Secure Stochastic Multi-party Computation for Combinatorial Problems

Marius C. Silaghi[†] and Gerhard Friedrich[‡]
[†]Florida Institute of Technology, USA
[‡]University Klagenfurt, Austria

May 21, 2005

Abstract

High levels of security often imply that the computation time should be independent of the value of involved secrets. When the expected answer of the solver is either a solution or *unsatisfiable*, then the previous assumption leads to algorithms that take always the computation time of the worst case. This is particularly disturbing for NP-hard combinatorial problems.

In this work we start from the observation that sometimes (specially for hard problems) users find it acceptable to receive as answer either a solution, the answer *unsatisfiable* or a failure with meaning *don't know*. More exactly users accept *incomplete* solvers. As argued in [Sil05b], for certain problems privacy reasons lead users to prefer having an answer meaning *don't know* even when the secure multi-party computation could have proven *unsatisfiable* (to avoid revealing that all alternatives are infeasible). While the solution proposed in [Sil05b] is slower than complete algorithms, here we show secure stochastic solutions that are faster than complete solvers, allowing to address harder problem instances.

1 Introduction

Typical examples of combinatorial problems are meeting scheduling, resource allocation, time-tabling, auctions with several possible winners. Such a problem is typically defined by a set of variables and constraints on the satisfiable assignments to these variables. The set of all (satisfiable and unsatisfiable) simultaneous assignments of values to all variables defines the *search space* of the problem. An element of the search space is also referred to as an alternative to be considered as a solution to the problem, or simply *alternative*.

A complete solver is one that reports a solution whenever a solution exists. The answer of such a technique is either a solution or *unsatisfiable*. Combinatorial problems can be very hard and therefore we no not have efficient complete secure multi-party computation solvers. Several complete secure solvers were proposed in the past for such problems, and high levels of security always require a computation time that is given by the worst possible case (over all possible values of the secrets).

It was shown that for problems that are solved only once, minimization of privacy loss often requires that the solution be picked randomly, preferably with a uniform distribution among the existing solutions [SR04]. Such a random selection can be achieved if the problem is shuffled prior to solving [Sil03, Sil04]. Two families of techniques were proposed for shuffling a shared description of a combinatorial problem,one based on mix-nets and one based on arithmetic circuits [Sil05a].

Sometimes, the security requirements themselves require an incomplete solver (when the proof of unsatisfiability of the problem leads to unacceptable privacy loss, by revealing that all alternatives are infeasible) [Sil05b]. The answer of such a solver is either a solution or *unsatisfiable*. However, the solution proposed in [Sil05b] is actually slower than complete solutions. It first computes a solution with a complete secure solver and then it hides the solution with some small probability.

In this work we show how the shuffling performed on problem descriptions prior to solving allows to build an incomplete secure stochastic multi-party solver where a high level of privacy is offered. The answers of the solver consists in either a solution or in *don't know*, and nothing is revealed about the set of alternatives that were not explored (except for its size). Notably, these algorithms are strictly faster than the corresponding complete versions and are parametrized with the percentage of the search space to be explored (the search space is the set of all alternatives that may or may not satisfy the combinatorial problem).

By specifying the percentage of the combinatorial problem to be explored, one practically specifies the exact amount of computation (time) that the solver should perform. The proposed techniques are different for shuffling with mix-nets and for shuffling with arithmetic circuits.

2 Background

Combinatorial problems have been often discussed in Computer Science and many examples are known to be very hard. For example SAT was the first proven NP-complete problem and Constraint Satisfaction Problems are largely addressed with stochastic and incomplete solvers.

A Constraint Satisfaction Problem (X,D,C) is defined by a set of variables $X = \{x_1, ..., x_m\}$, a set of domains $D = \{D_1, ..., D_m\}$ where D_i is the domain for x_i, and a set of constraints $C = \{\phi_1, ..., \phi_c\}$. Each constraint ϕ_j specifies the acceptable combinations of assignments of values to a subset X_j of the variables. A tuple is a vector of assignments of values to distinct variables. A solution of the CSP is a tuple of assignments of values to all the variables and that satisfies all the constraints. The search space of the CSP is defined by the Cartesian product $D_1 \times ... \times D_m$. An element of the search space is called an *alternative*. The i^{th} alternative is denoted by ϵ_i.

A distributed CSP is a CSP (X, D, C) where a set of participants $A = \{A_1, ..., A_n\}$ have secret shares of C, none of them knowing the whole set C.

2.1 Shuffling an array of shared secrets

Secure multi-party computations can simulate any arithmetic circuit [BOGW88] or boolean circuit [Kil88, Gol04] 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. Each leaf is a constant.

The secure multi-party simulation of arithmetic circuit evaluation proposed in [BOGW88] exploits Shamir's secret sharing [Sha79]. This sharing is based on the fact that a polynomial $f(x)$ of degree $t-1$ with unknown parameters can be reconstructed given the evaluation of f in at least t distinct values of x, using Lagrange interpolation. 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 τ_i is assigned to each participant A_i. The value of the pair $(\tau_i, f(\tau_i))$ is sent over a secure channel (e.g. encrypted) to each participant A_i. This is called a (t, n)-threshold scheme. We will assume that all computations are performed in a field $\mathbb{Z}_q$ for some prime number q. Once secret numbers are shared with a (t, n)-threshold scheme, evaluation of an arbitrary arithmetic circuit can be performed over the shared secrets, in such a way that all results remain shared secrets with the same security properties (the number of supported colluders, $t-1$) [BOGW88, Yao82]. For [Sha79]'s technique, one knows to perform additions and multiplications when $t \leq (n-1)/2$. Since any $\lfloor n/2 \rfloor$ participants cannot find anything secret by colluding, such a technique is called $\lfloor n/2 \rfloor$-private [BOGW88]. It is also known how to evaluate with computational securely any arithmetic circuit on additively shared secrets.

Shuffling with mix-nets In [Sil03, Sil04, Sil05a] it is shown how a mix-net can shuffle a vector of shared secrets and can unshuffle a vector of the same size using the inverse permutations. Each participant encrypts his share of each secret using a $(+ \mod q, X)$ public encryption scheme for which it holds the secret key, and sends a vector holding each encrypted share to A_1. The vectors with the encrypted shares are passed along each participant in A, each of the applying the same secret permutation on all vectors. A shared 0 is also added to each sharing of a secret using the homomorphism of the encryption. Each participant will provide the others with a zero-knowledge proof for the correctness of his shuffling (respectively unshuffling).

Shuffling with arithmetic circuits Assume that we have composable multi-party computations [Kil05] for computing:

- $\delta_K(x, y)$: Kronecker's delta returning a shared 1 when $x = y$ and 0 otherwise
- $cmp(x, y)$ returns 1 when $x < y$ and 0 otherwise
- $RS(m, M)$: random secret generator, generating a shared secret in the interval m, M.

It is possible to design an arithmetic circuit for shuffling secrets, using the Algorithm 3. This algorithm uses Algorithm 1 for a permutation of two elements on secret

positions in a vector. The random permutation is defined by a random vector computed with Algorithm 2. Unshuffling can be done with the Algorithm 4.

function *Perm (s,i,r,m,M,k*
  $s_i = \sum_{j=m}^{M}(\delta_K(r,j) * s_j)$;
  for $j \in (i,k]$ **do**
    $s_j = s_j + (s_i - s_j) * \delta_K(r,j)$;

Algorithm 1: Permuting element s_i with s_r for a secret value $r \in [m,M]$ in vector s with k shared secrets

function *RandomVector(k)*
  for $j = 1$ *to* $k-1$ **do**
    $r[j] = RS(j,k)$;
  return r;

Algorithm 2: Shuffling a vector s with k shared secrets

function *Shuffle(s,k,r)*
  for $j = 1$ *to* $k-1$ **do**
    Perm(s,j,r[j],j,k,k);

Algorithm 3: Shuffling a vector s with k shared secrets, and a random vector r obtained with Algorithm 2

This permutation was shown in [Sil05a] to lead to a random shuffling (taken from a uniform distribution). Note that the random vector defining the permutation could have been built allowing each element to belong to any value between 1 and k. This would be computationally more expensive as it would require each call to the procedure $Perm$ to recompute all the elements of the vector to be shuffled (see Algorithm 5).

2.2 MPC-DisCSP4

In [Sil05b] we have proposed a multi-party computation technique, called MPC-DisCSP4, that extracts a random solution of a distributed CSP. MPC-DisCSP4 uses general multi-party computation building blocks. General multi-party computation techniques can solve securely certain functions, one of the most general classes of solved problems being the arithmetic circuits. A distributed CSP is not a function. A DisCSP can have several solutions for an input problem, or can even have no solution. Two of the three reformulations of DisCSPs as a function (see [SR04]) are relevant for MPC-DisCSP4:

i A function DisCSP^1() returning the first solution in lexicographic order, respectively an invalid valuation τ when there is no solution.

```
function Shuffle(s,k,r)
    for j = k − 1 to 1 do
        Perm(s,j,r[j],j,k,k);
```

Algorithm 4: Un-shuffling a vector s with k shared secrets, when the shuffling was defined by random secret vector r.

```
function Shuffle(s,k,r)
    for j = 1 to k do
        Perm(s,j,r[j],1,k,k);
```

Algorithm 5: Shuffling a vector s with k shared secrets, and a random vector r where each element is obtained with $RS(1, k)$.

ii A probabilistic function DisCSP() which picks randomly a solution if it exists, respectively returns τ when there is no solution.

For privacy purposes only the 2^{nd} alternative is satisfactory. DisCSP() only reveals what we usually expect to get from a DisCSP, namely *some* solution. DisCSP^1() intrinsically reveals more [SR04]. MPC-DisCSP4 implements DisCSP() in five phases:

1. Share the secret parameters of the input DisCSP using Shamir's secret sharing. The value of each publicly possible assignment (allocation) is securely evaluated.
2. The shared DisCSP problem is shuffled in a cooperative way, reordering values (and eventually variables), with a permutation that is not known to anybody [Sil05a].
3. A version of DisCSP^1() where the operations performed by agents are independent of the input secrets (to avoid leaking the secrets), is executed by simulating arithmetic circuits evaluation with the technique in [BOGW88].
4. The solution returned by DisCSP^1() at Step 3 is translated into the initial problem formulation using a transformation that is inverse of the shuffling at Step 2 [Sil05a].
5. Construct the solution from its secret shares.

It is also possible and very simple to find all solutions [HCN$^+$01]. However, when only a single solution is needed, this leaks a lot of information. At Step 3, MPC-DisCSP4 requires a version of the DisCSP^1() function whose cost is independent of the input, since otherwise the users can learn things like: *The returned solution is the only one, being found after unsuccessfully checking all other tuples, all other tuples being infeasible.* Since the used DisCSP^1() has to be independent of the problem details, its cost is exponential (at least as long as nobody proves P=NP).

Note that other alternative techniques are available, notably MPC-DisCSP1 [Sil03], MPC-DisCSP2 [SM04], and MPC-DisCSP3 [Sil04]. We call them generically MPC-

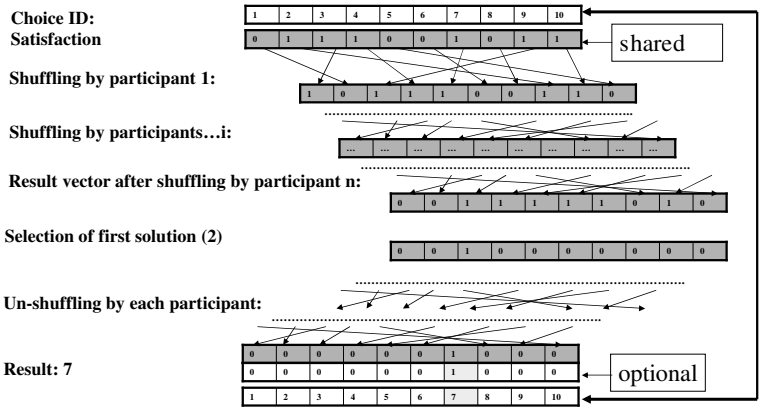

Figure 1: MPC-DisCSP4 using mix-nets

DisCSPx. In this paper we only address multi-party computations without trusted servers. A family of secure solvers based on trusted servers is proposed in [YSH02].

2.3 Hiding existence of solution

When no solution is found, all the participants learn that each alternative is infeasible. For certain problems this leak of secrets may be considered unacceptable and a *don't know* answer is prefered to learning the infeasibility. But the *don't know* answer is believable only if the algorithm may indeed miss some solutions. An algorithm for missing the solution with some predefined probability p is described in [Sil05b]. It consists of computing a solution using a MPC-DisCSPx algorithm and then setting the assignments in the result to the invalid value 0 with a probability p.

2.4 Stochastic algorithms

In the CSP world it is known that complete algorithms are ineffective for hard problem instances. For large problems, most applications apply stochastic search procedures. With stochastic search, only a subset of the search space is analyzed. Typical examples of stochastic search are based on some type of hill climbing. With hill-climbing the solver starts with a random alternative and searches the neighbouring search space for solutions.

3 Secure Stochastic Search

Let us finally detail our proposed techniques for tractable secure stochastic search, allowing to address hard problems. The idea is that only a subset of t alternatives from, the search space will be explored. This could be achieved by adding a public constraint that removes the remaining search space. However, to ensure privacy in case of failure

(that the infeasibility of this particular sub-space is not revealed), we propose to take advantage of the shuffling of the whole problem. We select the subspace to be explored from the shuffled problem. This hides the exact search subspace that is analyzed and the only secret leaked in case of failure is that there are t infeasible alternatives (but they are not known).

3.1 Secure Stochastic Search with Mix-nets

Each MPC-DisCSPx solving algorithm using mixnets can be modified into a corresponding secure stochastic search protocol that will be called Stochastic Multi-Party Computation for Distributed CSPs (SMPC-DisCSPx). Each SMPC-DisCSPx differs from the corresponding MPC-DisCSPx by the fact that only the first t tuples of the shuffled search space are used to compute the shuffled solution. Each stochastic solver is parametrized by the number t of alternatives to be explored (t beeing smaller or equal to the size of the search space). To be noted that a stochastic solver can be seen as a generalization of the corresponding complete solver, which is obtained when t equals the size of the search space.

SMPC-DisCSP4 For example, SMPC-DisCSP4 is shown in Algorithm 6.

```
function SMPC-DisCSP4(t,(X,D,C))
    for i=1 to k do
        S[i]=∏_{φ∈C} φ(ε_i);
    SHUFFLE(S) //using the mixnet;
    h[1]=1;
    for i=2 to t do
        h[i]=h[i-1]*(1-S[i-1]);
        S[i]=S[i]*h[i];
6.1 /* S[t]=S[t]*cmp(RS(0,q-1),p*q)// fine tuning*/;
    UNSHUFFLE(S);
    return S// the solution can be extracted from S as in [Sil05b];
```

Algorithm 6: SMPC-DisCSP4 for solving a CSP (X, D, C) with k alternatives allowed by the public constraints, and exploring t alternatives.

SMPC-DisCSP4 requires $k(c-1)$ multiplications of secrets to build the vector S and $2t$ multiplications of secrets to select the solution. Also, the shuffling and unshuffling require each $O(kn^2)$ expensive operations, $O(kn)$ for each participant. While SMPC-DisCSP4 leads to a reduction with up to $2k$ multiplications of secrets, the complexity remains the same, dictated by the shuffling.

It can be noted that the probability that a solution is lost can be fine tuned (e.g. for the application in [Sil05b]) by discarding the alternative ϵ_t with probability p. This can be done by uncommenting the Line 6.1 in the Algorithm 6.

SMPC-DisCSP1 The stochastic algorithm obtained from MPC-DisCSP1 is more successful, and is sketched in Algorithm 7.

```
function SMPC-DisCSP1(t,(X,D,C))
    SHUFFLE(X,D,C) //using the mixnet;
    for i=1 to t do
        S[i]=∏_{φ∈C} φ(ε_i);
    /* S[t]=S[t]*cmp(RS(0,q-1),p*q)// fine tuning*/;
    F=DisCSP1(t,(X,D,C));
    UNSHUFFLE(F);
    return F;
```

Algorithm 7: SMPC-DisCSP1 for solving a CSP (X, D, C) with k alternatives and exploring t alternatives.

DisCSP1 is the arithmetic circuit proposed in [Sil03], with the only modification that function `gconsistent()` only integrates the first t tuples (rather than the whole search space). The result F returned by DisCSP1 is a set of vectors, one for each variable. A vector contains shared 0s on all positions, except for a 1 on the position corresponding the the value of the corresponding variable in the found solution. If there is no solution, then all elements of the vectors are 0.

The cost of SMPC-DisCSP1 is only $O(t(md + c))$ multiplications of secrets. Of these, $t(c - 1)$ are used to compute S. DisCSP1 computes `gconsistent` md times, each of them requiring at most $O(t)$ multiplications. The cost of shuffling in SMPC-DisCSP1 can be small even for large and hard problems, if the maximim constraint arity (number of involved variables) is small.

3.2 Secure Stochastic Search with arithmetic circuits

The secure stochastic algorithms based on mixnets suffer from the fact that the cost of shuffling remains the same as for the non-stochastic complete approaches. This was particularly negative in the case of SMPC-DisCSP1 where the cost of the shuffling is the main cost.

This problem is reduced in algorithms with shuffling based on arithmetic circuits. Namely, with shuffling based on arithmetic circuits one does not need to compute the whole shuffling. With SMPC-DisCSP4, it is possible to only compute the first t elements of the shuffled problem (see Algorithms 8, 9), and 10).

```
function Shuffle(s,k,r,t)
    for j = 1 to t do
        Perm(s,j,r[j],j,k,k);
```

Algorithm 8: Shuffling a vector s with k shared secrets, and a random vector r obtained with Algorithm 2

```
function Shuffle(s,k,r,t)
    for j = t to 1 do
        Perm(s,j,r[j],j,k,k);
```

Algorithm 9: Un-shuffling a vector s with k shared secrets, when the shuffling was defined by random secret vector r.

```
function SMPC-DisCSP4ac(t,(X,D,C))
    for i=1 to k do
        S[i]=∏_{φ∈C} φ(ε_i);
    R=RandomVector(t);
    SHUFFLE(S,k,R,t) //using the mixnet;
    h[1]=1;
    for i=2 to t do
        h[i]=h[i-1]*(1-S[i-1]);
        S[i]=S[i]*h[i];
    /* S[t]=S[t]*cmp(RS(0,q-1),p*q)// fine tuning*/;
    for i=t+1 to k do
        S[i]=0;
    UNSHUFFLE(S,k,R,t);
    return S// the solution can be extracted from S as in [Sil05b];
```

Algorithm 10: SMPC-DisCSP4ac, solving a CSP (X, D, C) with k alternatives allowed by the public constraints, and exploring t alternatives.

It can be noted that in secure stochastic algorithms based on arithmetic circuits we succeed to reduce the cost of shuffling and unshuffling from $O(k^2)$ to $O(kt)$ multiplications of secrets. With this improvement the complexity of SMPC-DisCSP4ac decreases, but remains high since k is large for hard problems (can be exponential in the problem size).

In conclusion the most appropriate algorithm for Stochastic Search is SMPC-DisCSP1 which has polynomial space requirements and whose computational (time) complexity can be bounded to low values being linear in t and in the problem size.

SMPC-DisCSP4ac (with arithmetic circuits) has a time complexity significantly smaller than MPC-DisCSP4 ($O(k(t+c))$ versus $O(k^2)$). This implies that the size of the problems solvable with SMPC-DisCSP4' is larger than the size solvable with MPC-DisCSP4, which had the best complexity among complete algorithms.

Remark 1 (SMPC-DisCSP1ac) *Arithmetic circuit shuffling for SMPC-DisCSP1 works by separately permuting each domain (with a separate random vector for each of them). The improvement that can be brought is to only compute the permuted constraint elements that are part of the first t tuples.*

The shuffling for SMPC-DisCSP1 is not expensive. Therefore possible improvements in versions based on arithmetic circuit shuffling are less significant, not changing the time complexity.

4 Conclusions

In this work we have proposed a new family of secure solvers for distributed Constraint Satisfaction Problems (disCSPs). While most existing techniques were complete and inapplicable to large instances, the new techniques can be used to address large problems.

We have proposed stochastic versions for each of the complete secure multiparty algorithms MPC-DisCSP1 and MPC-DisCSP4, based on shuffling with mixnets or with arithmetic circuit. MPC-DisCSP1 is remarkable for its polynomial space requirements while MPC-DisCSP4 for its low time complexity and for the uniform distribution in selecting solutions.

The new versions only explore a subset of the search space of the problem, subset whose size is specified as a parameter. We have thus analyzed in detail three newly obtained versions: SMPC-DisCSP1, SMPC-DisCSP4, and SMPC-DisCSP4ac.

As its complete counterpart, SMPC-DisCSP1 requires only polynomial space. Unexpectedly, the versions obtained from MPC-DisCSP4 are much less appropriate for addressing large problems, but maintain the desirable property pf selecting solutions with a uniform distribution. Among SMPC-DisCSP4 and SMPC-DisCSP4ac, the latter (based on arithetic circuits) presents the largest speed-up in comparison to its complete version. The algorithm of choice for tackling large problems are therefore the ones based on MPC-DisCSP1 (SMPC-DisCSP1 and SMPC-DisCSP1ac), and their time complexity is linear in the problem size and in a parameter deciding the size of the explored search space.

References

[BOGW88] M. Ben-Or, S. Goldwasser, and A. Widgerson. Completeness theorems for non-cryptographic fault-tolerant distributed computating. In *STOC*, pages 1–10, 1988.

[Gol04] Oded Goldreich. *Foundations of Cryptography*, volume 2. Cambridge, 2004.

[HCN+01] 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.

[Kil88] J. Kilian. Founding cryptography on oblivious transfer. In *Proc. of ACM Symposium on Theory of Computing*, pages 20–31, 1988.

[Kil05] Eike Kiltz. Unconditionally secure constant round multi-party computation for equality, comparison, bits and exponentiation. Cryptology ePrint Archive, Report 2005/066, 2005. `http://eprint.iacr.org`.

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

[Sil03] M.-C. Silaghi. Solving a distributed CSP with cryptographic multi-party computations, without revealing constraints and without involving trusted servers. In *IJCAI-DCR*, 2003.

[Sil04] M.-C. Silaghi. Meeting scheduling system guaranteeing n/2-privacy and resistant to statistical analysis (applicable to any DisCSP). In *3rd IC on Web Intelligence*, pages 711–715, 2004.

[Sil05a] M.-C. Silaghi. Zero-knowledge proofs for mix-nets of secret shares and a version of elgamal with modular homomorphism. Cryptology ePrint Archive, Report 2005/079, 2005. `http://eprint.iacr.org/`.

[Sil05b] Marius-Călin Silaghi. Hiding absence of solution for a discsp. In *FLAIRS'05*, 2005.

[SM04] M.-C. Silaghi and D. Mitra. Distributed constraint satisfaction and optimization with privacy enforcement. In *3rd IC on Intelligent Agent Technology*, pages 531–535, 2004.

[SR04] M.-C. Silaghi and V. Rajeshirke. The effect of policies for selecting the solution of a DisCSP on privacy loss. In *AAMAS*, pages 1396–1397, 2004.

[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 *Proc. of the AAMAS-02 DCR Workshop*, Bologna, July 2002.