Secure Multi-party Computation for selecting a solution according to a uniform distribution over all solutions of a general combinatorial problem

Marius-Călin Silaghi
Florida Tech

Abstract

Secure simulations of arithmetic circuit and boolean circuit evaluations are known to save privacy while providing solutions to any probabilistic function over a field. The problem we want to solve is to select a random solution of a general combinatorial problem. Here we discuss how to specify the need of selecting a random solution of a general combinatorial problem, as a probabilistic function. Arithmetic circuits for finding the set of all solutions are simple to design [24].

We know no arithmetic circuit proposed in the past, selecting a single solution according to a uniform distribution over all solutions of a general constraint satisfaction problem. The only one we are able to design has a factorial complexity in the size of the search space (O($d^m!d^m$) multiplications of secrets), where m is the number of variables and d the maximal size of a variable's domain.

Nevertheless, we were able to develop a methodology combining secure arithmetic circuit evaluation and mix-nets, able to compile the problem of selecting a random solution of a CSP to a n/2-private multi-party computation assuming passive attackers. The complexity of this solution is more acceptable, O(d^m) multiplications, being therefore applicable for some reasonable problems, like meeting scheduling.

Constraint satisfaction is a framework extensively used in some areas of artificial intelligence to model problems like meeting scheduling, timetabling, the stable marriages problem, and some negotiation problems. It is based on abstracting a problem as a set of variables, and a set of constraints that specify unacceptable combination of values for sets of distinct variables.

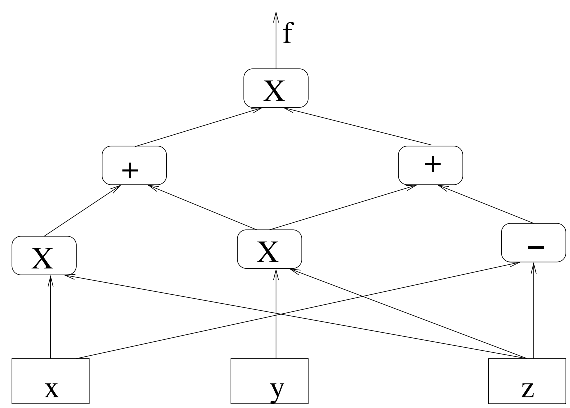

Figure 1. An arithmetic circuit, $f{=}(xz + yz)(yz{+}(x{-}z))$. Each input can be the secret of some participant. The output may not be revealed to all participants. All intermediary values remain secret to everybody.

1. Introduction

Secure multi-party computations can simulate any arithmetic circuit [2] or boolean circuit [25, 20] evaluation. 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 (see Figure 1). Each leaf is a constant. In a secure arithmetic circuit evaluation, a set of participants perform the operations of an arithmetic circuit over some inputs, each input being either public or an (encrypted/shared) secret of one of them. The results of the arithmetic circuit are the values of some predefined nodes. The protocol can be designed to reveal the result to only a subset of the agents, while none of them learns anything about intermediary values. We say that the multi-party computation *simulates* the evaluation of the arithmetic circuit. A *boolean circuit* is similar, just that the leafs are boolean truth values, false or true, often represented as 0 and 1. The rest of the nodes are boolean operators like AND or XOR.

In this work we only concentrate on arithmetic circuits. A function does not have to be represented in this form to be solvable using general secure arithmetic circuit evaluation. It only needs to have such an equivalent representation. For example, the operation $\sum_{i=B}^{E} f(i)$ is an arithmetic circuit if B and E are public constants and $f(i)$ is an arithmetic circuit. The same is true about $\prod_{i=B}^{E} f(i)$. Such constructs are useful when designing arithmetic circuits.

The arithmetic circuit evaluation can also implement probabilistic functions. A (probabilistic) function f over a finite field F is defined in [2] as $f : F^n \times R^m \rightarrow F^n$, where R a random variable with uniform distribution over F. For example, each participant can provide a secret random number drawn with uniform distribution over F, and the random input can be taken as their sum.

The challenge is to apply secure arithmetic circuit evaluation to find a solution to combinatorial problems that are represented in the handy framework of Constraint Satisfaction Problems.

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 finite domains such that x_i can take values only from $D_i = \{v_1^i, ..., v_{d_i}^i\}$. $C = \{\phi_1, ..., \phi_c\}$ is a set of constraints. A constraint ϕ_i limits the legality of each combination of assignments to the variables of an ordered subset $X_i = \{x_{i_1}, ..., x_{i_{k_i}}\}$ of the variables in X, $X_i \subseteq X$. An assignment is a pair $\langle x_i, v_k^i \rangle$ meaning that the variable x_i is assigned the value v_k^i.

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 $\epsilon_{|X_i}$. The search space of a CSP is the Cartesian product of the domains of its variables.

Example 1 *In a problem P, one has to find a place (x_1) and time (x_2) for meeting. 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. $D_2 = \{T, W\}$.*

x1 \ x2	T	W
P	0	1
Q	1	0

Figure 2. A constraint between two variables, place (x_1) $x_1 \in \{Paris(P), Quebec(Q)\}$, and time ($x_2$) $x_2 \in \{Tuesday(T), Wednesday(W)\}$. The 0s mark rejected tuples. I.e. this constraint allows only the pairs (P,W) and (Q,T), and can be written $\{(P, W), (Q, T)\}$.

There are two constraints: $\phi_1 = \{(P, W), (Q, T)\}$, and $\phi_2 = \{(P, W), (Q, T), (Q, W)\}$. ϕ_1 is depicted in Figure 2.

The problem is to find values for x_1 and x_2 satisfying both ϕ_1 and ϕ_2.

Example 2 *The timetabling problem can be modeled with CSPs by using a separate variable to model the task to be allocated to each time slot. The constraints specify the relations that have to be assured between tasks in different slots.*

Example 3 *The stable marriages problem is the problem of finding a set of matches between a set of males and a set of females such that if any person from the set of females, $Alice$, prefers some male, Bob, to the partner selected for her, then Bob prefers his current partner to $Alice$. Also, if any male, Bob, prefers some female, $Alice$, to the partner selected for him, then $Alice$ prefers her current partner to Bob.*

In a CSP representation of the stable marriages problem, there is one variable for each female, specifying the male assigned to her by the solution, or the state single*. The constraints are obtain by preprocessing the input of participants about their preferences.*

Some simple arithmetic circuits can implement this preprocessing. Let us give an example:

We can denote the females with $A_1, ...A_m$ and the males with $B_1, ...B_{n-m}$

The input of each female A_i can specify a preference value $P_{A_i}(B_j, B_k)$ for each pair of males. Each male B_i can specify a preference value $P_{B_i}(A_j, A_k)$

for each pair of females.

$P_{A_i}(B_j, B_k)$=1 *if and only if* A_i *prefers* B_j *to* B_k. *Otherwise* $P_{A_i}(B_j, B_k)$=0. $P_{B_i}(A_j, A_k)$=1 *if and only if* B_i *prefers* A_j *to* A_k. *Otherwise* $P_{B_i}(A_j, A_k)$=0.

Then, a constraint ϕ^{ij} *is defined between each two variables* x_i *and* x_j*, specifying the males assigned to the females* A_i *and* A_j. $\phi^{ij}[u, v]$ *is the acceptance value of the pair of matches:* $(A_i, B_u), (A_j, B_u)$.

One synthesizes $m(m-1)/2$ *constraints:*

$$\begin{aligned}\phi^{i,j}[u, v] &= (1 - P_{A_i}(B_v, B_u)) + \\ &\quad P_{A_i}(B_v, B_u) * P_{B_v}(A_j, A_i) \\ \phi^{i,j}[u, u] &= 0\end{aligned}$$

We consider that a set of participants are the source of these problems, each agent has his own private constraint and one has to find agreements for a solution, from the set of possible alternatives, that satisfies everybody. The handy formulation modeling this kind of problems is called Distributed Constraint Satisfaction [46, 7, 44]. There are several versions of this formalism, and we select one that explains best the security problem.

Definition 1 *A Distributed CSP (DisCSP) is defined by five sets* (A, X, D, C, M). 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, i>0*, is known only to one agent, being the secret of that agent. There may exist a public constraint in* C, ϕ_0*, known to everybody.* M *is a set of sets of agents from* A, M=$\{A^1, ..., A^m\}$. *It specifies a mapping of variables to their owners. Each variable* x_i *is owned by the set of agents* A^i*, that are entitled to learn its assignment in the solution.*

Example 4 *In another view of the problem P, two persons Alice* (A_1) *and Bob* (A_2) *want to find a common place* (x_1) *and time* (x_2) *for meeting.* 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 on the possible time and place of their meeting. Alice accepts only* $\{(P, T), (P, W), (Q, T)\}$ *which defines* ϕ_1. *Bob accepts either of* $\{(P, W), (Q, T), (Q, W)\}$*, defined by* ϕ_2. *There is also a publicly known constraint,* ϕ_0*, which due to an announced strike forbids a meeting in Paris on Tuesday,* $\phi_0 = \{(P, W), (Q, T), (Q, W)\}$.

The problem is to publish values for x_1 *and* x_2 *satisfying all constraints and without revealing anything else about* ϕ_2 *to Alice or about* ϕ_1 *to Bob.* A^1=A^2=$\{A_1, A_2\}$

Remark 1 *Note that once a distributed CSP has all its secret parameters (i.e. constraints) shared among agents (e.g. with Shamir's scheme [37]) in view of a multi-party computation, all differences between a CSP and a DisCSP disappear, except for the existence of the participants and of the sets of owner agents that will receive the results of the computation.*

Problem subtleties. The problem is how to formalize the DisCSP as an arithmetic circuit! An arithmetic circuit whose outcome is the set of all solutions was designed in [24]. 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. For example, with meeting scheduling it will tell that everybody is available and can reach the corresponding places on the days in the alternative solutions. It also suggest 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 until a solution is found has similar potential leaks of privacy.

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 [38]. However, knowing that the solution was computed in this way leaks the fact that the alternatives placed before it in the lexicographical order on the search space are rejected by some agent.

Therefore, what we need is a probabilistic arithmetic circuit that returns a solution picked randomly among the possible solutions to the problem. An alternative we considered is to compute the lexicographically first solution for all permutations of domains [44] (and eventually variables). The solution will then be selected randomly among the existing solutions. 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 permutations of domains and variables do not always lead to a selection of the solution with a uniform distribution over the possible solutions. Therefore, when an agent uses his constraints in several computations using the same algorithm, 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 some other alternatives, then it can be inferred that some participant can go to Quebec only Tuesday, with higher probability than what statistics ignorant of the used permutation algorithm could infer.

In this paper we analyze this leak and design schemes 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 not to the algorithm.

Moreover, it is easy to extend the technique such that alternatives already known to be accepted by an agent are verified first, if it is acceptable to save someone's privacy in the detriment of the others.

Next we present the background techniques, then we propose an arithmetic circuit that returns the lexicographically first solution of a distributed CSP, which will be used in subsequent algorithms. In Section 4 we present a set of theoretical results concerning the properties of the distributions of solutions achieved with different permutation methods. In Section 4.2 we prove a method guaranteeing a uniform distribution of the returned solutions over the set of all solutions. In the subsequent sections we propose an arithmetic circuit and then a faster multi-party computation implementing the method proposed in Section 4.2.

2. Background

CSPs. Constraint Satisfaction Problems are used to modeling combinatorial problems. The have matured as an area starting with the work of [33, 42, 29, 17, 12]. Distributed CSPs have been addressed under different formulations in the past [35, 41, 26, 28, 46, 7, 44, 13, 8, 3, 27, 23, 47], and secrecy within DisCSPs has been stressed several times [32, 16, 38, 36, 45, 15].

Secure Arithmetic Circuit Evaluation. We use the technique for secure evaluation of functions with secret inputs described in [2]. Several recent versions are based on (oblivious) boolean circuit evaluation [25, 20]. There exists a large amount of work on multi-party computations [18, 6, 1, 21, 22, 10, 11, 4, 9, 30, 14].

We use $(+, \times)$-homomorphic encryption functions E_{K_E}, i.e. respecting:

$$\forall m_1, m_2, E_{K_E}(m_1)E_{K_E}(m_2) = E_{K_E}(m_1 + m_2).$$

Some encryption functions take a randomizing parameter r. However, we write $E_i(m)$ instead of $E_i(m, r)$, to simplify the notation. A good example of a $(+, \times)$-homomorphic scheme with randomizing parameter is the Paillier encryption [34].

To destroy the visibility of the relations between the initial problem formulation and the formulation actually used in computations one can exploit random joint permutations that are not known to any participant. In one of the techniques presented here we reformulate the initial problem by reordering its parameters. Such permutations appeared in Chaum's mix-nets [5] and in [31]. The shuffling is obtained by a chain of permutations (each being the secret of a participant) on the encrypted secrets.

The secure arithmetic circuit evaluation in [2] exploits Shamir's secret sharing [37]. 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. It can be done 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, computations of an arbitrary agreed function of a certain class 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) [2, 43]. For Shamir's technique, one knows to perform addition and multiplications when $t \leq (n-1)/2$. At multiplication, [2] proposes to perform a randomization of the shares of the result by adding to them shares of 0. We are going to exploit a related method in one of the proposed protocols.

All Possible Schedules. It is pretty simple to design a technique revealing all possible solutions to a constraint satisfaction problem. In [24] one tries to solve meeting scheduling problems, computing 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. As we discussed in the introduction, this is more than what one may want to leak when a single solution is searched for.

It is true that sometimes people think that is better to see all solutions before choosing one. Nevertheless, the well-known book of Garey and Johnson [19] claims that this is a typical example of an ill set problem. Namely, most often one should formulate such a problem as an optimization problem. A way to extend the techniques propose here to some optimization problems is straightforward but will not be presented now.

3. Arithmetic Circuit for finding the first solution in lexicographic order

Let us now present an arithmetic circuit for finding the lexicographically first solution of a (distributed) CSP. This is a component of all the subsequent protocols proposed in this article.

A lexicographically first solution is the first solution that would be found by traversing all the possible com-

function value-to-unary-constraint2(v, M)

1. Jointly, all agents build a vector u, $u = \langle u_0, u_1, ..., u_M\rangle$
   with $3M-1$ multiplications of secrets, computing:
   1. $\{x_i\}_{0\leq i\leq M}$, $x_0=1$, $x_{i+1}=x_i*(v-i)$
   2. $\{y_i\}_{0\leq i\leq M}$, $y_M=1$, $y_{i-1}=y_i*(i-v)$
   then, $u_k = \frac{1}{k!(M-k)!}x_k y_k$, where $0! \stackrel{\text{def}}{=} 1$.
2. Return u.

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

binations in the search space of the problem, ordered lexicographically. The lexicographic order is defined by the order on the variables, and then by the order of each domain for each variable. It is similar to the order used on words by dictionaries, considering that each letter in a word is the value of a variable, the position of each letter in the word is induced by the order on variables, and the alphabetical order is given by the order on the domain of that variable. An arithmetic circuit finding the lexicographically first solution of a CSP was described in [38], but it is $O(md)$ times slower than the following one (m being the number of variables and d the maximum domain size). Take a CSP, P=(X,D,C). The size of the search space (total number of tuples) is $\Theta = \prod_{k=1}^{m} d_k$.

We define $p(\epsilon) = \prod_{\phi_k\in C}\phi_k(\epsilon_{|X_k})$, and ϵ_k denotes the k^{th} tuple in the lexicographic order.

A vector S' is defined as $S'[k] = p(\epsilon_k)$.

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

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

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

A result of 0 means that there is no solution. The cost of this computation is $(c+1)d^m$ multiplications of secrets, md times less than the technique in [38], which is $O((cm+m^2)d^{m+1})$, where $d=\max_i(d_i)$.

One can then compute the values of the different variables in the found solution. We first transform the index id of the solution computed with the arithmetic circuit in Equation 1 into a shared vector S of size Θ where only the id^{th} element is 1 and all other elements are 0. This is achieved using the arithmetic circuits called in Equation 2. The arithmetic circuits for transforming the solution to a vector, shown in Algorithm 1, has $3M$.

The value of the u^{th} variable in the t^{th} tuple of the search space is $\eta_u(t)$, computed with Equation 3. The arithmetic circuit, $f_i(P)$, (see Equation 4), can now be used to compute the value of each variable x_i in the solution.

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

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

$$f_i(P) = \sum_{t=1}^{\Theta} (\eta_i(t)+1) * S[t] \quad (4)$$

It can be noticed that the space required for computing S is $O(d^m)$. This can be reduced by not reusing intermediary results in Algorithm 1 and computing S on demand during the evaluation of f functions, but with a significant efficiency loss, namely $O(d^{2m})$ multiplications of secrets.

Example 5 *Let us see a full example of how this arithmetic circuit is applied to Example 1.*

The lexicographic order (using actually the inverse of the order on variables) is {(P,T),(Q,T),(P,W),(Q,W)}
$p(P,T)$=0, $p(Q,T)$=1, $p(P,W)$=1, $p(Q,W)$=0.
$h_1(P)$=1, $h_2(P)$=1, $h_3(P)$=0, $h_4(P)$=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={0,0,1,0,0}.

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.
$f_1(P)$=2, $f_2(P)$=1.

The solution chosen by this arithmetic circuit is therefore x_1=Quebec and x_2=Tuesday.

Note that the solution leaks the fact that it is not possible to meet in Paris on Tuesday.

A faster alternative to isolate a random element with the desired value out of an array is to use the following arithmetic circuit. We define:

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

The lexicographically first solution can be isolated by using the terms of the h series:

$$S[i] = S'[i] * h_i(P) \quad (5)$$

4. A uniform distribution over the solutions of a CSP

A question to be asked is whether a solution computed over a random permutation of variables and domains could help remove the aforementioned leaks shown for the previous arithmetic circuit.

Theorem 1 *For any CSP whose search space has size Θ, and for any j, $0{\le}j{<}\Theta$ there exists a shuffling of the values in its domains such that a solution with any initial lexicographic position i in this search space is mapped into the position j of the obtained problem.*

Proof. This can be proven by constructing the shuffling. First, we find the positions p_k^j and p_k^i of each value in the domain of each variable x_k for the tuples with lexicographic positions j and i. This is done by iteratively computing for k from n to 1, $p_k^j := j\%d_k$, $j = \lfloor j/d_k \rfloor$. Next, the permutation, π_k, for the domain of each variable x_k is chosen such that $\pi_k[p_k^j] = p_k^i$. The shuffling defined by permutations π_k satisfies the requirements and the theorem is proven. □

Corollary 1.1 *For any CSP and a given solution, there exists a shuffling of the values in its domains mapping that solution into the lexicographically first tuple of the obtained problem.*

As follows from the previous corollary, one cannot extract with certitude any secret by an inference based on the identity of the solution of the problem shuffled with unknown permutations of the domains (except that the solution is accepted by everybody). However, statistical information may be leaked as seen further.

4.1. Shuffling Domains and Variables

The solution can be seen as being generated by a random variable over the set of tuples ϵ that have $p(\epsilon) = 1$. Let us analyze this random variable for the case where values and eventually variables are permuted randomly according to a uniform distribution over the set of all possible permutations.

Theorem 2 *Shuffling the 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 4, without the constraint ϕ_1.

Applying random permutation of domains drawn from a uniform distribution over the set of possible distributions:

- the solution (P, W) appears 1/4% of the times.
- the solution (Q, T) appears 1/2% of the times.
- the solution (Q, W) appears 1/4% of the times.

It can be noticed that the frequency with which the solution is drawn is inverse proportional to the frequency of its values among the other solutions. □

The next question is whether adding random permutations of variables could lead to a uniform distribution.

Theorem 3 *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 again the CSP induced by the DisCSP of Example 4, without the constraint ϕ_1.

Applying random permutation of domains and variables drawn from a uniform distribution over the set of possible distributions:

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

It can be noticed that the frequency with which the solution is drawn is inverse proportional to the frequency of its values among the other solutions. The lack of uniformity is slightly less accentuated than for the case where only domains are reordered. □

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.

4.2. Selection according to a uniform distribution over all solution

Now we define an abstract method that will be proven to select a solution of a constraint satisfaction problem according to a uniform distribution over the set of all solutions.

Theorem 4 *Consider the application of the following process to a CSP:*

- *Create a (big) 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 the three steps above is chosen according to a uniform distribution over all solutions (tuples having $p(\epsilon) = 1$).

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. □

5. Arithmetic circuit for a uniform distribution over all solutions

In the previous section we have constructed a method that can select a solution with a uniform distribution over the set of solutions of a CSP, using a random permutation. Here we explain ways in which the random permutation can be achieved with an arithmetic circuit. The technique has to perform the computation for all possible permutation and the solution is picked randomly in a way that hides the selected permutation.

It is possible to design an arithmetic circuit that has as input a set of coin tosses and the constraints, and selects randomly a solution with a uniform distribution over the set of all solutions. We identify the next solution.

Selecting among results computed for each permutation. This approach requires $O(\Theta!\Theta)$ multiplications. The arithmetic circuit is constructed as follows:

For each of the $k \in [1..\Theta!]$ possible permutations π_k of all tuples in the search space, evaluate securely an arithmetic circuit that finds the first solution. The result for each variable x_i is stored in a vector $F_i[k]$.

The agents generate a random shared secret r, uniformly distributed between 0 and $\Theta!-1$. If the computation is performed in a finite field not much larger than $\Theta!$, then r can be generated by simply having each agent A_i share a random secret number r_i, picked with a uniform distribution over the elements of the field. r is computed by summing up all the r_i numbers and testing that the result is not larger or equal than $\Theta!$. If r does not pass the test, one starts from scratch building a new r.

When a secret random number $r, r \in [0..\Theta!-1]$ is obtained, it is transformed into a vector R with:

$$R = \text{value-to-unary-constraint2}(r, \Theta!-1)$$

Each element $R[k]$ of the vector R is multiplied to each value $F_i[k+1]$ for each i.

The values for each variable of all the $\Theta!$ solutions are summed with each other: $f_i = \sum_k F_i[k]$. The results of f_i are revealed to the agents owning x_i.

Remark 2 *If a uniform distribution is not desired, one can create the previous circuit only for all permutations of domains and eventually of variables. The complexity decreases accordingly.*

Remark 3 *Note that in the previous algorithm it is possible to exploit the knowledge about tuples rejected by the public constraint ϕ_0, during the simulation of the arithmetic circuit that computes the lexicographically first solutions. Namely the rejected tuples can be permuted at the end of S', with a permutation computed similar to the permutation π from the next technique. After computing the vector S (Equation 2) over the remaining tuples, the inverse of the π permutation has to be applied before computing the functions f_i.*

In the following section we show a multi-party computation combining simulations of arithmetic circuit evaluation with a mix-net for creating and respectively decoding the random permutation of the tuples.

6. Faster multi-party protocol for a uniform distribution over all solutions

In the previous sections we have studied the unnecessary leaks of secrets due to the way solutions to constraint satisfaction problems can be picked by multi-party computation protocols. We have proposed a method proved to generate solutions of a CSP with a uniform distribution over the set of all solution of the CSP. We have also shown an implementation of that method using arithmetic circuits, the complexity being given by the factorial of the size of the search space, $O(\Theta!\Theta)$ multiplications of secrets. Now we are going to propose another technique combining arithmetic circuits with mix-nets [5] and whose complexity is only $O(\Theta)$ multiplications of secrets.

We start by sharing the $\{0,1\}$ encoded elements of each constraint 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:

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

Each $p(\epsilon)$ is placed in the vector S' on the position defined by the lexicographical position of ϵ among all tuples. 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 is 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 i and j are stored such that scanning starts with them 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.

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. The shuffling performed by the mix-net is reversible since the agents remember the permutations that they have performed. It can be undone, i.e. *unshuffled*.

The index of the lexicographically first solution can be computed as in Equation 1. A result of 0 means that there is no solution. As for the circuit in Section 3, the cost of this computation is $(c+1)\Theta$ multiplications of secrets.

After computing $id(P)$ with the arithmetic circuit defined by the Equation 1, the vector S is computed with the arithmetic circuit defined by Equation 2.

One may check now whether the first element of S, S[0], is 1, which would mean that no solution exists and the computation can stop. However, this check can enable attacks by participants that refuse to continue after learning that a solution exists.

The vector S is then shifted one position to the left, discarding the first element. S is then unshuffled 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 contain the value 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 defined by Equation 6.

$$f_i(P) \quad = \quad \sum_{t=1}^{\Theta}(\eta_i(t)+1) * S'[t-1] \qquad (6)$$

Each assignment in the solution is defined by the results to the functions f_i are revealed to the agents owning it, namely those in A^i.

Mix-net for reordering shared secret DisCSPs. It remained to detail the way in which the previous technique shuffles the vector S'. Each agent A_i chooses a random secret permutation π_i, picked according to a uniform distribution over the set of possible permutations:

$$\pi_i : \quad [1..\Theta] \rightarrow [1..\Theta].$$

Each agent A_i chooses a pair of keys for a $(+,\times)$-homomorphic public encryption scheme and publishes the public key, K_i. The secret shares of the first Θ non-empty values computed in the vector S' are encrypted by A_i with his public key and then are serialized.

The serialized encrypted vectors are passed along a mix-net [2], mentioned in Section 2, and shuffled according to the permutation π_1 of A_1, then passed to A_2 which applies π_2, and so on, until the agent A_n which applies π_n. Then, each encrypted vector is sent shuffled 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 submitted to the mix-net, a randomization step is also applied at each shuffling. Each agent applies a randomization step on the set of shares for each value of S', by adding shares of 0, as in [2]. Because of the encryption, this randomization step is based on $(+,\times)$-homomorphic encryption. Namely, each agent A_j in the mix-net generates Θ sets of shares of 0, z^j_k, $k \in [1..\Theta]$. A_j first encrypts the i^{th} share, $z^j_k[i]$, of each set of shares of 0 with the public key of A_i, K_i. Then, A_j multiplies the encrypted $z^j_k[i]$, $E_{K_i}(z^j_k[i])$, to the encrypted shares of $S'[k]$ sent by A_i, for each i and each k.

Example 6 *Let us see an example of how the new multi-party computation is applied to the Example 4.*
$p(P,T)$ *not computed,* $p(Q,T)$=1, $p(P,W)$=1, $p(Q,W)$=0.
S'=(_,1,1,0)
After applying $\pi = (4,1,2,0)$.
S'=(0,1,1,_)
Shuffle (0,1,1), assume it remains unchanged
$h_1(P)$=1, $h_2(P)$=1, $h_3(P)$=0.
The index of the solution is computed with Equation 1, yielding id(P)=2. This is used according to Equation 2 to generate the vector S={0,0,1,0}.
Unshuffle S[1-3]=(0,1,0)
Apply $\pi^{-1} = (4,1,2,0)$
S=(_,0,1,0)
The vector S *is used to compute the values of the variables in the solution, using Equations 3 and 6:*

$\eta_1(1){=}0$, $\eta_1(2){=}1$, $\eta_1(3){=}0$, $\eta_1(4){=}1$. $\eta_2(1){=}0$, $\eta_2(2){=}0$, $\eta_2(3){=}1$, $\eta_2(4){=}1$. $f_1(P){=}1$, $f_2(P){=}1$.

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

7. Applications and Experiments

A set of applications of this approach to solving auctions is presented in [39]. An implementation within a new declarative programming language where users only need to specify constraints and the multi-party computation is assembled, is present at [40]. Tests show that a complex meeting scheduling can be solved securely in approximatively 1 minute, the main cost at this problem size being due to the mix-net (1 second/mixnet/participant).

8. Conclusions

In this article we have proposed a couple of multi-party computation techniques for extracting securely a solution selected according to a uniform distribution over the set of all solutions of a combinatorial problem formalized with the constraint satisfaction framework.

The constraint satisfaction framework is used to model combinatorial problems like meeting scheduling, timetabling, the stable marriages problem, and some negotiations. Many of these problems involve several participants and can involve secret constraints. A formalization that makes these requirements explicit is given by the distributed constraint satisfaction problem.

We have introduced subtleties related to how unnecessary secrets are leaked due the way in which solutions can be selected. We have proven that several simple approaches leak more secrets then necessary and we have identified some statistical attacks on such schemes.

We have then described and proved an abstract method for selecting a solution of a constraint satisfaction problem according to a uniform distribution over the set of all solutions. This distribution reduces the leaks of statistical information due to the way a solution is selected. The secret information that still can be extracted with such a method is no longer a characteristic of the solving algorithm but a characteristic of the problem.

The abstract method was then applied for designing an arithmetic circuit for a probabilistic function that can select randomly a solution for the problem. Its complexity is factorial in the size of the search space of the problem (i.e. the size of the Cartesian product of the domains of its variables).

We have then proposed a multi-party computation that also implements our abstract method by combining simulations of arithmetic circuits, with shuffling and secret share randomization using a mix-net. The obtained technique is much faster then the arithmetic circuit, its complexity being given by the size of the search space of the problem.

References

[1] M. Abadi and J. Feigenbaum. A simple protocol for secure circuit evaluation. In *STACS'88*, pages 264–272, 1988.

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

[3] C. Bessière, A. Maestre, and P. Meseguer. Distributed dynamic backtracking. In *Proc. IJCAI DCR Workshop*, pages 9–16, 2001.

[4] C. Cachin. Modeling complexity in secure distributed computing. In *Future Directions in Distributed Computing*, pages 57–61, 2003.

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

[6] D. Chaum, I. Damgard, and J. Graaf. Multiparty computations ensuring privacy of each party's input and correctness of the result. In *CRYPTO'87*, pages 87–119, 1988.

[7] Z. Collin, R. Dechter, and S. Katz. On the feasibility of distributed constraint satisfaction. In *Proceedings of IJCAI 1991*, pages 318–324, 1991.

[8] S. E. Conry, K. Kuwabara, and V. R. Lesser. Multistage negotiation for distributed constraint satisfaction. *IEEE Trans. on systems, man, and cybernetics*, 21(6):1462–1477, 1991.

[9] R. Cramer, I. Damgrd, and U. M. Maurer. General secure multi-party computation from any linear secret-sharing scheme. In *EUROCRYPT*, pages 316–334, 2000.

[10] R. Cramer, I. Damgrd, and J. B. Nielsen. Multiparty computation from threshold homomorphic encryption. In *EUROCRYPT*, pages 280–299, 2001.

[11] C. Crépeau, D. Gottesman, and A. Smith. Secure multi-party quantum computation. In *STOC*, pages 643–652, 2002.

[12] R. Dechter and A. Dechter. Belief maintenance in dynamic constraint networks. In *Proc. 7th National Conf. on Artificial Intelligence (AAAI)*, pages 37–42, St. Paul, MN, 1988.

[13] J. Denzinger. Distributed knowledge based search. IJCAI tutorial notes (MA2), 2001.

[14] S. Dziembowski. On the complexity of verifiable secret sharing and multiparty computation. In *STOC*, pages 325–334, 2000.

[15] B. Faltings. Incentive compatible open constraint optimization. In *Electronic Commerce*, 2003.

[16] E. Freuder, M. Minca, and R. Wallace. Privacy/efficiency tradeoffs in distributed meeting scheduling by constraint-based agents. In *Proc. IJCAI DCR*, pages 63–72, 2001.

[17] E. C. Freuder. A sufficient condition for backtrack-free search. In *JACM'82*, volume 29, pages 24–32, Jan 1982.

[18] Z. Galil, S. Haber, and M. Yung. Cryptographic computation: Secure fault-tolerant protocols and the public-key model. In *CRYPTO'87*, pages 135–155, 1988.

[19] M. R. Garey and D. S. Johnson. *Computers and Intractability - A Guide to the Theory of NP-Completeness*. W.H.Freeman&Co, 1979.

[20] O. Goldreich. *Foundations of Cryptography*, volume 2. Cambridge, 2004.

[21] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design. In *FOCS*, pages 174–187, Toronto, 1986.

[22] S. Haber. *Multi-party Cryptographic Computation: Techniques and Applications*. PhD thesis, Columbia University, 1988.

[23] Y. Hamadi. Optimal distributed arc-consistency. In *Proceedings of CP'99*, Oct 1999.

[24] 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.

[25] J. Kilian. Founding cryptography on oblivious transfer. In *Proc. of ACM Symposium on Theory of Computing*, pages 20–31, 1988.

[26] V. R. Lesser. A retrospective view of FA/C distributed problem solving. *IEEE Transactions on systems, man, and cybernetics*, 21(6):1347–1362, Nov/Dec 1991.

[27] J. Liu, H. Jing, and Y. Tang. Multi-agent oriented constraint satisfaction. *Artificial Intelligence*, 136:101–144, 2002.

[28] J. Liu and K. P. Sycara. Exploiting problem structure for distributed constraint optimization. In *ICMAS*, 1995.

[29] A. K. Mackworth. Consistency in networks of relations. *Artificial Intellegence*, 8(1):99–118, 1977.

[30] U. M. Maurer. Secure multi-party computation made simple. In *SCN*, pages 14–28, 2000.

[31] M. Merritt. *Cryptographic Protocols*. PhD thesis, Georgia Inst. of Tech., Feb 1983.

[32] P. Meseguer and M. Jiménez. Distributed forward checking. In *CP'2000 Distributed Constraint Satisfaction Workshop*, 2000.

[33] U. Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Sciences*, (7):95–132, 1974.

[34] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Eurocrypt'99*, volume 1592 of *LNCS*, pages 223–238, 1999.

[35] A. Sathi and M. Fox. *Distributed Artificial Intelligence*, volume 2, chapter Constraint-Directed Negotiations of Resource Reallocations, pages 163–193. Morgan Kaufmann, California, 1989.

[36] S. Sen and E. Durfee. A formal study of distributed meeting scheduling. *Group Decision and Negotiation*, 7:265–289, 1998.

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

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

[39] M.-C. Silaghi. Incentive auctions and stable marriages problems solved with n/2-privacy of human preferences. Technical Report CS-2004-11, FIT, 2004.

[40] M.-C. Silaghi, V. Rajeshirke, and J. Nzouonta. Meeting scheduling with privacy. http://www.cs.fit.edu/ msilaghi/secure/, 2004.

[41] K. Sycara, S. Roth, and M. Fox. Distributed constrained heuristic search. *IEEE Transactions on systems, man, and cybernetics*, 21(6):1446–1461, Nov/Dec 1991. To read.

[42] D. L. Waltz. *The Psychology of Computer Vision*, chapter Understanding line drawings of scenes with shadows, pages 19–91. McGraw-Hill, p.w. winston edition, 1975.

[43] A. Yao. Protocols for secure computations. In *FOCS*, pages 160–164, 1982.

[44] M. Yokoo, K. Suzuki, and K. Hirayama. Secure distributed constraint satisfaction: Reaching agreement without revealing private information. In *Proc. of the AAMAS-02 DCR Workshop*, Bologna, July 2002.

[45] M. Yokoo, K. Suzuki, and K. Hirayama. Secure distributed constraint satisfaction: Reaching agreement without revealing private information. In *CP*, 2002.

[46] Y. Zhang and A. K. Mackworth. Parallel and distributed algorithms for finite constraint satisfaction problems. In *Proc. of Third IEEE Symposium on*

Parallel and Distributed Processing, pages 394–397, 1991.

[47] H. Zhou and B. Choueiry. Characterizing the bahavior of a multi-agent search by using it to solve a tight, real-world resource allocation problem. In *CP-03W: Immediate Applications of CP*, pages 79–99, 2003.