Meeting Scheduling Guaranteeing n/2-Privacy and Resistant to Statistical Analysis (Applicable to any DisCSP)

Marius Călin Silaghi
Florida Institute of Technology, Computer Sciences Department

Abstract

Distributed problems raise privacy issues. The user would like to specify securely his constraints (desires, availability, money) on his computer once. The computer is expected to compute and communicate for searching an acceptable solution while maintaining the privacy of the user.

Even without computers infested with spy viruses that capture the interaction with the user, most agent based approaches reveal parts of one agent's secret data to its partners in distributed computations [7]. Some cryptographic multi-party computation protocols [1] succeed to avoid leaking secrets at the computation of some functions with private inputs. They have been applied to find the set of all solutions for the meeting scheduling problem [3].

However, nobody yet succeeded to apply those techniques for finding a random *solution to the meeting scheduling problem. Note that revealing all solutions, when you only need a single one, leaks a lot of data about when others are, or are not, available. Some answers were proposed in our previous approaches to distributed constraint problems [4]. They guarantee that no agent can infer with certitude a secret from the identity of the solution of the problem (other than the acceptance of the solution), but guarantee nothing about inference of probabilistic information about secrets. Our new technique answers this problem, too.*

1. Introduction

Meeting scheduling is an old, recurrent and easily described problem that continues to fascinate researchers. An important reason is the advent of some requirements like privacy of user's constraints, or availability of associated resources. Informally, a set of agents want to meet. They search for a convenable meeting place and time that satisfies the private constraints of each of them. There can also exist some public constraints known by everybody, like the impossibility of meeting somewhere on a certain date due to known risks or legal considerations. One can add constraints on additional resources, such as needed material.

x1 \ x2	T	W
P	1	0
Q	0	1

Figure 1. Constraint: 0s mark rejected tuples.

The meeting scheduling problem is strictly equivalent to the so called distributed constraint satisfaction problem (CSP), with two variables. Additional resources can typically be modeled with new variables. In our algorithm we exploit the handiness of the CSP formalism.

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 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* $\phi 3$, *to Bob about* ϕ_1 *and* ϕ_3, *or to Carol about* ϕ_1 *and* ϕ_2.

Arithmetic Circuit The arithmetic circuits are a class of functions that can be solved 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), $f : F^i \rightarrow F^j$. 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.

Problem subtleties The subtlety is how to formalize the meeting scheduling 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 [4]. 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 and MPC-DisCSP2 [5], 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.

The remaining problem is that the permutation in [4] 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 this paper we analyze this leak and design a scheme, MPC-DisCSP3, where the solutions are picked with a uni-

form 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 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.

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. In IAT2004 will appear an example of how to extend our techniques to optimization.

2.2. MPC-DisCSP1

MPC-DisCSP1 [4] 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 [4]) 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 [4]. 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 offers a simpler and faster version, parts of which are reused here.

3. Uniformly Distributed Selection

MPC-DisCSP1 and MPC-DisCSP2 give a chance to each solution to be returned [5]. The solution can be seen as a random variable over the set of tuples ϵ that have $p(\epsilon) = 1$.

$$p(\epsilon) = \prod_{\phi_k \in C} \phi_k(\epsilon_{|x_k})$$

However, we have proven that none of the existing techniques returns solutions according to a uniform distribution:

Theorem 1 *Shuffling variables and domains for a CSP does not guarantee that the first solution in the obtained lexicographic order is selected according to a uniform distribution over the set of all solutions.*

Proof. Consider the CSP induced by the DisCSP of Example 1, without the constraints ϕ_1 and ϕ_3. Applying random permutation of domains and eventually variables drawn from a uniform distribution over the set of possible distributions:

- the solution (Q, W) appears 3/8% of the times.
- the solution (Q, T) appears 1/4% of the times.
- the solution (P, T) appears 3/8% of the times.

The frequency with which a solution is drawn is inverse proportional to the frequency of its values among other solutions. □

Therefore, if an agent participates with the same constraints in several computations, statistical information can be extracted concerning the occurrence of the values in other solutions of the agent. Namely, a solution that occurs very often indicates that some of its assignments are rare. Let us now present a technique called MPC-DisCSP3 that is slightly more complex than MPC-DisCSP2 and that returns solutions according to a uniform distribution.

Theorem 2 *Consider the following process on a CSP:*

- *Create a (large) vector S' containing the values $p(\epsilon)$ for all search space tuples ϵ, in lexicographic order.*
- *Shuffle the vector S' according to a permutation π picked with a uniform distribution over the possible permutations.*
- *Pick the first value of S' having $p(\epsilon) = 1$. Choose ϵ as the solution to be returned.*

The tuple returned by these three steps is chosen according to a uniform distribution over all solutions (equivalent to picking it randomly from a set with all solutions).

Proof. For any sufficiently large number of applications of the described procedure, the possible permutations π applied to S' are drawn a relatively equal number of times, due to their uniform distribution. Therefore, all obtained permutations of the values of S' will result a relatively equal number of times. By symmetry, each ϵ with $p(\epsilon) = 1$ will be placed an equal number of times before all the other solutions. Therefore, the method defines its outcome as a random variable with uniform distribution over the set of all solutions. □

function value-to-unary-constraint2(v, M)
  $\{x_i\}_{0 \le i \le M}$, x_0=1, x_{i+1}=$x_i * (v-i)$
  $\{y_i\}_{0 \le i \le M}$, y_M=1, y_{i-1}=$y_i * (i-v)$
  $u_k = \frac{1}{k!(M-k)!} x_k y_k$, where $0! \stackrel{\text{def}}{=} 1$.
  return u.

Algorithm 1: Transforming secret value $v \in \{0, 1, 2, ..., M\}$ to a shared secret unary constraint.

4. MPC-DisCSP3

Now let us present MPC-DisCSP3, a multiparty computation simulating securely the method of the Theorem 2.

MPC-DisCSP3 starts by sharing the encoded constraints with the Shamir secret sharing scheme. Then, a vector S' of size $\Gamma = \prod_{k=1}^{m} d_k$ is computed by evaluating for each tuple ϵ compatible with ϕ_0, the arithmetic circuits $p(\epsilon)$. Each $p(\epsilon)$ is placed in the vector S' on the position defined by the lexicographical order on ϵ. Each agent applies on its shares of S' a common permutation π:

$$\pi : [1..\Gamma] \rightarrow [1..\Gamma].$$

that moves the tuples rejected by ϕ_0 to the end of S'.

π can be the permutation defined by a sort algorithm that scans S' from low indexes and exchanges each empty element, $S'[i]$ with the last non-empty element $S'[j]$. If the last obtained i and j are stored such that scanning avoids to repeat tests and ends when $i = j$, then the cost of building the permutation π is $O(\Gamma)$. The number of tuples that are not rejected by ϕ_0 is denoted by Θ. Alternatively, $S'[1..\Theta]$ can be obtained as the value of $p(\epsilon)$ in all solutions ϵ' of ϕ_0, as found by a deterministic (backtracking) search techique.

The problem is now shuffled and the shares are randomized with a mix-net. One actually shuffles only the first Θ elements of the vector S'. 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:

$$id(P) = \sum_{i=1}^{\Theta} i * p(\epsilon_i) * h_i(P) \qquad (1)$$

A result of 0 means that there is no solution. The cost of this computation is $(c+1)\Theta$ multiplications of secrets, like for MPC-DisCSP2. After computing $id(P)$ with the arithmetic circuit in Equation 1, the vector S is computed with Equation 2, which calls Algorithm 1.

$$S\text{=value-to-unary-constraint2}(id-1, \Theta-1) \qquad (2)$$

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. π^{-1} is applied to S. Any index after the end of S, is considered by π^{-1} to be empty (they were rejected by ϕ_0). The value of the u^{th} variable in the t^{th} tuple of the search space is $\eta_u(t)$, computed with Equation 3. In the end, the values in the solution are computed with the arithmetic circuits in Equation 4.

$$\eta_u(t) = \lfloor (t-1) / \prod_{k=1}^{u-1} d_k \rfloor \mod d_u \quad (3)$$

$$f_i(P) = \sum_{t=1}^{\Theta} (\eta_i(t)+1) * S'[t-1] \quad (4)$$

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-DisCSP3'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 $\forall i, i \leq n$, 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-DisCSP3 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'=(1,0,_,1): After applying $\pi = (0,1,4,3)$, S'=(1,0,1,_)
Shuffle (1,0,1), (assume it remains unchanged)
$h_1(P)$=1, $h_2(P)$=0, $h_3(P)$=0.
The index of the solution is computed with Equation 1, yielding id(P)=1. This is used according to Equation 2 to generate the vector S=$\{1,0,0\}$.
Unshuffle S=(1,0,0): Apply $\pi^{-1} = (0,1,4,3)$, S=(1,0,0,_)
The vector S is used to compute the values of the variables in the solution, using Equations 3 and 4:
$\eta_1(1)$=0, $\eta_1(2)$=1, $\eta_1(3)$=0, $\eta_1(4)$=1. $\eta_2(1)$=0, $\eta_2(2)$=0, $\eta_2(3)$=1, $\eta_2(4)$=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.

Analysis and Conclusions When compared with classical agent approaches to solving distributed meeting scheduling and CSPs [7], the advantages and drawbacks of MPC-DisCSP3 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-DisCSP3 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, as guaranteed by the Theorem 2. From the space requirements point of view, it has the same exponential complexity as MPC-DisCSP2, namely O(d^m), since it uses the same data structures (having to store and manipulate the whole vector S). The only additional structures, namely the permutations π and π_i have the same size as S, and do not change the complexity. From this point of view MPC-DisCSP3 is clearly inferior to MPC-DisCSP1 which has polynomial space requirements.

In terms of time complexity, its worst performance (namely when Γ=Θ) can be worse than the one of MPC-DisCSP2. This is due to the fact that the messages for shuffling are larger and the shuffling involves more computations, given by the size of S'. Compared to MPC-DisCSP1, which is $O(dm)$ times slower than MPC-DisCSP2, MPC-DisCSP3 will be faster. This is because MPC-DisCSP1 requires passing more than just the vector S.

When $\Gamma >> \Theta$, MPC-DisCSP3 can be much faster then competitors, which do not exploit public constraints.

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. *Communications of the 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. Solving a distributed CSP with cryptographic multi-party computations, without revealing constraints and without involving trusted servers. In *IJCAI-DCR*, 2003.

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

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

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