Solving a distributed CSP with cryptographic multi-party computations, without revealing constraints and without involving trusted servers*

Marius-Călin Silaghi
Florida Institute of Technology
msilaghi@cs.fit.edu

May 23, 2003

Abstract

Everybody has its own constraint satisfaction problem, private concerns that owners prefer to keep as secret as possible. Resources may be shared and cause the need for cooperation. Here we consider the case where privacy is an overwhelming requirement and we assume that a majority of the participants are incorruptible. Namely, given n participants, at least an $n/2$ unknown subset of them are trustworthy and not corrupted or controlled by attackers. This is a common assumption in cryptographic multi-party computations where techniques exploiting such assumptions are known as *threshold schemes*.

This work shows how a random solution of the described problem can be offered with a secure protocol that does not reveal anything except the existence of the solution and tells each participant the valuations corresponding to its subproblem. The technique is based on the properties of the recent Paillier cryptosystem and needs no external arbiter.

1 Introduction

Each participating agent has his problems. Private concerns can often be formulated as constraint satisfaction problems (CSPs) and then be solved with any of the applicable CSP techniques. But life is more difficult than this: we must share our streets, busses, shops, green spaces, security, civic infrastructure, national research budget, and even our polluted air and ozone layer. We may have to share our tools and consumables and coordinate on our agendas. Therefore one has to find agreements with the others for a solution from the set of possible alternatives that satisfy his subproblem. The general framework modeling this kind of combinatorial problems is called Distributed Constraint Satisfaction.

A distributed constraint satisfaction problem (DisCSP) is defined as a set of agents, $A_1, ..., A_n$, each agent A_i willing to enforce a corresponding set of private constraints, C_i. A constraint $c \in C_i$ is defined by a predicate on a set of variables, V_c. The sets of variables involved in the constraints of different agents may not be disjoint. The union of all the variables is $\{x_1, ..., x_m\}$ and each variable x_i can take values from an associated domain $D_i = \{v_1^i, ..., v_d^i\}$. Here we assume that agents know the variables involved in the constraints of each other (false variables can be added to hide this). Instead, agents want to avoid that others find details about the exact combinations allowed by the constraints they enforce.

The methods proposed here allows the n participating agents to securely find a solution by interacting directly without any external arbiters and without divulging any secrets. A common assumption in some multi-party computations is that an unknown

*Patent pending

majority of the participating agents are trustworthy and not corrupted by any adversary. There, given a problem with n agents, no subset of t malicious agents that follow the protocol (typically called *passive adversaries*), $t<n/2$, can find anything about others' problems except what is revealed by the solution. This threshold scheme applies to our result as well.

First we describe an algorithm that computes securely one solution of a DisCSP, namely the first in the lexicographic order induced by a predefined ordering on variables and values. The returned result divulges more information than what was needed. Namely *it divulges the fact that there is no solution in tuples lexicographically ordered before the returned solution.* This information on a part of the search space is typically not a requirement in the problem descriptions and therefore may bring unnecessary and unfortunate privacy losses. Subsequent research effort led to the next described algorithm which is based on the Paillier cryptosystem and on Merritt's election protocol. This time (for satisfiable problems) a random solution is returned without leaking any information about the existence of a solution in any other search subspace. The algorithm is refutation complete and terminates. This is a polynomial space complexity complement of the methods we proposed in previous documents and which are based on differential bids. It can deal with both satisfaction and optimization problems.

Our protocol is based on three cryptographic techniques: Paillier public key cryptosystem [Pai99], Shamir's secret sharing [Sha79], and Merritt's election protocol [Mer83]. After an example, we start our presentation by first introducing these elements. Next we describe the technique that we developed and we end with a discussion about the motivations and potential of other design choices.

2 Example

A common problem, meeting scheduling, consist of finding allocations for a set of shared resources like:

- Place of the meeting.
- Date of the meeting.
- Topics to be discussed.
- Tools to be brought by participants.
- Hotel availability.
- Transportation availability.

The problem can be stated by using a set of first order logic predicates whose variables' values represent possible allocations of the shared resources. Many research communities treat this case as a particular problem case of larger theories: artificial intelligence will refer to it as a constraint satisfaction/optimization problem, operations research calls this problem a particular case of integer programming.

Problem Formulation When we try to solve a meeting scheduling problem, the first step is to gather all the data: resources (variables), possible allocations (variable domains), and the logic predicates that have to be satisfied. In mathematically approaching the problem it is often tempting to jump too easily over this step. However, a serious effort to bring it into practice will encounter here the most difficult problem: *How to elicit the predicates in which the participants are interested?* Let us see a couple of these predicates:

- Mr. A needs that the meeting place should be reachable from his location with the 500$ available through his funding. He also needs an overhead projector.
- Mr. B is supported by a funding paying only national travels up to 300$.
- Mr. C works for a famous company that requests his employees to sustain its image by only using at least 3 stars hotels, but cannot pay more than 100$ per night and 900$ for transportation.
- Mr. E is the manager of a large chain of hotels that cooperates with this conference. He wants to make sure that he has the needed number of rooms and that the price chosen for rooms does not lead to financial losses.

- Mr. F would like to also cover a holiday at a sea or a lake with the funding for the conference, for which the ticket should anyhow be cheaper than 400$
- Mrs. E has no available funding but she can use her United frequent flyer program, and she also wants to avoid a conference during the academic year.

As we can see from the examples before, some of the constraints of the participants can be made public easily (e.g. the requirement by Mr. A of having an overhead projector). However, some of the requirements should (if possible) be concealed (e.g. Mr. F's, Mrs. E's), while some must not be divulged (e.g. the ones of Mr. C).

Two conclusions can be drawn from this example:

- The divulgence of some predicates should be avoided. One has to keep track of who enforces private information. To help hiding even the presence of secrets (steganography) the assumption will have to be that everybody has private information. To ensure satisfaction of the private requirements, the link between the identity of the participants and the problem cannot be lost.
- Some predicated can be easily elicited from participants (e.g. Mr. A's). It is efficient to elicit such predicates and to use them in preprocessing the problem. Such preprocessing can consist of finding good cutting planes or of some local consistency technique.

The first of the aforementioned conclusions states that the identity of the participants generating the problem should become a part of the problem description. This is the reason why DisCSPs must be defined separately from simple constraint satisfaction problems. *A DisCSP $P'(A, X, D, C)$ is a finite constraint satisfaction problem $P(X, D, C)$ where each predicate c in C is known only to a single participant, A_i. $A_i \in A$ where A is the set of all participants.* Note that any predicate $p(\epsilon)$ can be seen as a function $p : X \to \{0, 1\}$ where 0 stands for unsatisfied and 1 stands for satisfied.

3 Public key cryptosystems

By *cryptosystem* (or *cryptographic system* or *cipher*) one refers a system used for encryption and decryption of data. One always stresses that a cipher has two distinct elements: encryption and decryption. The cipher is also referred to as a pair of algorithms E, D together with their keys, K_E, K_D. The input to an encryption algorithm is called plaintext and the output is the ciphertext. Given a plaintext m, we obtain the ciphertext by $E_{K_E}(m)$. Decryption retrieves the plaintext $m = D_{K_D}(E_{K_E}(m))$. A public key cryptosystem is a cipher where the encryption and decryption use different keys. Theoretically, such asymmetric cryptosystems may still require that both keys are secret, but particular importance is given to ciphers where one of the two keys is public and the other one is the secret of a single user. It should be intractable to obtain the secret key from the public key. When the encryption key is the private one then the system can be used for generating digital signatures. The exchange of secrets is achieved with public key cryptosystems where the encryption key, K_E, is public.

The first published public key cryptosystem is the Diffie-Hellman key exchange algorithm [DH76], a version of which is also known under the form of the ElGamal cryptosystem [ElG84]. Next developed techniques, also known as knapsack cryptosystems, were based on the well known intractability of the Knapsack problem. Some knapsack cryptosystems had other nice properties, we mean homomorphisms of type $\exists f, \forall m_1, m_2, f(E_{K_E}(m_1), E_{K_E}(m_2)) = E_{K_E}(m_1 + m_2)$ which are of particular interest for this work [MH78]. Unfortunately, many knapsack algorithms were proven to be insufficiently secure.

3.1 Paillier cryptosystem

Our technique needs an encryption function $E_{K_E} : S_P \to S_C$ (S_P is the domain of the plaintext and S_C the domain of the ciphertext) that has the $(+, \times)$-homomorphic encryption property:

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

This property is offered by the Paillier cryptosystem [Pai99]. The Paillier cryptosystem uses modular arithmetic, mod n and mod n^2, with n chosen as product of two large primes $n = pq$ (similar to RSA). The prime numbers p and q have to be chosen very large and approximatively of the same number of digits. Primes can be found efficiently with acceptable factor of certitude by using the Miller-Rabin algorithm [Knu98]. The recent technique of Agarwal et.al. [ASK02] allows for an exact test with polynomial cost for the primality of any given number.

We recall that $\mathbb{Z}_n$ is the set of residues mod n. For the encryption of a plaintext m, $m \in \mathbb{Z}_n$, Paillier proposed to select a number g, to randomly chose a number $r \in \mathbb{Z}_n^*$, and then to call the next function returning the ciphertext c:

$$E_{g,n} : \mathbb{Z}_n \times \mathbb{Z}_n^* \rightarrow \mathbb{Z}_{n^2}^* \quad E_{g,n}(m,r) \stackrel{\text{def}}{=} g^m r^n \mod n^2.$$

g's order is a nonzero multiple of n (i.e. $\exists \alpha \in \mathbb{Z}_{n^2}^*, g^{\alpha n} \equiv 1 \mod n^2$). The requirement[1] on g is needed in order to ensure that $E_{g,n}$ is bijective (having an inverse). For decryption of a ciphertext c, $c < n^2$, the computation is as follows:

$$m = \frac{\frac{(c^{\lambda(n)} \mod n^2)-1}{n}}{\frac{(g^{\lambda(n)} \mod n^2)-1}{n}} \mod n$$

Carmichael's lambda function at n, $\lambda(n)$, is the size of the largest cyclic subgroup of the multiplicative group modulo n, $\mathbb{Z}_n$. $\lambda(n)$ returns therefore the smallest number λ such that for any m with $\gcd(m,n)=1$ the congruence $m^\lambda \mod n = 1$ holds, where gcd stands for *greatest common divider*. Its definition for a number N with prime factorization $N = p_1^{a_1} ... p_k^{a_k}$ is $\lambda(N) = \text{lcm}[\lambda(p_1^{a_1}), ..., \lambda(p_k^{a_k})]$, where lcm stands for *least common multiple* and

$$\lambda(p_i^{a_i}) \stackrel{\text{def}}{=} \begin{cases} 2^{a_i-2} & \text{if } p_i = 2 \text{ and } a_i > 2, \\ p_i^{a_i-1}(p_i-1) & \text{otherwise} \end{cases}$$

For our case with n=pq, we get $\lambda(n) = \text{lcm}(p-1, q-1)$. The n^{th} residuosity class of w with respect to g, denoted $[\![w]\!]_g$, is the unique integer $x \in \mathbb{Z}_n$ for which $\exists y \in \mathbb{Z}_n^*$, such that $g^x y^n \mod n^2 = w$ (i.e. $[\![w]\!]_g$ is the decrypted w). The Composite Residuosity Class Problem is the problem of computing $[\![w]\!]_g$ given n and its complexity was shown to depend neither on w nor on g, but only on n. The complexity of the Composite Residuosity Class Problem is proven higher than the complexity of computing n-th residuosity modulo n^2 (i.e. $\exists y \in \mathbb{Z}_{n^2}^*$ with $z = y^n \mod n^2$, for given z), which is believed intractable [Pai99].

The proof of the correctnes of the Paillier decryption is immediate based on the observations that: $[\![w]\!]_{g_2} \equiv [\![w]\!]_{g_1} [\![g_2]\!]_{g_1}^{-1} \mod n$, $[\![g]\!]_{1+n} \equiv [\![1+n]\!]_g^{-1} \mod n$, and $\frac{w^{\lambda(n)} - 1 \mod n^2}{n} \equiv \lambda(n) [\![w]\!]_{1+n} \mod n$. The $(+,\times)$-homomorphic property of the Paillier cryptosystem results from the fact that $[\![w_1 w_2]\!]_g \equiv [\![w_1]\!]_g + [\![w_2]\!]_g \mod n$. In consequence, $\forall r_1, r_2, \exists r$, such that $E_{g,n}(m_1, r_1) E_{g,n}(m_2, r_2) = E_{g,n}(m_1 + m_2, r)$.

Example 1 *If p=5, q=3, then n=15, $\lambda(n)$=4, n^2=225, g can be 11 ($11^{4*15} \equiv 1 \mod 225$). If the encryption choses twice r=4 then $E_{11,15}(1,4) = 164$, $E_{11,15}(2,4) = 4$. Note that $164 * 4 = 656 \equiv E_{11,15}(1+2, 4*4) \mod 225$.*

In the following we only use Paillier encryption. The Paillier encryption presented above returns a ciphertext whose size is double the size of the plaintext. Paillier observed that the encryption being bijective, one can reconstruct both m and r from $E_{g,n}(m,r)$. As a result, by splitting m into its quotient m_1 and residue m_2 modulo n, one gets a second cryptosystem,

$E'_{g,n}(m) \stackrel{\text{def}}{=}$

$E_{g,n}(m_1 \stackrel{\text{def}}{=} floor(\frac{m}{n}), m_2 \stackrel{\text{def}}{=} (m \mod n))$, $m \in \mathbb{Z}_{n^2}$ where the ciphertext has the same size as the maximum size of the plaintext. m_2 is reconstructed as $m_2 = (E'_{g,n}(m) g^{-m_1} \mod n)^{n^{-1} \mod \lambda(n)} \mod n$.

3.2 Merritt's election protocol

We randomize the extraction of the solution by letting agents to jointly generate a secret reformulation of the problem. In order to destroy the visibility of the relations between the initial problem formulation

[1] It can be tested by checking that $\gcd(\frac{g^{\lambda(n)} - 1 \mod n^2}{n}, n)$=1.

and the formulation actually used in computations one can exploit random joint permulations that are not known to any participant. Such permutations appeared in Merritt's vote counting protocols [Mer83]. Here we reformulate the initial problem by reordering the values and the variables. Several descriptions and versions of Merritt's election protocol exist. We explain first this algorithm for values with one index. We use the index notations described in [GB96] upon the version used in [Gen95], with additional slight rephrasing for the current application and notations.

Merritt's election protocol was initially meant for accounting the votes during electronic elections and ensures the privacy of the relation vote-elector by reordering (shuffling) the indexes of the votes. The shuffling is obtained by a chain of permutations (each being the secret of an election center) on the encrypted votes. n agents, $A_1, ..., A_n$, are ordered in a chain (called in the following, Merritt chain). Each agent A_i distributes 2 public encryption algorithms E_{g_i,n_i} and $E_{g_{\cdot i},n_{\cdot i}}$, denoted for simplicity E_i, E_i^1, and keeps corresponding private decryption functions D_i and D_i^1. Similarly, E'_{g_i,n_i} is denoted by E'_i and its decryption function by D'_i. Sometimes we write $E_i(m)$ instead of $E_i(m,r)$, to simplify the notation. Each agent A_j that announces a value v_j for an index j chooses a random number h_j and random numbers $r_{1,j}, ..., r_{n,j}$ and computes:

$$E_1(E_2(...E_n(E'_1(E'_2(...E'_{n-1}(E_n^1(v_j,h_j))...)),r_{n,j})...,r_{2,j}),r_{1,j}) = y_{1,j}.$$

Given a function q, let $\{q(k)\}_k$ denote the vector of values of $q(k)$ taken in ascending order of k. The $y_{1,j}$ values gathered in a vector $\{y_{1,j}\}_j$ according to their second index, j, are posted through the chain of participants in order from A_1 to A_n. Each A_i choses a secret random permutation π_i : $[1..n] \rightarrow [1..n]$. We also define $\pi_i(\{x_k\}_{k\in[1..n]}) \stackrel{\text{def}}{=} \{x_{\pi_i^{-1}(k)}\}_{k\in[1..n]}$. After receiving $\{y_{i,j}\}_j$, A_i broadcasts $\{y_{i+1,j'}\}_{j'} = \pi_i(\{D_i(y_{i,j})\}_j)$ and discards all $r_{i,j}$. A_n broadcasts $\{y_{n+1,j^{(n)}}\}_{j^{(n)}}$:

$$y_{n+1,j^{(n)}} = E'_1(E'_2(...E'_{n-1}(E_n(y_{n+1,j},h_j))...))$$

Now values have been shuffled as the relation $(j,j^{(n)})$ was lost. The shuffled value can be found immediately by an additional decryption round in the order $A_1 \rightarrow A_2 \rightarrow ... \rightarrow A_n$. A_1 sends $\{y'_{2,j^{(n)}}\}_{j^{(n)}}$, $y'_{2,j^{(n)}} = D'_1(D'_1(y_{n+1,j^{(n)}}))$. Each agent $A_i, 1<i<n$ computes $y'_{i+1,j^{(n)}} = D'_i(D'_i(E'_{i-1}(y'_{i,j^{(n)}})))$. A_n broadcasts the pairs $\{v_j, h_j\} = D_n^1(E'_{n-1}(y'_{n,j^{(n)}})))$. Each A_j checks for the presence of its h_j.

Here, each E_i must have a different modulus n_i. In the rest of this paper the algorithms work with encryption functions having all the same modulus.

3.3 Shamir's secret sharing

We intend to have the participants interact directly (without intermediaries or trusted arbiters) to perform all the computations required for solving this problem. But the parameters used in their computations and representing other agents' problems should be random numbers, otherwise secrets are leaked.

Let us give a simple example of such techniques. Three faculty members, A_0, A_1, A_2, want to compute the average of their wages x_0, x_1, x_2 without revealing any one of them. Each professor A_i generates two random numbers, $r_{i,-1}, r_{i,1}$. A_i sends each $r_{i,j}$ to $A_{(i+j) \mod 3}$ through an encrypted channel. Each A_i computes

$$r_i = x_i + r_{((i+1) \mod 3),-1} + r_{((i-1) \mod 3),1} - r_{i,-1} - r_{i,1}$$

and publishes r_i. Their average wage is $(r_0+r_1+r_2)/3$ and no particular wage is revealed except if communication is intercepted, or if two professors collude. We show in this paper how such computations can be extended to solve general CSPs. In the simple example above, the secrets were shared among participants in the computation by using random numbers. A slightly more careful secret sharing technique is famous due to its properties that allow for some more general computations. This is Shamir's secret sharing that we present now.

Consider that a secret s should be split and shared among n participants in such a way that any t of them can reconstruct s but no $t-1$ participants could infer anything. A classical example is the access to the nuclear bombs in former USSR where allegedly out of

three political leaders, any two could have launched a nuclear weapon but no single one could do anything.

The idea of Shamir comes from the observation 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. This 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 s to n participants $A_1, A_2, ..., A_n$, 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. Each participant A_i is then communicated over a secure channel (e.g. encrypted with E_i) the value of the pair $(k_i, f(k_i))$.

Let all the values of $a_i, s, k_i, f(x)$ belong to a field $\mathcal{S}$. The previous claim about the security of s given the valuation of f in less than t distinct points, $f(k_{w_1})=l_1, ..., f(k_{w_{t-1}})=l_{t-1}$ holds only if $\forall s \in \mathcal{S}$, $\exists a_1, ..., a_{t-1} \in \mathcal{S}$ defining a polynomial f'' such that $\forall i, 0<i<t, f''(k_{w_i}) = l_i$. This condition holds when S is a Galois Field. Note that on computer implementations S cannot be the set of reals $\mathbb{R}$ since not all reals are representable in available floating point representations. Therefore, at least when the domain of the secret is finite, it is a good choice to have S chosen as the finite field of order p, $\mathbb{Z}_p$ (aka. $GF(p)$) for some prime p. We say that we achieved a (t, n) secret sharing threshold scheme.

Multi-party computations Once secret numbers are split and distributed with a (t, n) threshold scheme, distributed computations of an arbitrary agreed function (from a certain class) can be performed over the distributed shares, in such a way that all results remain shared secrets with the same security properties (the number of supported colluders, $t-1$) [Yao82]. For Shamir's secret sharing [Sha79], when the same distribution of k_i is used for all secrets, the computation of the *sum* function can be efficiently implemented as a sum of the shares for corresponding samples. Consider two shared secrets, s_1 and s_2, shared using the polynomials:

$$l(x) = s_1 + \sum_{i=1}^{t-1}((a_i)x^i), \quad (1)$$

$$g(x) = s_2 + \sum_{i=1}^{t-1}((b_i)x^i), \quad (2)$$

where each agent A_i knows $l(k_i)$ and $g(k_i)$. If each agent A_i adds the shares it has, $l(k_i)+g(k_i)$, it obtains a point on $h(x) = l(x) + g(x)$, namely $h(k_i)$. The secret values $h(k_1), ..., h(k_n)$ define a valid sharing of the secret s_1+s_2 with a (t,n) scheme:

$$h(x) = (s_1+s_2) + \sum_{i=1}^{t-1}((a_i+b_i)x^i), \quad (3)$$

Multiplication of a shared secret s_1 with a public value k is easy. One only multiplies each share with k. The n obtained shares are indeed the valuations of

$$kl(x)=(ks_1)+\sum_{i=1}^{t-1}((ka_i)x^i), \quad (4)$$

in the points $k_1, ..., k_n$, and this is a valid (t, n) sharing of the secret ks_1.

Multiplication of two shared secrets is more complex and recent research has focused on improving its efficiency [HMP00]. Consider again the two sharing of secrets in Equations 1 and 2. If each agent A_i multiplies the shares it has, it obtains a point on $h(x) = l(x)g(x)$, namely $h(k_i)$.

$$h(x) = s_1 s_2 + \sum_{i=1}^{2t-2}(c_i x^i), \quad (5)$$

where $c_i = s_1 b_i + s_2 a_i + \sum_{y=1}^{i}(a_y b_{i-y})$. Notice that whenever $2t-1 \leq n$, then Equation 5 and $h(k_1), ..., h(k_n)$, is a valid $(2t-1, n)$ sharing of the secret $s=s_1 s_2$. But what we want to get is a (t, n) threshold scheme sharing. One noticed that the computation needed to reconstruct s out of the secrets $h(k_1), ..., h(k_n)$ using Cramer's rule involves only additions of secrets and multiplications of secrets with public values:

$$s = (\prod_{j=1}^{j=T} k_{u_j}) \sum_{i=1}^{i=T} (\frac{h(k_{u_i})}{k_{u_i} \prod_{m \in [1..i-1,i+1,...T]} (k_{u_m} - k_{u_i})}) \quad (6)$$

where $\prod_j(k_j)$ and $\frac{(-1)^i}{k_i \prod_{(m>n)\wedge(i=m\vee i=n)}(k_m - k_n)}$ are public. Therefore, let each agent A_i share the secret product it knows, $h(k_i)$, with a (t, n) threshold scheme version of Shamir's secret sharing. Then, Equation 6 can be computed with repeated applications of the aforementioned techniques for sum and multiplication with public values.

Any function based on additions and multiplications can therefore be *compiled* unto secure cryptographic protocols. When a certain function has to be computed on shared secrets, all its steps can be performed using secure protocols for the corresponding steps. This can be done in such a way that any ($<t$) colluding participants that cannot directly find the initial secrets, cannot find anything else than the *official output* of the protocol. Techniques for compiling protocols for honest parties when less than n/2 players cheat, and that are based on intractability assumptions, are proposed in [GMW86, GMW87]. [CCD88a, CCD88b, BOGW88] show compiling techniques for cases where less than n/3 players cheat and that are secure without intractability assumptions. [BOGW88] details a quite efficient technique resisting less than n/2 cheaters, when all participants comply with the protocol.

Secret reconstruction A secret computed by multi-party computations on a (t,n) Shamir sharing can be reconstructed if t shares are revealed. In general, revealing the t shares of the computed shared secret does not allow to compute anything about the shares of the initial secret parameters used in the multi-party computation.

Nevertheless, specially for some techniques using verifications of some partial results in the computation, revealing the secret shares of the partial secret results may not be zero-knowledge. Now we show a technique that allows to reconstruct a shared secret without revealing any of its shares.

Let us consider the sharing $(s_1, ..., s_n)$ of a secret s where we want to reconstruct s. Let each agent A_i participating in computation secretly generate a (t,n) Shamir sharing of 0, $(z_1^i, ...z_n^i)$ and distribute z_j^i to A_j on a secure (encrypted) channel. Each agent produces a zero-knowledge proof that his shares stand for the secret 0, using one of the methods in the literature [Sta98, Gen95]. Each agent A_i computes then $s_i + \sum_{k \in 1..n} z_i^k$ and publishes the result. The result is a sharing of s, while the shares themselves lost any relation to the initial shares.

4 Searching for the first solution

As mentioned in previous sections, any algorithm that can be implemented solely with additions and multiplications (without branching with conditions on secrets) can be straightforwardly translated into a secure protocol with threshold schemes. We show such an algorithm for finding the first solution (given a lexicographic order) of a general CSP. It translates automatically to DisCSPs. Subsequent sections extend this algorithm to extract randomly a solution.

Imagine we want to solve the CSP $P = (X, D, C)$ where X is a set of variables $x_1, x_2, ...x_m$, C is a set of functions with results in the set $\{0, 1\}$ (the constraints), and D a set of finite discrete domains for X. The domain of x_i is $D_i \in D$, whose values are $v_1^i, v_2^i, ..., v_d^i$ (i.e. all domains are extended to d values). The solutions of P are the valuations ϵ (of type $\langle (x_1, v_{\epsilon_1}^1), ..., (x_m, v_{\epsilon_m}^m) \rangle$) with $\sum_{c \in C} c(\epsilon_{|_{V_c}}) = |C|$. Here $\epsilon_{|_{V_c}}$ denotes the projection of ϵ to the variables in V_c. All constraints are extended (e.g. with redundant variables) to obtain a constraint hypergraph [Tsa93] that is symmetric (in worst case each constraint will involve all variables, and constraints of a given agent can be composed).

Definition 1 *The* **first solution** *of P given a total order $\prec_1$ on its variables X, and a total order on its values $\prec_2$ is the first among solution tuples when these are ordered with the lexicographical order induced by $\prec_1$ and $\prec_2$.*

Often, one has to restrict a problem by adding additional constraining functions. We define the union

between a problem $P = (X, D, C)$ and a function c ($c : X' \rightarrow \{0,1\}$, $X' \subseteq X$) as the problem $P = (X, D, C \cup \{c\})$.

$$(X, D, C) \cup c \stackrel{\text{def}}{=} (X, D, C \cup \{c\})$$

Let us imagine that we have available a function `satisfiable`:$CSP \rightarrow \{0,1\}$ with the next property (an example is given later):

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

We will design now a set of functions: $f_1, f_2, ..., f_m$, $f_i : CSP \rightarrow \mathbb{N}$, such that each f_i will return the index of the value of x_i in the first solution (between 1 and d), or 0 if no solution exists.

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

Therefore, we first design the functions $g_{i,1}, g_{i,2}, ..., g_{i,d}$. $g_{i,j} : CSP \rightarrow \{0,1\}$.

$$g_{i,j}(P) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } (P \cup_{k<i} \Lambda^*_{k,P}) \text{ has solution for } x_i = v_j^i \\ 0 & \text{if } (P \cup_{k<i} \Lambda^*_{k,P}) \text{ is infeasible for } x_i = v_j^i \end{cases}$$

$g_{i,j}(P)$ is 1 if and only if the problem obtained by adding to the CSP P the function

$$\Lambda_i^j(\epsilon) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } x_i = v_j^i \text{ in valuation } \epsilon \\ 0 & \text{if } x_i \neq v_j^i \text{ in valuation } \epsilon \end{cases} \quad (7)$$

that selects the value v_j^i for x_i, and $\forall k, 0<k<i$, the functions $\Lambda^*_{k,P}$

$$\Lambda^*_{k,P}(\epsilon) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } x_k = v^k_{f_k(P)} \text{ in valuation } \epsilon \\ 0 & \text{if } x_k \neq v^k_{f_k(P)} \text{ in valuation } \epsilon \end{cases} \quad (8)$$

instantiating previous variables to their values in the first solution, is satisfiable. $v^k_{f_k(P)}$ is the $f_k(P)^{th}$ value of x_k, the value that x_k takes in the first solution. Namely, a simple implementation of $g_{i,j}$ is:

$$g_{i,j}(P) = \texttt{satisfiable}(P \cup \Lambda_i^j \cup_{k<i} \Lambda^*_{k,P}) \quad (9)$$

This is a recursion and is possible by first computing the value of the first variable in the first solution, f_1, based on $g_{1,j}$ as described next. Based on the result one can compute all $g_{2,j}$. Then, one can now compute f_2. The recursion continues with all $g_{i,j}$ that help in computing f_i for all i up to m.

To help computing f_j, we define the functions $t_{j,1}, t_{j,2}, ..., t_{j,d}$, $t_{j,i} : CSP \rightarrow \{0,1\}$. $t_{j,k}(P)$=1 if and only if in a chronological backtracking search tree **no** satisfiable subtree exists under any node x_j=v_k^j, k<i, when previous variables are assigned according to the values in the first solution:

$$t_{j,i}(P) = \prod_{0<k<i} (1 - g_{j,k}(P)) \quad (10)$$

Functions $t_{j,i}$ are obtained incrementally as follows:

$$t_{j,1}(P) = 1 \quad (11)$$
$$t_{j,i}(P) = t_{j,i-1}(P) * (1 - g_{j,i-1}(P)) \quad (12)$$

Once $t_{j,i}$ have been computed for all i, one can compute the index of the value of x_j in the first solution, namely f_j:

$$f_j(P) = \sum_{i=1}^{d} i * (g_{j,i}(P) * t_{j,i}(P)) \quad (13)$$

Lemma 1 *The functions g and f given by Equations 9 and 13 correspond to their definition.*

Proof The properties can be checked recursively starting with $g_{1,k}$ and f_1. After computing $g_{i,k}$ and f_i one can check $g_{i+1,k}$ followed by f_{i+1}, etc...

Implementing `satisfiable()`: Let $SS(P)$ be the ordered set of all tuples in the Cartesian-product of the domains of $P = \langle X, D, C \rangle$. Each function c in the set of constraints C can be transformed (by adding redundant parameters and reordering them) to a function, $G_c : SS(P) \rightarrow \{0,1\}$. The secret parameters of the computation are the various values $c(\varepsilon)$ where ε are tuples in the domain on which the corresponding c is defined. Let us define the function p, $p : SS(P) \rightarrow \{0,1\}$, defined as $p(\epsilon) = \prod_{c \in C} G_c(\epsilon)$.

procedure satisfiable(P) do

1. ϵ=first tuple; a=0; b=1;
2. loop: $a = a + p(\epsilon) * b$
3. if (problem space exhausted), then terminate and return a.
4. $b = b * (1 - p(\epsilon))$
5. ϵ=next tuple;
6. goto loop

Algorithm 1: satisfiable(P).

function value-to-unary-constraint1(v, M)

1. Jointly, all agents build a vector u, $u = \langle u_0, u_1, ..., u_M \rangle$ with $4M-2$ multiplications of shared secrets by computing:
   1. the shared secret vector: $\{x_i\}_{0 \leq i \leq M}$, x_0=1, x_{i+1}=$x_i * (v-i)$
   2. the shared secret vector: $\{y_i\}_{0 \leq i \leq M}$, y_M=1, y_{i-1}=$y_i * (i-v)$

   then, $u_k = \frac{1}{k!(M-k)!}(v-k+1)x_k y_k$, where $0! \stackrel{\text{def}}{=} 1$.
2. Return u.

Algorithm 2: Transforming secret value $v \in \{0, 1, 2, ..., M\}$ to a shared secret unary constraint. This is a multi-party computation using the shares of secret v.

Now we can finally design an implementation for `satisfiable` (see Algorithm 1).

$$\texttt{satisfiable}(P) = \sum_{\epsilon_i \in SS(P)} (p(\epsilon_i) \prod_{k<i} (1 - p(\epsilon_k)))$$

Proposition 1 *Given the previous definitions of the functions p,* `satisfiable`*, $g_{i,j}$, and f_i, and a problem P, the vector $\{v^i_{f_i(P)}\}_{i \in [1..m]}$ defines a solution of P (the first one).*

Proof Immediate from the definition of the functions f.

To avoid storing all the tuples in memory, the function `satisfiable` is computed similarly with the functions $g_{i,j}$, namely by using two temporary values.

Remark 1 *The computation of each $f_i(P)$ requires only additions and multiplications of the secrets. There exist some branches in loops but they do not involve evaluations of secrets (they are equivalent to completely unfolded versions).*

Remark 2 *Whenever an element of the vector $\{f_i(P)\}_{i \in [1..m]}$ is 0, the computation can be stopped since P is infeasible (Algorithm 3, step 2).*

procedure SecureSatisfaction do

1. Securely distribute to each agent A_j (e.g. by encryption) Shamir shares of the feasibility of each local tuple ε^i_k of A_i: $(\varepsilon^i_k, s^j_{i,k})$. $s^j_{i,k}$ is A_j's share of the secret $c(\varepsilon^i_k)$.
2. Jointly compute `satisfiable(P)` (using Algorithm 1 compiled into a multi-party computation). If P is not satisfiable (result 0), terminate with failure.
3. $j = 1$
4. Compute in parallel all $g_{j,k}$ (Eq. 9). Functions Λ^k_j are public, therefore one simply sets $p(\epsilon)$ to 0 when $\Lambda^k_j(\epsilon)$ is 0, and disregards Λ^k_j otherwise.
   Compute $t_{j,k}(P)$ for all k, from 1 to d (Eq. 11).
5. Compute $f_j(P)$ (Eq. 13).
6. The shared secret $f_j(P)$ will be transformed in a unary constraint extensively represented by a vector f'_j of size d. The $f_j(P)^{th}$ element denoted $f'_j[f_j(P)]$ is 1 and all the other elements are 0. This is achieved by the call `value-to-unary-constraint1(`$f_j(P)-1$`,`$d-1$`)`. The function `value-to-unary-constraint1` doing this is shown in Algorithm 2.
   f'_j is used to evaluate $\Lambda^*_{j,P}(\epsilon)$ in future steps 4 by returning the k-th element of the vector, $f'_j[k]$, for a parameter tuple ϵ having $x_j = v^j_k$.
7. if $j = m$, then terminate algorithm:
   7a For all k, let $f_k(P)$ be revealed to the owners of the x_j variable (agents that have functions involving x_j). This is done by reconstructing the corresponding secrets from their shares with Shamir's technique.
8. $j = j + 1$
9. goto step *4*

Algorithm 3: Algorithm performed by each agent A_i for finding a solution satisfying conditions where g functions are computed in parallel. It is possible to also compute them sequentially with lower space complexity.

Remark 3 *To compute the whole vector $\{f_i(P)\}_{i \in [1..m]}$, some constraints based on secrets also have to be dynamically shared according to*

Shamir's technique (see Equations 7, 8, 9). This is done according to Algorithm 2.

The secure algorithm obtained by compiling the computation of $\{v^i_{f_i(P)}\}_{i\in[1..m]}$ to a secure multi-party computation with threshold schemes (see previous section) is referred to as SecureSatisfaction. The steps that any agent has to follow here are given in Algorithm 3.

Algorithm 4 is just a small optimization of Algorithm 3, but we will see soon while this optimization is useless given some extensions.

Example 2 *Take a DisCSP with* n=3, d=2, *and* m=2. A_1 *(secretly) does not want* $((x_1{=}v^1_1),(x_2{=}v^2_1))$ *and* A_2 *(secretly) does not want* $((x_1{=}v^1_2),(x_2{=}v^2_2))$... *For simplicity, 0 is always shared as* $3x{+}0$ *and 1 as* $4x{+}1$, mod 7. *A tuple* $((x_i{=}v^i_a),(x_j{=}v^j_b))$ *is denoted in the following by* $\binom{i\,j}{a\,b}$.

During step 1 in Algorithm 3, A_2 *generates:* $(\binom{12}{11},E_1(3))$, $(\binom{12}{12},E_1(5))$, $(\binom{12}{21},E_1(5))$, $(\binom{12}{22},E_1(5))$, *that he sends to* A_1. A_2 *also generates for itself,* $(\binom{12}{11},6)$, $(\binom{12}{12},2)$, $(\binom{12}{21},2)$, $(\binom{12}{22},2)$, *and for* A_3*:* $(\binom{12}{11},E_3(2))$, $(\binom{12}{12},E_3(6))$, $(\binom{12}{21},E_3(6))$, $(\binom{12}{22},E_3(6))$.

A_1 *generates for itself:* $(\binom{12}{11},5)$, $(\binom{12}{12},5)$, $(\binom{12}{21},5)$, $(\binom{12}{22},3)$, *and also the shares to be delivered to* A_2 *and* A_3,
$(\binom{12}{11},E_2(2))$, $(\binom{12}{12},E_2(2))$, $(\binom{12}{21},E_2(2))$, $(\binom{12}{22},E_2(6))$,
$(\binom{12}{11},E_3(6))$, $(\binom{12}{12},E_3(6))$, $(\binom{12}{21},E_3(6))$, $(\binom{12}{22},E_3(2))$...

During step 2 in Algorithm 3 the agents jointly compute by multi-party multiplication (see Section 3) a sharing of the following secrets: $p(\binom{12}{11})=0$, $p(\binom{12}{12})=1$, $p(\binom{12}{21})=1$, $p(\binom{12}{22})=0$. *They are used in Algorithm 1 where we have four loops with the following states in its step 3:* $(a=0,b=1)$; $(a=1,b=1)$; $(a=1,b=0)$; $(a=1,b=0)$.

Now Algorithm 3 enters a cycle that will be run two times: In step 4 the agents compute jointly shares of the secrets: $g_{1,1}=1$, $g_{1,2}=1$. *The agents also compute sharing of secrets:* $t_{1,0}=1$, $t_{1,1}=0$.

Agents can now compute at step 5 in Algorithm 3: $f_1(P)=1*1*1+2*0*0=1$.

procedure SecureSatisfaction1 do

1. Securely distribute to each agent A_j (e.g. by encryption) Shamir shares of the feasibility of each local tuple ϵ^i_k of A_i: $(\epsilon^i_k, s^j_{i,k})$. $s^j_{i,k}$ is A_j's share of the secret $c(\epsilon^i_k)$.
2. Jointly compute `satisfiable(P)` (using Algorithm 1). If P is not satisfiable (result 0), terminate with failure.
3. $j=1$
4. Compute in parallel all $g_{j,k}$ (Eq. 9). The functions Λ^k_j are publicly known, therefore one simply sets $p(\epsilon)$ to 0 when $\Lambda^k_j(\epsilon)$ is 0, and disregards Λ^k_j otherwise.
   Compute $t_{j,k}(P)$ for all k, from 1 to $|D_j|$ (Eq. 11).
5. Compute $f_j(P)$ (Eq. 13).
6. if $j=m$, then terminate algorithm:
   6a For all k, let $f_k(P)$ be revealed to the owners of the x_j variable (agents that have functions involving x_j). This is done by reconstructing the corresponding secrets from their shares with Shamir's technique.
7. The shared secret $f_j(P)$ will be transformed in a unary constraint extensively represented by a vector f'_j of size d. Its $f_j(P)^{th}$ element $f'_j[f_j(P)]$ is 1 and all the other elements are 0. This is achieved by the call `value-to-unary-constraint(`$f_j(P){-}1$`,`$d{-}1$`)`, followed by multiplying each element of the returned vector by $\frac{1}{((-1)^{\sigma_j}(\sigma_j!)^2)}$ where $\sigma_j=(d{-}1)$. The function `value-to-unary-constraint` doing this is shown in Algorithm 2. The obtained vector is used to evaluate $\Lambda^*_{j,P}(\epsilon)$ in future steps 4 by returning the k-th element of the vector for a parameter tuple ϵ having $x_j=v^j_k$.
8. $j=j+1$
9. goto step *4*

Algorithm 4: Algorithm followed by each agent A_i for finding a solution satisfying conditions where g functions are computed in parallel. It is possible to also compute them sequentially with lower space complexity.

At step 6 in Algorithm 3 one applies Algorithm 2: value-to-unary-constraint(1-1,2-1) that translates the shared secret $f_1(P)$ *into a vector of shared secrets:* $\langle 1,0\rangle$ *by first computing* $\langle 0,-1\rangle$, *then in a second step* $\langle 1*(-1)*1,(-1+1)*(-1-1)*(-1+1)\rangle$ *obtaining*

$\langle -1, 0\rangle$ *and at the end by scalarly dividing with* $\sigma_1 = -1$. *This shares* $\Lambda^*_{1,P}$.

In the following a new loop is started with step 4 by computing shared secrets: $g_{2,1} = 0$, $g_{2,2} = 1$. *The agents also compute sharing of secrets:* $t_{1,0} = 1$, $t_{1,1} = 1$.

Agents can compute at step 5 in Algorithm 3: $f_2(P) = 1*0*1+2*1*1 = 2$. *Now the solution is recovered unto interested agents:* $f_1(P) = 1, f_2(P) = 2$.

Alternative Implementation of `satisfiable` Let $q(\epsilon)=\sum_{c\in C} c(\epsilon)$. A solution of a CSP (X,D,C) is a valuation ϵ with $q(\epsilon)=|C|$. So, one can (less efficiently) compute:
$p(\epsilon) = (q(\epsilon)-|C|+1)\frac{\prod_{i=0}^{|C|-1}(q(\epsilon)-i)}{|C|!}$.
To find a solution where exactly x_0 constraints are satisfied:

$$p(\epsilon) = (q(\epsilon)-x_0+1)\frac{\prod_{i=0}^{x_0-1}(q(\epsilon)-i)\prod_{i=x_0+1}^{|C|}(i-q(\epsilon))}{x_0!(|C|-x_0)!}. \quad (14)$$

Notably, to find a solution where the maximum number of constraints are satisfied, one can insert in X a variable x_0 with domain $|C|..1$ and use its current value in Eq. 14.

Theorem 2 *The described technique offers t-privacy (No collusion of less than t attackers can learn anything else than the final solution, and whatever can be inferred from it).*

Proof The technique is based on the evaluation of a set of functions consisting solely of additions and multiplication. It has been proven in [Yao82, GMW86, GMW87, CCD88a, CCD88b, BOGW88] that the compilation to multi-party computations of such a technique is t-private.

5 Secure search of a randomly chosen solution

With CSPs one is often not interested in getting the first solution given some order on variables and values, but rather in getting any solution. Note that finding the first solution ϵ_0 reveals two distinct things:

- ϵ_0 is a solution (or at least that the elements communicated to each participant are part of a solution ϵ_0).
- there exists no solution lexicographically ordered before ϵ_0.

But this is more information than what we want to reveal! We only want to find a solution while the information that in a certain search space there exists no solution is redundant and can be used by adversaries to infer details on secret constraints. The remaining question is *how could one modify the previous technique to return a randomly chosen solution rather than the first solution.*

Let us approach this problem by agreeing on an acceptable definition of a randomly chosen solution.

Definition 2 *We will say that a solution ϵ is randomly chosen if no agent can infer based on it the density of solutions in some other search sub-space.*

Surely, if it is found that the whole problem is unsatisfiable, then this information cannot be hidden. The basic observations exploited in this subsection are that:

- It is sufficient to choose random orders on variables and values in order to get a random solution with a deterministic protocol returning the first solution.
- If we succeed to hide the amount of effort and the orderings used by the deterministic first-solution-returning solver, then participants cannot infer anything on the characteristic of other search subspaces.

5.0.1 Adapting Merritt's reordering protocol

In Section 3.2 we have seen how a random hidden reordering can be obtained using a version of the Merritt's election protocol. Merritt's protocol can use a set of trusted parties as election centers or, as in our algorithm, the function of the election centers is taken by the n participants in the DisCSP ordered in a predefined chain, e.g. $A_1, A_2, ..., A_n$.

Assume that each agent A_i has to share a set of secrets $\{s^i_k\}$ where indexes k, are taken from disjoint

sets for distinct agents $(\forall s^i_{k_1}, s^j_{k_2}, (i{\neq}j){\Rightarrow}(k_1{\neq}k_2))$. The value of k, $0{<}k{\leq}K$, is not transmitted unchanged in the Merritt chain, making sure that the source and identity of the secret becomes hidden to everybody.

A pair (ε_k, v_k), where $v_k \in \{0,1\}$ is the evaluation of a constraint of A_i for the tuple ε_k, is called an *atomic predicate*. Each agent A_i submits to A_1 all its atomic predicates, namely all information that each (partial) valuation is or is not feasible. For each (partial) valuation $\varepsilon_k = ((x_{k_1}, v^{k_1}_{k^1}), ..., (x_{k_t}, v^{k_t}_{k^t}))$, having A_j's share of the secret feasibility value $v_k \in \{0,1\}$ equal to s^j_k, the submission takes the form:

$$\langle\langle(k_l, k^l)\rangle_{l\in[1..t]}, k, E_j(s^j_k), j\rangle$$

All the submissions are made to A_1, the first in the chain of permutation agents. Assuming the total number of submitted atomic predicates is K, each agent A_i generates $Z{=}K{+}mn$ sets of Shamir secret shares of 0, $\{z^k_j | j \in [1..n], z^k_j{=}\sum_{u=1}^{t-1} a_{k,u}(k_j)^u\}\}$, for some $a_{k,u}$, $k{\in}[1..Z]$, and secret permutations:

$$\begin{aligned} \pi: &\quad [1..m] \to [1..m], && \text{(for variables)}\\ \pi_1, ..., \pi_m: &\quad [1..d] \to [1..d], && \text{(for domains)}\\ \pi_0: &\quad [1..K] \to [1..K]. && \text{(for atomic predicates)}\end{aligned}$$

When A_1 or a subsequent A_i receives (all the elements of) a vector $\{\langle\{(k_l, k^l)\}_{l\in[1..t]}, k, E_j(s^j_k), j\rangle\}_{k\in[1..K]}$, it generates a permutation w^k and the vector $\pi_0(\{\langle\{(\pi(k_{w^k(l)}), \pi_{k_{w^k(l)}}(k^{w^k(l)}))\}_{l\in[1..t]}, \pi_0(k),$ $E_j(s^j_k)E_j(z^k_j), j\rangle\}_{k\in[1..K]})$, which is sent to A_{i+1}. The position of pairs inside each valuation are randomly shuffled according to w^k. A_n distributes the vectors to corresponding A_j.

Example 3 *Take a DisCSP with* $n{=}3$, $d{=}2$, *and* $m{=}2$ *(for* $n{\leq}2$, *the achieved privacy is irrelevant).* A_1 *(secretly) does not want* $((x_1{=}v^1_1), (x_2{=}v^2_1))$, A_2 *(secretly) does not want* $((x_1{=}v^1_2), (x_2{=}v^2_2))$, *and* A_3 *(secretly) does not want* $((x_1{=}v^1_1), (x_2{=}v^2_1))$. *Consider for simplicity that 0 is always shared as* $3x{+}0$ *and 1 as* $4x{+}1$, mod 7. $((x_i{=}v^i_a), (x_j{=}v^j_b))$ *is denoted in the following:* $\binom{i\ j}{a\ b}$.

A_2 *sends to* A_1: $\langle(\binom{12}{11}, 1, E_1(3), 1), (\binom{12}{12}, 2, E_1(5), 1), (\binom{12}{21}, 3, E_1(5), 1), (\binom{12}{22}, 4, E_1(5), 1)\rangle$, *and* A_2 *also sends to* A_1 *with target* A_2, *and* A_3
$\langle(\binom{12}{11}, 1, E_2(6), 2), (\binom{12}{12}, 2, E_2(2), 2), (\binom{12}{21}, 3, E_2(2), 2), (\binom{12}{22}, 4, E_2(2), 2)\rangle,$
$\langle(\binom{12}{11}, 1, E_3(2), 3), (\binom{12}{12}, 2, E_3(6), 3), (\binom{12}{21}, 3, E_3(6), 3), (\binom{12}{22}, 4, E_3(6), 3)\rangle.$
A_1 *submits to itself:* $\langle(\binom{12}{11}, 5, E_1(5), 1), (\binom{12}{12}, 6, E_1(5), 1), (\binom{12}{21}, 7, E_1(5), 1), (\binom{12}{22}, 8, E_1(3), 1)\rangle$, *and shares to be transmitted to* A_2, *and* A_3
$\langle(\binom{12}{11}, 5, E_2(2), 2), (\binom{12}{12}, 6, E_2(2), 2), (\binom{12}{21}, 7, E_2(2), 2), (\binom{12}{22}, 8, E_2(6), 2)\rangle,$
$\langle(\binom{12}{11}, 5, E_3(6), 3), (\binom{12}{12}, 6, E_3(6), 3), (\binom{12}{21}, 7, E_3(6), 3), (\binom{12}{22}, 8, E_3(2), 3)\rangle...$
A_1 *applies permutations* $\pi{=}(2,1)$, $\pi_1{=}(1,2)$, $\pi_2{=}(2,1)$, $\pi_0{=}(3,5,2,7,1,8,4,6,10,12,9,11)$, *and adds new random shares of zero: for simplicity always* $2x$ mod 7. A_2 *receives from* A_1*:*
$\langle(\binom{21}{12}, 1, E_1(0), 1), (\binom{12}{22}, 2, E_1(0), 1), (\binom{21}{12}, 3, E_1(5), 1), (\binom{21}{22}, 4, E_1(0), 1), (\binom{21}{11}, 5, E_1(0), 1), (\binom{21}{21}, 6, E_1(5), 1), (\binom{21}{21}, 7, E_1(0), 1), (\binom{21}{11}, 8, E_1(0), 1), ...\rangle,$
$\langle(\binom{21}{12}, 1, E_2(6), 2), (\binom{12}{22}, 2, E_2(6), 2), (\binom{21}{12}, 3, E_2(3), 2), (\binom{21}{22}, 4, E_2(6), 2), (\binom{21}{11}, 5, E_2(6), 2), (\binom{21}{21}, 6, E_2(3), 2), (\binom{21}{21}, 7, E_2(6), 2), (\binom{21}{11}, 8, E_2(6), 2), ...\rangle...$
A_2 *and* A_3 *perform permutations in a similar way as* A_1. A_3 *distributes the atomic predicates for launching the search with SecureSatisfaction.*

Decoding solutions After SecureSatisfaction is run, the shares of the results of functions f have to be revealed without revealing the permutation. The vectors $\{\langle\{E_j(f'^j_i[t])\}_{t\in[1..d]}, j\rangle\}_{i\in[1..m]}$, for each j, are sent backward through the chain of agents, where $f'^j_i[t]$ is A_j's share for $f'_i[t]$ (see step 6 in Algorithm 3). When A_k receives $\{\langle\{E_j(f'^j_i[t])\}_{t\in[1..d]}, j\rangle\}_{i\in[1..m]}$, it generates and sends to A_{k-1} the vector $\pi^{-1}(\{\langle\pi^{-1}_{\pi^{-1}(i)}(\{E_j(f'^j_i[t])E_j(z^{K+(i-1)(t-1)+1}_j)\}_{t\in[1..d]}), j\rangle\}_{i\in[1..m]})$. A_1 broadcasts them.

Hiding shares To avoid that everybody learns all the secret shares, these are sent encrypted with the public key of their destination participant (see Algo-

procedure SecureRandomSolution do

1. Each agent A_i generates for each agent A_j and for each of the tuples t_k in its predicates a secret share s_k^j. Agents have disjoint sets of indexes k.
2. $(t_k, k, E_j(s_k^j), j)$ is submitted through the chain of agents according to our version of Merritt's reformulation protocol.

   For each k, A_i generates a new random set of Shamir shares of 0, $\langle z_k^1, z_k^2, ..., z_k^n \rangle$. Each of these shares are encrypted with the public keys of the corresponding agents obtaining $\langle E_1(z_k^1), E_2(z_k^2), ..., E_n(z_k^n) \rangle$. The operation $\cdot$ is applied on this vector and on the encrypted shares received for the secret k.
3. After the chain of encryptions and permutations, each obtained atomic predicate $(t_{k'}^{(n)}, k', E_j(s_k'^j), j)$ is sent by A_n to A_j. A_j decrypts $s_k'^j$ and learns it as its share for the secret k' of a predicate on the tuple $t_{k'}^{(n)}$.
4. The SecureSatisfaction algorithm (Algorithm 3 without the steps 1 and 7a) is now applied to get the first solution of the reformulated CSP.
5. The secret shares of the solution to the CSP are submitted through the chain of participants in reverse direction undoing the permutations. To allow unshuffling secret indexes of values, they are transmitted via the unary constraint representation of each f_j, namely f_j' computed in Algorithm 3.
6. Each participant in the Merritt chain, applies the same procedure as described in step 2 with the sole difference that the permutations are reversed (but still random shares of 0 are added to encrypted shares of the solution). After unshuffling, the encrypted shares are opened by their destination agents, and the owners of each resource are communicated the shares defining the allocation in the obtained solution.

Algorithm 5: Algorithm to find a random solution of a distributed CSP

rithm 5 steps 1 and 2). What can happen if one would simply shuffle the shares before using Algorithm 3 or the decoding is that the agents that receive back their own solution shares for the problem can match these against the values that they sent for themselves and retrieve part of the overall permutation of the shuffling. This has been avoided as seen before. Let us explain this in more detail.

Each agent A_i in the Merritt chain generate random sets of n shares for 0 with the technique of Shamir. The j^{th} share in the k^{th} set is denoted by z_j^k. Whenever A_i performs a shuffling/unshuffling of a set of encrypted secret shares $\langle E_1(s_k^1), ..., E_n(s_k^n) \rangle$, A_i uses a new set of shares for 0 and multiplies the corresponding encrypted shares with the point product.

$$\langle E_1(s_k^1), E_2(s_k^2), ..., E_n(s_k^n) \rangle \cdot \langle E_1(z_k^1), E_2(z_k^2), ..., E_n(z_k^n) \rangle =$$
$$\langle E_1(s_k^1)E_1(z_k^k), E_2(s_k^2)E_2(z_2^k), ..., E_n(s_k^n)E_n(z_n^k) \rangle$$
$$= \langle E_1(s_k^1 + z_1^k), E_2(s_k^2 + z_2^k), ..., E_n(s_k^n + z_n^k) \rangle.$$

Therefore, since we use Paillier encryption, the obtained shares represent the sum of the secret s_k with 0 and is a re-sharing of the secret defined by $\langle s_k^1, ..., s_k^n \rangle$.

Theorem 3 *The SecureRandomSolution algorithm (Algorithm 5) is correct and no coalition of less than n/2 passive attackers can find anything except what can be inferred solely from the solution and from their previous knowledge.*

Proof In this paper we proposed a CSP solver that has all the properties needed for compilation into a secure protocol with the claimed characteristics [Yao82, GMW86, CCD88b]. The only problem after the previous section remained how to secretly reformulate the problem (permuting variables and values) to hide the space searched before finding the first solution. We have just shown how a slight adaptation of Merritt's protocol offers exactly what we needed. A related technique appears in [BT94].

Decoding solutions is similar with the Encoding (see Example 3), just that the permutations are reversed and all the unshuffled atomic predicates are unary.

Synchronization The agents perform all these multi-party computations in rounds. A computation round consists of an eventually empty contiguous sequence of additions and multiplications of secrets with public values, ending with the (re-)sharing of a secret. As noted in Algorithm 3 at step 4, the technique can be optimized by performing several unrelated operations in the same round in parallel

(e.g. computations of $g_{j,k}$s). Round starts are synchronized by any synchronization technique at implementation's choice. For example, agents can keep a counter of the current round number, and the round number tags each generated (re-)sharing message. For flow control, the (re)-sharing of round r_i is not sent before all messages of round $r_{i-b}, b>0$, were received from all other agents. This limits the buffer requirements to the messages of $2b$ rounds.

Proof If all round r_{i-b} shares were sent, it means that everybody got and processed all round r_{i-2b} shares.

6 Conclusions and Alternatives

DisCSPs are a very active research area. Privacy has been recently stressed in [SSHF00, MJ00, WF02, SF02, YSH02] as an important goal in algorithm design. For problems with very small domains, it is possible to use instead of the Paillier cryptosystem a version of ElGamal where discrete logarithms have to be computed [Gen95], which is intractable. A similar algorithm based only on $(\times, \times)$-homomorphic public key cryptosystems is possible but much more expensive and complex. We know to develop some cheaper cryptographic techniques but they would offer weaker security, e.g. variable running time that together with the solution can reveal additional secrets.

A similar algorithm based only on $(\times, \times)$-homomorphic public key cryptosystems can also be developed. That is nevertheless with orders of magnitude more expensive, as additions of shares with shared 'Zero' have to be replaced by multiplications with shared 'One'. This can be achieved by starting locally with the computation of the point product as we did here, achieving a $(2t, n)$-threshold scheme distribution of the product. But then one has at each such multiplication to distribute each product of shares to the owners of the corresponding secret keys. This has to be done in a random order of the secrets such that the participants cannot track their shares. Those agents have to complete the resharing of the result from the $(2t, n)$-scheme back to the (t, n)-scheme, as it is typically done for multiplication. Then the results are sent back to the corresponding agent in the Merritt chain, encrypted with the sender's public key.

The described version is not robust when participants do not behave according to the protocol. While we work on techniques that may provide such robustness, as for other multi-party computations, they were not addressed in this discussion (notably see (t,n) threshold schemes with $t<n/3$) [BOGW88].

We presented a technique where agents that need to cooperate and whose problems can be modeled as CSPs can find a random solution without any leak of additional information about their constraints. It is the first such technique requiring no need of trusted servers (To be noted that this cherished property was the historically claimed driving force behind the development of public key cryptography [DH76]). It is assumed that all adversaries are passive (they follow the protocol) and that only a minority of the participants may be corrupted by any attacker. It is also assumed that the set of variables involved in the CSP of each agent are public knowledge, requirement that can be by-passed if each agent declares a single constraint involving all variables. Most of these assumptions can be relaxed with different trade-offs and this is an important research field.

References

[ASK02] M. Agarwal, N. Saxena, and N. Kayal. Primes is in P. In *www.cse.iitk.ac.in/primality.pdf*, August 2002.

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

[BT94] Josh Benaloh and D. Tuinstra. Receipt-free secret ballot elections. In *26th ACM Symposium on Theory of Computing*, pages 544–533, 1994.

[CCD88a] D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols. In *Proc. CRYPTO 87, LNCS 293*, page 462, 1988.

[CCD88b] D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols. In *Proc. 20th ACM (STOC)*, pages 11–19, Chicago, 1988.

[DH76] W. Diffie and M. Hellman. Multiuser cryptographic techniques. *IEEE Transactions on Information Theory*, 1976.

[ElG84] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Crypto'84*, volume 196 of *LNCS*, 1984.

[GB96] S. Goldwasser and M. Bellare. Lecture notes on cryptography. MIT, 1996.

[Gen95] R. Gennaro. 6.915 computer and network security, lecture 24, Dec 1995.

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

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game — a completeness theorem for protocols with honest majority. In *STOC*, pages 218–229, 1987.

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

[Knu98] D. Knuth. *The art of computer programming*, volume 2. A-W, 1998.

[Mer83] M. Merritt. *Cryptographic Protocols*. PhD thesis, Georgia Inst. of Tech., Feb 1983.

[MH78] R. Merkle and M. Hellman. Hiding information and signatures in trapdoor knapsacks. *IEEE Transactions on Information Theory*, 24:525–530, 1978.

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

[Pai99] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Eurocrypt'99*, volume 1592 of *LNCS*, pages 223–238, 1999.

[SF02] M. C. Silaghi and B. Faltings. A comparison of discsp algorithms with respect to privacy. In *3rd DCR-02 Workshop*, Bologna, July 2002.

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

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

[Sta98] M. Stadtler. Publicly verifiable secret sharing. In *Proc.*, 1998.

[Tsa93] E. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1993.

[WF02] R.J. Wallace and E.C. Freuder. Constraint-based multi-agent meeting scheduling: Effects of agent heterogeneity on performance and privacy loss. In *DCR*, pages 176–182, 2002.

[Yao82] A. Yao. Protocols for secure computations. In *FOCS*, pages 160–164, 1982.

[YSH02] M. Yokoo, K. Suzuki, and K. Hirayama. Secure distributed constraint satisfaction: Reaching agreement without revealing private information. In *CP*, 2002.