-resilient protocol.

6 Zero-knowledge

No presentation of cryptographic protocols is complete without the definition of zero-knowledge introduced in [GMR89]. Intuitively, a proof made by a prover to a verifier in an interactive system is zero-knowledge if a polynomially-bounded malicious verifier cannot learn anything else than the existence of the proof.

An interactive system is composed of two agents, A and B, where A wants to prove to B the existence of a certain property $x \in L$, without letting B learn anything about x that it cannot compute alone in polynomial (in x) time and resources. An interactive proof consists of a series of rounds where A makes a statement, B tosses a random coin for choosing a challenge and A has to answer the challenge. Any failure of A to answer a challenge decides the failure of the proof. A sufficiently long sequence of successful answers by A exponentially increases the confidence of B in the fact that A knows a proof. The protocol ends successfully when B is satisfied with its confidence. Such a protocol is denoted (A,B).

Goldwasser describes the interactive Turing machine (ITM). An interactive protocol can be formalized as an ordered pair of ITMs.

Two notions are needed to define zero knowledge. The *indistinguishability of random variables* is meant to compare the language defined by the messages of the prover.

- Two families of random variables are perfectly indistinguishable when they are equal.
- Two families of random variables are statically indistinguishable on L when no judge differentiate any random polynomially-bounded (in $|x|$) samples in L.
- Two families of random variables are computationally indistinguishable on L when no polynomially-time (in $|x|$) judge differentiate any random polynomially-bounded (in $|x|$) samples.

The *perfectly/statistically/computationally approximability of a random variable U(x)* on a language L checks that there exists a probabilistic Turing machine running in polynomial time such that its acceptance result-function on its input x is perfectly/statistically/computationally indistinguishable from $U(x)$. This notion is meant to quantify the knowledge gained by a verifier.

Definition 6.1 ([GMR89]) *Let $L \subset \{0,1\}*$ be a language and (A,B) a protocol. Let B' be a verifier trying to cheat using an extra input H bounded polynomially. (A,B) is perfectly/statistically/computationally zero-knowledge on L for B' if the family of random variables defining the final knowledge of B' is perfectly/statistically/computationally approximable for the language L' obtained by concatenating any element of L with H.*

(A,B) is perfectly/statistically/computationally zero-knowledge on L if it is perfectly/statistically/computationally zero-knowledge on L for all probabilistic polynomial time ITM B'.

7 Multi-party computations and DisCSPs

A classical approach for applying a secure multi-party computation to a problem consists in choosing a trusted party solution and compiling its steps [GMW87, CCD88a, BOGW88].

There exist generic compiling techniques for functions composed of additions and multiplications. I will show now how DisCSP techniques can be transformed to such functions that can be easily compiled. The DisCSP techniques that I address is: *Finding a solution to a (Dis)CSP.*

7.1 DisCSPs

We consider naturally distributed constraint satisfaction problems.

Definition 7.1 (DisCSP) *A* **DisCSP** *(A,V,D,C) is given by a set of agents $A_1, ..., A_n$ where each agent A_i wants to enforce some private constraint C_i in C. The set of shared variables involved in C_i is V_i (from V). The agents want instantiation of the variables in V_i in the corresponding domains D_i.*

Once the constraints of different agents are securely shared, securely solving the DisCSP reduces to securely solving the CSP obtained by the union of all constraints enforced by the different agents.

7.2 Secure Search of a First Solution

Imagine we want to solve a CSP $P = (X, C, D)$ where X is a set of variables $x_1, x_2, ...x_n$, C is a set of constraints and D a set of domains for X. The domain of x_i is D_i, whose values are $v_1^i, v_2^i, ..., v_{|D_i|}^i$.

Definition 7.2 (first solution) *The first solution of a CSP given a total order on its variables and a total order on its values is the first among solution tuples when these are ordered lexicographically.*

Let *gconsistent(P)* be a function:

$$gconsistent(P) = \begin{cases} 1 & \text{if } P \text{ has a solution} \\ 0 & \text{if } P \text{ is infeasible} \end{cases}$$

We will design now a set of functions: $f_1, f_2, ..., f_n$, $f_i : CSP \rightarrow \mathbb{N}$, such that each f_i will return the index of the value of x_i in the first solution, or 0 if no solution exists.

$$f_i(P) = \begin{cases} k & \text{if } P \text{ has the first solution for } x_i = v_k^i \\ 0 & \text{if } P \text{ has no solution} \end{cases}$$

Let us first design the functions $g_{i,1}, g_{i,2}, ..., g_{i,|D_i|}$. $g_{i,j} : CSP \rightarrow \{0,1\}$.

$$g_{i,j}(P) = \begin{cases} 1 & \text{if } P \text{ has a first solution for } x_i = v_j^i \\ 0 & \text{if } P \text{ is infeasible for } x_i = v_j^i \end{cases}$$

$$g_{i,j}(P) = gconsistent(P \cup \{x_i = v_j^i\} \cup_{k<i} (x_k = v_{f_k(P)}^k)) \quad (1)$$

$$f_j(P) = \sum_{i=1}^{|D_j|} i * (g_{j,i}(P) * \prod_{k<i} (1 - g_{j,k}(P))) \quad (2)$$

Lemma 7.1 *The functions g and f given by Equations 1 and 2 correspond to their definition.*

Proof. The properties can be checked recursively starting with g_1, k and f_1. □

It remains to find an efficient implementation of *gconsistent()*.

Lemma 7.2 *The existence of a polynomial implementation for gconsistent() would prove that $P = NP$*

Proof. Satisfiability of a CSP is NP-complete. □

Unfortunately we have not yet proven that NP=P and for the moment we give a solution with exponential cost.

Remark 7.1 *It is improbable that one will ever find secure protocols more efficient than generate and test. This is due to the fact that any trimming of a branch reveals information when the "test" operator cannot be implemented securely (see [Sil02]).*

Let $SS(P)$ be the ordered set of all tuples in the cross-product of the domains of P. Each constraint c in the set of constraints C is a function, $c : SS(P) \rightarrow \{0,1\}$. The secret parameters of the distributions are the various values $c(t)$ where t is a tuple. Let us define the function p, $p : SS(P) \rightarrow \{0,1\}$, defined as $p(t) = \prod_{c \in C} c(t)$.

$$gconsistent(P) = \sum_{t_i \in SS(P)} (p(t_i) \prod_{k<i} (1 - p(t_k)))$$

Proposition 7.1 *Given the previous definitions of the functions p, gconsistent(), $g_{i,j}$, and f_i, and a (Dis)CSP P, the vector $\langle v^i_{f_i(P)} \rangle$ defines a solution of P (the first one).*

Proof. See the definition of the functions f. □

Remark 7.2 *The computation of the vector $\langle f_i(P) \rangle$ requires only additions and multiplications and can be easily compiled unto a secure protocol using any of the classic techniques mentioned in this chapter.*

The secret parameters of the computation are the values c(t).

Remark 7.3 *Actually whenever an element of the vector $\langle f_i(P) \rangle$ is 0, the computation can be stopped since P is infeasible.*

The secure algorithm obtained by compiling the computation of $\langle v^i_{f_i(P)} \rangle$ is referred to as Secure CSP Solver (SCSPS). To circumvent the exponential number of intermediary results and to have an acceptable space requirement, the computation of *gconsistent* should be done tuple after tuple.

Theorem 7.2 *Submitted shares of an input value, v, of a tuple in a constraint can be verified by checking that $v(v-1)=0$.*

Proof. This proves that v is either 0 or 1. □

8 Secure CSOP Solver (SCSOPS)

Definition 8.1 (CSOP) *A constraint satisfaction and optimization problem (X, D, F) is defined by a set of variables $X = \{x_1, x_2, ..., x_n\}$ and a set of functions $f_i \in F$, $f_i : D^1_i \times D^2_i \times ... \rightarrow \mathbb{N}$, $D^k_i \in D$ is the domain of a variable $x^k_i \in X$.*

The problem is to find a valuation t of all variables in X, that maximizes $\sum_i f_i(t)$.

Definition 8.2 (DisCSOP) *A distributed CSOP (A, X, D, F) is defined by a set of agents, each of them owning a private CSOP with eventual shared variables, and looking for a solution that maximizes the CSOP (X, D, F) obtained by the union of the variables and constraints in all the agents.*

function value-to-differential-bid(c_t, K)

1. Jointly, all agents build a vector a_t^0 for the secret value c_t.
   $a_t^0 =< c_{t,0}^0, c_{t,1}^0, ..., c_{t,K}^0 >$. Actually,
   $a_t^0 =< c_t, c_t - 1, c_t - 2, ..., c_t - K >$.
   To achieve this, each agent A_i computes
   $a_t^0(i) =< c_{t,0}^0(i), c_{t,1}^0(i), ..., c_{t,K}^0(i) >$. $c_{t,0}^0(i) = c_t(i)$, and for $k > 0$, $c_{t,k}^0(i) = c_{t,k-1}^0(i) - s(i)$.

2. Jointly, all agents build a vector a_t^1 for each possible tuple t.
   $a_t^1 =< c_{t,0}^1, c_{t,1}^1, c_{t,2}^1, ..., c_{t,K}^1 >$.
   $c_{t,k}^1 = (c_{t,k}^0 + 1) * \prod_{0<h\leq K} (c_{t,k}^0 - h)(c_{t,k}^0 + h)$.

3. Return a_t^1.

Figure 2: Transforming a secret value $c_t \in \{1, 2, ..., K\}$ to a differential bid. The share of c_t to the agent A_i is $c_t(i)$.

8.1 Intuitive Description

Now I describe a protocol for securely solving a DisCSOP (A, X, D, F) where all the functions in F return results in the set $\{0, 1, ..., h\}$. Many other situations, like negative values, can be mapped to this case. Let $m = h|F|$. The main steps of the SCSOPS are as follows.

1. Share constraint tuples of the problem of each agent.
2. Verify the value, v, of each submitted constraint tuple by checking that $v(v-1)...(v-h)=0$.
3. Securely compute the value of each tuple of the global problem, using the existing shares.
4. Make values into differential bids with the non-zero term y, $y = (-1)^m(m!)^2$.
5. Apply any of the known standard secure protocols to determine the winner differential bid.

9 Detailed Protocol

Assume a DisCSOP with n agents. By s(i) we denote i's share of the secret '1'. We say that a tuple t_i is in a larger tuple t when the projection of t on the variables of t_i yields t_i.

1. The agents generate secret shares for the value '1', s(i).
2. Each agent A_i generates secretly a value v^{t_i} for each tuple t_i in CSOP(A_i).

3. According to Shamir's scheme, each agent A_i generates secretly a polynome for each secret value v^{t_i}, and generates n secret shares, $v^{t_i}_{(i,j)}$, $1 \leq j \leq n$.

4. Each agent A_i sends to A_j the share $v^{t_i}_{(i,j)}$.

5. Verify that $v^{t_i}(v^{t_i}-1)...(v^{t_i}-h)=0$, for each submitted value v^{t_i}. Eliminate agents that lie.

6. All agents compute for each tuple t the global cost $c_t = \sum_{\forall i, t_i; t_i \in t} v^{t_i}$.
   To achieve this, each agent A_i computes $c_t(i) = \sum_{\forall k, t_k \in t} v^{t_k}_{(k,i)}$.

7. Jointly the agents perform a_t=value-to-differential-bid(c_t,m) and verify with one of the standard techniques that the result is a correct differential bid with the non-zero element y, $y=(-1)^m(m!)^2$.

8. Run a standard secure protocol for deciding the winner among a_t for all t, considered as differential bids.

9. The results for the agents are revealed and the winner tuple with its value are found (as in the other standard protocols).

10 Summary

In this article we first describe secure multi-party computations. We then show how classic secure protocols (for addition and multiplication) can be applied to DisCS(O)Ps.

References

[Bab85] L. Babai. Trading group theory for randomness. In *Proc. of 17th STOC*, pages 421–429, 1985.

[Bea91] D. Beaver. Secure multiparty protocols and zero-knowledge proof systems tolerating a faulty minority. In *Journal of Cryptology*, pages 75–122, 1991.

[Bla79] G.R. Blakley. Safeguarding cryptographic keys. In *Proc AFIPS 1979 NCC*, volume 48, pages 313–317, Arlington, Va, June 1979.

[BOGW88] M. Ben-Or, S. Goldwasser, and A. Widgerson. Completeness theorems for non-cryptographic fault-tolerant distributed computating. In *Proc. 20th ACM Symposium on the Theory of Computing (STOC)*, pages 1–10, 1988.

[CCD88a] D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols. In *Proc. 20th ACM Symposium on the Theory of Computing (STOC)*, pages 11–19, Chicago, 1988.

[CCD88b] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols. In Springer-Verlag, editor, *Proc. CRYPTO 87, LNCS 293*, page 462, 1988.

[CGMA85] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *Proceedings of the 26th IEEE Symposium on Foundations of Computer Science*, pages 383–395, Portland, 1985.

[GB96] Shafi Goldwasser and Mihir Bellare. Lecture notes on cryptography. MIT, July 1996.

[GMR85] S. Goldwasser, S. Micali, and C. Rackoff. Knowledge complexity of interactive proofs. In *Proc. of 17th STOC*, pages 291–304, 1985. Earlier version: Knowledge complexity, unpublished manuscript, (submitted to FOCS, 1984).

[GMR89] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18:186–208, 1989.

[GMW86] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design. In *Proc. of 27th IEEE Symp. on Foundations of Comp. Science*, pages 174–187, Toronto, 1986.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game — a completeness theorem for protocols with honest majority. In *Proc. 19th ACM Symposium on the Theory of Computing (STOC)*, pages 218–229, 1987.

[HMP00] Martin Hirt, Ueli Maurer, and Bartosz Przydatek. Efficient secure multi-party computation. In *Advances in Cryptology - ASIACRYPT'00*, volume 1976 of *LNCS*, pages 143–161, Dec 2000.

[MJ00] P. Meseguer and M. A. Jiménez. Distributed forward checking. In *CP DCS Workshop*, 2000.

[RBO89] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *appeared*, pages 73–85, 1989.

[Sha79] A. Shamir. How to share a secret. *Communications of the ACM*, 22:612–613, Nov 1979.

[Sil02] Marius-Călin Silaghi. *Asynchronously Solving Distributed Problems with Privacy Requirements.* 2601, Swiss Federal Institute of Technology (EPFL), CH-1015 Ecublens, June 27, 2002.

[SSHCF01] M.-C. Silaghi, D. Sam-Haroud, M. Calisti, and B. Faltings. Generalized English Auctions by relaxation in dynamic distributed CSPs with private constraints. In *Proc. of the IJCAI-01 DCR Workshop*, pages 45–54, Seattle, August 2001. submitted to AA'2001.

[SSHF00] M.-C. Silaghi, D. Sam-Haroud, and B. Faltings. Asynchronous search with private constraints. In *Proc. of AA2000*, pages 177–178, Barcelona, June 2000.

[Yao82] A.C. Yao. Protocols for secure computations. In *Proceedings of 23rd IEEE Symposium on the Foundations of Computer Science (FOCS)*, pages 160–164, 1982.