Improving Arithmetic Circuits for Solving Constraint Satisfaction and Optimization Problems

Marius C. Silaghi
Florida Institute of Technology

October 12, 2004

Technical Report CS-2004-14

Abstract

In the recent years we have proposed a set of secure multiparty computations for solving distributed constraint satisfaction and optimization problems, with applications to areas like distributed scheduling, configuration, team-making, and auctions. Some of the newest versions were based on an arithmetic circuit for selecting a random element out of the elements with a given value in a (secret) array. Here we show how to improve that arithmetic circuit by a factor of at least 4. The improvement is based on an optimization of the functions and on the usage of CSP solvers to exploit public constraints.

x1 \ x2	T	W
P	1	0
Q	0	1

Figure 1: Constraint: 0s mark rejected tuples.

1 Introduction

In this article we propose techniques to solve distributed problems with constraints that are secret to agents. The main applications are: auctions, distributed scheduling, configuration, and team-making.

CSP A *constraint satisfaction problem* (CSP) is defined by three sets: (X, D, C). $X = \{x_1, ..., x_m\}$ is a set of variables and $D = \{D_1, ..., D_m\}$ is a set of domains such that x_i can take values only from $D_i = \{v^i_1, ..., v^i_{d_i}\}$. $C = \{\phi_1, ..., \phi_c\}$ is a set of constraints, ϕ_i involving an ordered subset $X_i = \{x_{i_1}, ..., x_{i_{k_i}}\}$ of the variables in X, $X_i \subseteq X$, and constrains the legality of each combination of assignments to the variables in X_i. An assignment is a pair $\langle x_i, v^i_k \rangle$ meaning that variable x_i is assigned the value v^i_k.

A tuple is an ordered set. The projection of a tuple ϵ of assignments over a tuple of variables X_i is denoted $\epsilon_{|X_i}$. A solution of a CSP (X,D,C) is a tuple of assignments ϵ with one assignment for each variable in X such that each $\phi_i \in C$ is satisfied by $\epsilon_{|X_i}$.

For applications like meeting scheduling, each agent has his own private constraints and one has to also find an agreement for a solution, from the set of possible meeting places and dates, that satisfies everybody. Distributed constraint satisfaction is a handy formulation that can model these issues.

Definition 1 *A Distributed CSP (DisCSP) is defined by five sets (A, X, D, C, O). A=$\{A_1, ..., A_n\}$ is a set of agents. X, D, C and the solution are defined like in CSPs. Each constraint ϕ_i is known only by one agent, being the secret of that agent. There may exist a public constraint in C, ϕ_0.*

Example 1 *Alice (A_1), Bob (A_2), and Carol (A_3) want to find a common place (x_1) and time (x_2) to meet. x_1 is either Paris (P) or Quebec (Q), i.e. $D_1 = \{P, Q\}$. x_2 is either Tuesday (T) or Wednesday (W), i.e. $D_2 = \{T, W\}$. Each of them has a secret constraint. Alice accepts only $\{(P, T), (P, W), (Q, W)\}$ which defines ϕ_1. Bob accepts either of $\{(P, T), (Q, T), (Q, W)\}$, defined by ϕ_2. Carol has $\phi_3 = \{(P, T), (Q, W)\}$. ϕ_3 is shown in Figure 1. There is also a publicly known constraint, ϕ_0, which due to an announced strike forbids a meeting in Paris on Wednesday, $\phi_0 = \{(P, T), (Q, T), (Q, W)\}$. The problem is to publish values for x_1 and x_2 satisfying all constraints and without revealing anything else to Alice about ϕ_2 and ϕ3, to Bob about ϕ_1 and ϕ_3, or to Carol about ϕ_1 and ϕ_2.*

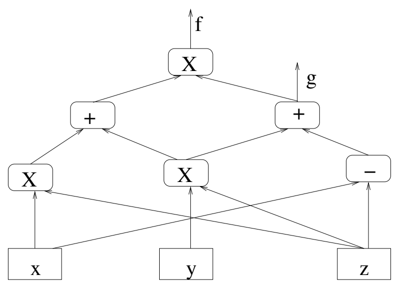

Figure 2: An arithmetic circuit, $g = yz + (x - z)$ and $f{=}(xz + yz)g$. Each input can be the secret of some participant. The output may not be revealed to all participants. All intermediary values remain secret to everybody.

Arithmetic Circuit The arithmetic circuits are a class of functions that can be evaluated securely and are exploited in our technique [1]. An arithmetic circuit is a function f, using solely the addition/subtraction and multiplication operations of a finite set $F = [0..(\nu{-}1)]$ modulus a prime number ν (Z_ν), $f : F^i \to F^j$ (see Figure 2). An arithmetic circuit can be intuitively imagined as a directed graph without cycles where each node is described either by an addition/subtraction or by a multiplication operator. Each source node is a (public or secret) constant. The only outputs of the circuit are the sinks of its graph (anything else can remain secret). $\sum_{i=b}^{e} f(i)$ and $\prod_{i=b}^{e} f(i)$ are arithmetic circuits if b and e are public constants and $f(i)$ is an arithmetic circuit.

Intuition Consider a constraint in its multidimensional matrix representation, where each element restricts the compatibility of some values for distinct variables. Each element encoded as 0 (forbidden) or 1 (possible) is encrypted with a shared key (it can be decrypted only when the majority of the agents agree). One can perform additions and multiplications of such values, while they are encrypted.

The agents cooperate to generate a secret permutation of the encrypted problem parameters, that cannot be manipulated by any of them. To avoid that agents get a chance to learn the final permutation by matching final encrypted parameter values with the ones they generated, a randomization step is applied at each shuffling. Each agent applies a randomization step on the encryption for each secret tuple acceptance/rejection encoding. Because the secrets are encrypted, this randomization step exploits the homomorphic properties of some encryption schemes.

We also give a fix (exponential) set of additions and multiplications that, applied on the constraints encrypted in the aforementioned way, returns the encrypted assignments in a solution picked according to a uniform distribution over the set of possible solutions. The agents may show now their share of the keys for the assignments in the solution. Each agent learns only the assignments of interest to him.

Complexity To hide the secret parameters of the problem, the distributed computation must not depend on those parameters. Since the problem is NP-complete, an algorithm that does not exploit problem structure will be exponential, as long as we do not prove P=NP. Therefore, the privacy requirements leave us no alternative from an exponential cost. The good news is that experiments show that problems with acceptable size (10-50 alternatives) can be solved in a few seconds.

Related Work The subtlety is how to formalize a distributed constraint satisfaction problem as an arithmetic circuit! An arithmetic circuit whose outcome is the set of all solutions was designed in [3]. If one tries to use that approach when only one solution is needed, the result returned by the function will reveal to everybody a lot more information than needed. It will tell, for example, that everybody is available and can reach the corresponding places on the days in the alternative solutions. It also reveals that at least one person is busy on each alternative that is not a solution. Some of this information can lead to undesired leaks of privacy. The approach of testing each alternative one by one has similar leaks.

In consequence, one needs to design arithmetic circuits returning only one solution. There is still the problem of which solution should be returned. It is possible to return the first solution in the lexicographical order on the search space [8]. However, knowing that the solution was computed in this way leaks that the alternatives placed before it in that lexicographical order are rejected by some agents.

Therefore, what we need is a probabilistic arithmetic circuit that returns a solution picked randomly among the possible solutions to the problem. MPC-DisCSP1 [9] and MPC-DisCSP2 [10], generate a secret permutation of domains (and eventually variables) on an encrypted description of the problem. The permuted encrypted problem is then input to an arithmetic circuit that computes an encryption of the first solution in lexicographic order. The solution is then translated with the inverse permutations to the initial problem formulation, before being decrypted. The used permutation guarantees to give each solution a chance to be returned, so that no secret about meeting acceptance/rejection can be inferred from the returned result. If there is no solution, this will intrinsically reveal to everybody that each alternative is constrained by some agent, but this leak is inherent to the problem and not to the algorithm[1].

The remaining problem is that the permutation in these techniques does not guarantee that solutions are picked with a uniform distribution over all solutions. Therefore, when an agent uses his constraints in several such computations, some statistical information can be extracted about his secrets, besides his acceptance of the solution. For example, if the returned solutions often specify a meeting in Quebec on Tuesday and rarely other alternatives, then it can be inferred that “some agent can go to Quebec only Tuesday”, with higher probability than what can be inferred by statistics ignorant of the used permutation algorithm.

In [5] we analyzed this leak and designed a scheme, MPC-DisCSP3, where the solutions are picked with a uniform distribution over the possible solutions. Repeated use of the same constraint in different problems will still suggest that a certain meeting is the only one possible, if it is always returned. However, the likelihood of the inference is lower than in the previous techniques and this time it is inherent to the problem and

[1]This can also be avoided by modifying the problem. Namely asking for a failure to be reported with a certain probability even if there exists a solution.

not to the algorithm.

It is easy to extend the technique such that alternatives known to be accepted by an agent are verified first, which saves someone's privacy in the detriment of the others.

Here we propose a new technique, MPC-DisCSP4, that improves on MPC-DisCSP3 both by improving efficiency and privacy. It can be used with optimization, by adding the same extensions as in [10].

2 Secure Arithmetic Circuit Evaluation

Secure evaluation of functions (arithmetic circuits) with secret inputs is introduced in [1]. For randomizing the representation of shuffled secrets we use $(+, \times)$-homomorphic encryption functions $E_{K_E} : Z_\mu \rightarrow Z_{\mu^2}$, i.e. respecting:

$$\forall m_1, m_2 \in Z_\mu : E_{K_E}(m_1)E_{K_E}(m_2) = E_{K_E}(m_1 + m_2).$$

Some encryption functions take a randomizing parameter r. However, we write $E_i(m)$ instead of $E_i(m, r)$, to simplify the notation. An example of a $(+, \times)$-homomorphic scheme with randomizing parameter is the Paillier encryption.

To destroy the visibility of the relations between the initial problem formulation and the formulation actually used in computations we design random joint permutations that are not known to any participant. Here we reformulate the initial problem by reordering its parameters. Related permutations appeared in Chaum's mix-nets [2]. The shuffling is obtained by a chain of permutations (each being the secret of a participant) on the encrypted secrets.

The secure multi-party simulation of arithmetic circuit evaluation proposed in [1] exploits Shamir's secret sharing. This sharing is based on the fact that a polynomial $f(x)$ of degree $t-1$ with unknown parameters can be reconstructed given the evaluation of f in at least t distinct values of x, using Lagrange interpolation. Instead, absolutely no information is given about the value of $f(0)$ by revealing the valuation of f in any at most $t-1$ non-zero values of x. Therefore, in order to share a secret number s to n participants $A_1, ..., A_n$, one first selects $t-1$ random numbers $a_1, ..., a_{t-1}$ that will define the polynomial $f(x) = s + \sum_{i=1}^{t-1}(a_i x^i)$. A distinct non-zero number k_i is assigned to each participant A_i. The value of the pair $(k_i, f(k_i))$ is sent over a secure channel (e.g. encrypted) to each participant A_i. This is called a (t, n)-threshold scheme. Once secret numbers are shared with a (t, n)-threshold scheme, evaluation of an arbitrary arithmetic circuit can be performed over the shared secrets, in such a way that all results remain shared secrets with the same security properties (the number of supported colluders, t) [1]. For Shamir's technique, one knows to perform addition and multiplications when $t \leq (n-1)/2$.

2.1 All Possible Schedules

In [3] one computes for each possible meeting, ϵ, a boolean circuit: $\bigwedge_{\phi_k \in C} \phi_k(\epsilon_{|X_k})$. The results of all these boolean circuits are revealed. Everybody learns whether each alternative meeting is possible or not. This is more than what one may want to leak (see Introduction).

Some people desire to examine all solutions before choosing one. Course-books claim that this may be a sign of an ill set problem. One should formulate such a problem as an optimization.

2.2 MPC-DisCSP1

MPC-DisCSP1 [9] is a multi-party computation technique. Former multi-party computation techniques can solve securely only certain functions, one such class of solved problems being the arithmetic circuits over finite fields. A Distributed CSP is not a function. A DisCSP can have several solutions for an input problem, or can even have no solution. Two of the three reformulations of DisCSPs as a function (see [9]) are relevant here: i) A function DisCSP^1() returning the first solution in lexicographic order, respectively an invalid valuation τ when there is no solution. ii) A probabilistic function DisCSP() which picks randomly a solution if it exists, respectively returns τ when there is no solution. For privacy purposes only the 2^{nd} alternative is satisfactory. DisCSP() only reveals what we usually expect to get from a DisCSP, namely *some* solution. DisCSP^1() intrinsically reveals more [9]. MPC-DisCSP1 implements DisCSP() in three phases:

1. The input DisCSP problem is jointly shuffled by reordering values (and eventually variables) randomly by composing secret permutations from each participant agent, and randomizing secret shares.

2. A version of DisCSP^1() where operations performed by agents are independent of the input secrets, is computed by simulating a certain arithmetic circuit evaluation with the technique in [1].

3. The solution returned by the DisCSP^1() at step 2 is translated into the initial problem definition using a transformation that is inverse of the shuffling at Step 1, and randomizing secret shares.

At step 2, MPC-DisCSP1 requires a version of the DisCSP^1() function whose cost is independent of the input since otherwise the users can learn things like: *The returned solution is the only one, being found after unsuccessfully checking all other valuations, all other valuations being infeasible.* The DisCSP^1() used by MPC-DisCSP1 is very complex, and MPC-DisCSP2 and MPC-DisCSP3 offers a simpler and faster version, parts of which are reused here.

3 MPC-DisCSP4

Now let us present MPC-DisCSP4, a multiparty computation simulating securely the method of [4]. MPC-DisCSP4 is shown in Algorithm 1.

MPC-DisCSP4 starts by sharing the encoded constraints with the Shamir secret sharing scheme. A complete (e.g. backtracking) CSP solver is used to generate two vectors: $S'' = \{\epsilon | \phi_0(\epsilon)\}$ and $S'[i] = p(S''[i])$, of size Θ. S'' contains all tuples ϵ that satisfy ϕ_0, and S' the shares obtained for securely evaluating $p(\epsilon)$ using [1].

```
procedure MPC-DisCSP do
    1. Each agent shares the {0,1} encoded secret constraints for each tuple.
    2. Compute $\epsilon$ returned by the CSP solver for $\phi_0$, and place the results in a shared secret vector $S''$, at the index given by the lexicographical order.
    3. Compute a vector $S'$, $S' = p(S''[\epsilon])$.
    4. Each agent $A_i$ encrypts shares in its vector $S'$ with his own public key, and submits the resulting vector to the mix-net.
    5. The mix-net shuffles the vectors of shares, $S'$, randomizing the shares at each permutation by adding shares of 0 exploiting homomorphic encryption.
    6. The shuffled $S'$ is broadcasted by the mix-net to agents.
    7. Compute now a vector $S$ with the Arithmetic Circuit of Function 1
    8. The mix-net decodes the vectors of shares, $S$, randomizing the shares at each inverse permutation by adding shares of 0 exploiting homomorphic encryption.
    9. The shuffled $S$ is broadcasted by the mix-net to agents.
    10. Compute the assignments of the variables in solution with Arithmetic Circuits in Equation 2.
    11. Reveal assignments to the interested agents.
```

Algorithm 1: MPC-DisCSP4

S' is now shuffled and the shares are randomized with a mix-net. Details are given later. Let ϵ_k denote the k^{th} tuple in the lexicographic order. We define:

$$\begin{aligned} h_1(P) &= 1 \\ h_i(P) &= h_{i-1}(P) * (1 - S'[i-1]) \end{aligned}$$

The index of the lexicographically first solution can be computed by accumulating the weighted terms of the h series:

$$S[i] = S'[i] * h_i(P) \tag{1}$$

The vector S is computed with Equation 1.

The vector S is now decoded by traversing the mix-net in the inverse direction and with the inverse permutations, randomizing the shares as at shuffling. The solution is the tuple of S'' situated at the same index as the only element of S' that is different from 0.

Alternatively[2], the solution can also be transformed into a set of indexes of values of the variables by following the following method. Assume the value of the u^{th} variable in the t^{th} tuple of the search space is denoted $\eta_u(t)$. In the end, the values in the solution are computed with the arithmetic circuits in Equation 2.

$$f_i(P) \quad = \quad \sum_{t=1}^{\Theta}(\eta_i(t) + 1) * S'[t-1] \tag{2}$$

Each variable x_i is assigned in the solution to the value in D_i at index given by the functions f_i, and can be revealed.

MPC-DisCSP4's mix-net for reordering vectors of shared secrets. Each agent A_i chooses a random secret permutation π_i, picked with a uniform distribution over the set of possible permutations: $\pi_i : [1..\Theta] \rightarrow [1..\Theta]$.

Each agent chooses a pair of keys for a $(+, \times)$-homomorphic public encryption scheme and publishes the public key. The secret shares, of the non-empty values computed in the vector S', are encrypted by each A_i with her public key and are serialized. The serialized encrypted vectors are sent to A_1. A_1 shuffles the serialized vectors according to her permutation π_1, then passes them to A_2 which applies π_2, etc., until the agent A_n which applies π_n. A_n sends each vector to the agent that originated it.

To avoid that agents get a chance to learn the final permutation by matching final shares with the ones that they encrypted, a randomization step is also applied at each shuffling. Each agent applies a randomization step on the set of shares for each element of S', by adding corresponding shares of zero. Since operands are encrypted, to be able to perform this summation we propose to exploit the $(+, \times)$-homomorphic properties of some encryption schemes. For each secret in S', a 0's Shamir shares are computed, and $\forall i, i{\le}n$, the $0's\ i^{th}$ share is encrypted with the public key of A_i, then it is multiplied to the corresponding A_i's encrypted share of the secret (resulting in resharing the secret). This assumes $\mu{>}\nu(n{+}1)$, for the decryption to be correct in Z_ν.

Example 2 *Let us see an example of how MPC-DisCSP4 is applied to the Example 2.*
$p(P, W)$ is not computed (ϕ_0).
$p(P,T)$=1, $p(Q,T)$=0, $p(Q,W)$=1.
S"=((P,T),(Q,T),(Q,W))
S'=(1,0,1)

[2]The technique of MPC-DisCSP3 can also be used, reconstructing first the permutation π from S''.

Shuffle (1,0,1), (assume it remains unchanged)
$h_1(P)$=1, $h_2(P)$=0, $h_3(P)$=0.
This is used according to Equation 1 to generate the vector S={1,0,0}.
Unshuffle S=(1,0,0):
S=(1,0,0)
The vector S is used to compute the values of the variables in the solution, using Equation 2:
$\eta_1(1)$=0, $\eta_1(2)$=1, $\eta_1(3)$=1. $\eta_2(1)$=0, $\eta_2(2)$=0, $\eta_2(3)$=1. $f_1(P)$=1, $f_2(P)$=1.

This signifies that the solution chosen by this arithmetic circuit is x_1=Paris and x_2=Tuesday.

Hiding the (in)existance of a solution. To hide the fact that a solution does not exist for a problem, the participants may prefer to miss an existing solution with a probability p. Then, the fact that no solution is found does not prove that no solution exist, and certain secrets induced by the lack of a solution will not be leaked. This can be achieved by generating secret random number(s) K, such that with probability p it has value 0, otherwise it has value 1. An example appears in [7].

Now, each of the secret elements of the vector S (or the secret values f) are multiplied with such a number K, losing the solution with a probability p.

Analysis and Conclusions When compared with classical agent approaches to solving distributed meeting scheduling and CSPs [11], the advantages and drawbacks of MPC-DisCSP4 are the ones defined by t-privacy [1], and highlighted in [6] (i.e. no collusion of less than t participants can learn anything, but the final solution with its quality, and what can be inferred from it).

The main advantage of MPC-DisCSP4 over its previous alternatives MPC-DisCSP1 and MPC-DisCSP2, is that it offers the solutions picked according to a uniform distribution over the total set of solutions. Compared to MPC-DisCSP3, it also hides the (in)existance of a solution.

From the space requirements point of view, it has the same worst case exponential complexity as MPC-DisCSP2 and MPC-DisCSP3, namely O(d^m), since it uses the same data structures (having to store and manipulate the whole vector S). It gains over MPC-DisCSP3 as it does not need the permutations π and π_i. From this point of view MPC-DisCSP4 is inferior to MPC-DisCSP1 which has polynomial space requirements.

In terms of time complexity, its worst performance (namely when Γ=Θ) is always approximatively 4 times better then MPC-DisCSP3 by not needing to translate values to vectors of size Θ. Compared to MPC-DisCSP1, which is $O(dm)$ times slower than MPC-DisCSP2, MPC-DisCSP4 will be faster. This is because MPC-DisCSP1 requires passing more than just the vector S.

When $\Gamma >> \Theta$, MPC-DisCSP4 can be much faster then competitors, which do not exploit public constraints.

In conclusion, MPC-DisCSP4 offers a privacy that is stronger than any of the previously existing methods, and is approximatively 4 times faster than MPC-DisCSP3, which has the closest performance in privacy.

References

[1] M. Ben-Or, S. Goldwasser, and A. Widgerson. Completeness theorems for non-cryptographic fault-tolerant distributed computating. In *STOC*, pages 1–10, 1988.

[2] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Com. of ACM*, 24(2):84–88, 1981.

[3] T Herlea, J. Claessens, G. Neven, F. Piessens, B. Preneel, and B. Decker. On securely scheduling a meeting. In *Proc. of IFIP SEC*, pages 183–198, 2001.

[4] M. Silaghi and V. Rajeshirke. The effect of policies for selecting the solution of a DisCSP on privacy loss. In *AAMAS*, pages 1396–1397, 2004.

[5] M.-C. Silaghi. Meeting scheduling guaranteeing n/2-privacy and resistant to statistical analysis (applicable to any DisCSP). In *3rd IC on Web Intelligence*, 2004.

[6] M. C. Silaghi and B. Faltings. A comparison of DisCSP algorithms with respect to privacy. In *AAMAS-DCR*, 2002.

[7] Marius Silaghi. A suite of secure multi-party algorithms for solving DisCSPs. Technical Report CS-2004-04, FIT, 2004.

[8] M.C. Silaghi. Arithmetic circuit for the first solution of distributed CSPs with cryptographic multi-party computations. In *IAT*, Halifax, 2003.

[9] M.C. Silaghi. Solving a distributed CSP with cryptographic multi-party computations, without revealing constraints and without involving trusted servers. In *IJCAI-DCR*, 2003.

[10] M.C. Silaghi and D. Mitra. Distributed constraint satisfaction and optimization with privacy enforcement. In *3rd IC on Intelligent Agent Technology*, pages 531–535, 2004.

[11] R. Wallace and M.C. Silaghi. Using privacy loss to guide decisions in distributed CSP search. In *FLAIRS'04*, 2004.