A Cryptographic Solution for Private Distributed Simple Meeting Scheduling

Javier Herranz [a,1], Stan Matwin [b,2], Pedro Meseguer [b,3] and Jordi Nin [c,4]
[a] *Dept. Applied Mathematics IV , UPC, Jordi Girona 1-3, 08034 Barcelona, Spain*
[b] *IIIA, CSIC, Campus UAB, 08193 Bellaterra, Spain*
[c] *LAAS, CNRS, 7, Avenue du Colonel Roche, 31077 Toulouse, France*

Abstract. Meeting Scheduling is a suitable application for distributed computation, motivated by its privacy requirements. Previous work on this problem have considered some cryptographic techniques as well as new search strategies. In this paper, we provide a cryptographic and conceptually clear approach to solve a simple case of Meeting Scheduling, even achieving complete privacy.

Keywords. meeting scheduling, privacy, homomorphic public key encryption

1. Introduction

The Meeting Scheduling problem (*MS*) consists of a set of agents, each having a personal calendar (where previous private appointments may appear), a set of meetings among them, and a set of locations. The aim of *MS* is to determine *when* and *where* these meetings could occur [9]. This problem is naturally distributed because (i) each agent only knows his own personal personal calendar and (ii) agents desire to keep private their personal calendars during *MS* resolution. In a centralized approach, all agents must give their private information to a central server, which solves the problem and returns a solution. The presence of a trusted authority is not possible in most of cases. This motivates its reformulation in terms of distributed computation.

MS can be formulated as Distributed Constraint Satisfaction (*DisCSP*) with privacy requirements. To enforce privacy in *DisCSP*, two main approaches have been used. One considers the use of some cryptographic techniques (SMC-based solutions [13], with serious problems to scale up). Alternatively, other authors try to enforce privacy by different search strategies [9,8]. In some cases [3,9,8], solving causes revealing some private information, which could not be acceptable for some of the *MS* participants.

[1] Javier Herranz enjoys a Ramon y Cajal grant from the Spanish *Ministerio de Ciencia e Innovación*
[2] On sabbatical from University of Ottawa, Canada. Also affiliated with the Institute of Computer Science, Polish Academy of Sciences. Partially supported by the Natural Sciences and Engineering Research Council of Canada and the Ontario Centres of Excellence.
[3] Corresponding Author: IIIA CSIC, Campus UAB, 08193 Bellaterra, Spain; email: pedro@iiia.csic.es. This work is partially supported by the project TIN2006-15387-C03-01.
[4] Partial support by the European Community through the 7th Framework Programme Marie Curie Intra-European fellowship, contract No 235226 is acknowledged. Partial support by the Spanish MEC (projects ARES – CONSOLIDER INGENIO 2010 CSD2007-00004 – and eAEGIS – TSI2007-65406-C03-02).

Here we consider a simple case of *MS*, when all agents are involved in one meeting in one or several possible locations. We provide two cryptographic solutions for this simple case: one offers complete privacy, while the other –computationally more efficient– offers high privacy. Although we do not consider the general case of multiple meetings in multiple locations, we believe that the proposed approach addresses a realistic situation and contains technical elements to be of interest for the AI community.

This paper is organized as follows. In section 2, we present a formal definition of *MS*, and the basics for cyptographic techniques used in the paper. In section 3, we provide the two cryptographic solutions. In section 4, we discuss performance issues of possible implementations. Finally, we conclude in section 5.

2. Preliminaries

2.1. Meeting Scheduling Problem

The *MS* problem [9] involves a set $A = \{a_1, a_2, ..., a_n\}$ of n agents, a set $M = \{m_1, m_2, ..., m_p\}$ of p meetings, a set $S = \{s_1, s_2, ..., s_r\}$ of r slots in any agent's calendar, and a set $P = \{p_1, p_2, ...p_q\}$ of q locations where these meetings can occur. Initially, agents may have several slots reserved for already filled planning in their calendars. A solution must assign a time and a location to each meeting, such that the following constraints are satisfied (i) all meeting attendees must agree *where* and *when* it will occur; (ii) m_i and m_j cannot be held at same time if they have one common attendee; (iii) each attendee a_i of meeting m_j must have enough time to travel from the location where he/she is before the meeting to the location where the meeting m_j will be.

MS is a truly distributed benchmark. Each attendee desires to keep private their already planned meetings, while he/she wants to achieve a globally consistent solution. This problem is very suitable to be treated by distributed techniques, trying to provide more autonomy to each agent while enforcing privacy.

To assess solving difficulty, we analyze four cases (from simple to general): (i) 1 meeting, 1 location, (ii) 1 meeting, q locations, (iii) p meetings, 1 location, and (iv) p meetings, q locations (general case). In the centralized setting (when all information is combined into a single server), the three first cases can be solved in polynomial time (for simplicity, we assume that every meeting lasts one slot). In case (i), each agent a_i has a vector of r slots defined as $v_{a_i}[k] = 1$ if slot k is available for the meeting at the location indicated, 0 otherwise. The conjunction of these vectors, $res[k] = \bigwedge_{i=1}^{n} v_{a_i}[k]$, indicates the solution: those slots k with $res[k] = 1$ are good for every agent to hold the meeting. A similar approach solves case (ii), where each agent has q of these vectors (one per location), the conjunction has to be done for each location p_j, producing $res^{p_j}[k]$. Those slots k and locations p_j such that $res^{p_j}[k] = 1$ are solutions. For case (iii), each agent has p of these vectors (one per meeting), the conjunction has to be done for each meeting m_j, obtaining $res^{m_j}[k]$. The number of 1's in the disjunction $\bigvee_{j=1}^{p} v_{a_i}[k]$ indicates the number of meetings that can be scheduled. Case (iv) involves the evaluation of a number of combinations which could be exponential in the worst case (a similar situation happens when meetings can last more than one slot in case (iii)).

Moving into the distributed realm, cases (i) and (ii) can be extended without difficulty, keeping their temporal complexities polynomial. For case (iii), this is more in-

volved. When every agent is involved in every meeting, extension is not difficult and this case is covered by our approach. Otherwise, there are some issues with the proposed cryptographic techniques, so we do not consider this case here. We define the Simple Meeting Scheduling as the *MS* when there is one meeting only in one or q possible locations when every agent is involved in that meeting. The distributed version occurs when the information is distributed among agents, as done in [3,4].

2.2. Homomorphic Encryption

A public key encryption scheme $PKE = (\mathcal{KG}, \mathcal{E}, \mathcal{D})$ consists of three probabilistic and polynomial time algorithms. The key generation algorithm $\mathcal{KG}$ takes as input a security parameter (for example, the desired length for the secret key) and outputs a pair (sk, pk) of secret and public keys. The encryption algorithm $\mathcal{E}$ takes as input a plaintext m and a public key pk, along with some randomness, and outputs a ciphertext $c = \mathcal{E}_{pk}(m)$. Finally, the decryption algorithm $\mathcal{D}$ takes as input a ciphertext and a secret key, and gives a plaintext $m = \mathcal{D}_{sk}(c)$ as output.

Such a scheme has an *homomorphic* property if there exist two operations, defined on the set of ciphertexts and plaintexts, such that the result of operating two ciphertexts is an encryption of the result of operating the two corresponding plaintexts. For example, a public key cryptosystem is *additively* homomorphic if there exists an operation $\oplus$ defined on the set of ciphertexts, such that the message encrypted in $c_1 \oplus c_2$ is $m_1 + m_2$, where m_i is the message encrypted in c_i, for $i = 1, 2$. Formally, this property is written as $\mathcal{D}_{sk}\big(\mathcal{E}_{pk}(m_1) \oplus \mathcal{E}_{pk}(m_2)\big) = m_1 + m_2$. Homomorphic cryptosystems have a lot of applications, including electronic auctions and electronic voting. An additively homomorphic encryption scheme allows re-randomization: if $c = \mathcal{E}_{pk}(m)$ is an encryption of m, then $c' = c \oplus \mathcal{E}_{pk}(0)$ is a new and random encryption of m. We will need a cryptosystem that also supports secure (t, t)-threshold decryption: the key generation algorithm does not output sk but a share sk_i for each member of some set $\mathcal{P} = \{P_1, \ldots, P_t\}$ of t users; the encryption algorithm is the same, and the decryption algorithm must be jointly performed by all the t users in $\mathcal{P}$.

Paillier's cryptosystem [10] satisfies all the properties that we want: it is additively homomorphic and it supports (t, t)-threshold decryption, as shown in [7].

2.3. (n, n)-Threshold Secret Sharing

Secret sharing is a cryptographic primitive that was independently introduced in [2,12] and that has been proved very useful in distributed scenarios, to distribute among a set of users the power of performing some secret operation (like decryption, signatures, etc.).

In the case of (n, n)-threshold secret sharing, a secret value s is distributed in shares $\{s_i\}_{i=1,\ldots,n}$ in such a way that all the n shares are necessary to recover the original secret s. An easy way to implement this primitive, for secrets $s \in \mathbb{K}$ in some finite field, is as follows: take $s_1, s_2, \ldots, s_{n-1} \in \mathbb{K}$ at random, and define $s_n = s - (s_1 + \ldots + s_{n-1})$. It is easy to see that the secret can be recovered as the sum of all the shares, but that $n - 1$ (or less) shares do not provide any information about the secret at all.

One interesting feature of this primitive is that it also provides homomorphic properties: if $\{s_1, \ldots, s_n\}$ is an (n, n)-threshold sharing of a secret s, and $\{t_1, \ldots, t_n\}$ is an (n, n)-threshold sharing of a secret t, then we have that $\{s_1 + t_1, \ldots, s_n + t_n\}$ is

an (n, n)-threshold sharing of the secret $s + t$. Therefore, each user can locally add his shares of different secrets to obtain a valid share of the sum of the secrets.

Considering n agents, this primitive can be used when an agent a_i can mask a secret input $x^{(i)}$ among the set of agents: he can compute an (n, n)-threshold sharing of this value, resulting in shares $\{x_1^{(i)}, \ldots, x_n^{(i)}\}$, and then broadcast all the shares but his own share $x_i^{(i)}$, that he keeps private. If users mask their secret inputs in this way, each user can then locally add his shares to obtain a share of the sum of all the secret inputs, for example.

3. Cryptographic Solutions For Distributed Simple Meeting Scheduling

3.1. Basic Cryptographic Sub-Protocols

The cryptographic skeleton of our solution basically makes use of three protocols: $\texttt{Mask}, \texttt{Comb}, \texttt{Unmask}$. Through protocol $\texttt{Mask}$, anyone can hide a secret input x; the masked version of x is a *ciphertext* $c = \texttt{Mask}(x)$. Through protocol $\texttt{Comb}$, different ciphertexts $\{c^{(j)} = \texttt{Mask}(x^{(j)})\}_{j=1,\ldots,n}$ are combined in such a way that the resulting ciphertext $c = \texttt{Comb}(c^{(1)}, \ldots, c^{(n)})$ is a masked version of $x^{(1)} + \ldots + x^{(n)}$. Finally, in protocol $\texttt{Unmask}$, all the n users must cooperate to obtain, from a ciphertext $c = \texttt{Mask}(x)$, the corresponding hidden secret x. We will write $x = \texttt{Unmask}(c)$.

These three basic protocols can be implemented by using either an homomorphic encryption scheme with threshold decryption (such as Paillier, see Section 2.2), or an (n, n)-threshold secret sharing scheme (see Section 2.3).

Specifically, when using an homomorphic encryption scheme $(\mathcal{KG}, \mathcal{E}, \mathcal{TD})$ with threshold decryption, one first runs the key generation part, in such a way that a public key pk is published, and the corresponding secret key sk is shared in pieces $\{sk_i\}_{i=1,\ldots,n}$, one for each of the n users. To mask a value x, everyone can compute and publish $c = \texttt{Mask}(x) = \mathcal{E}_{pk}(x)$. Since the encryption scheme is homomorphic, many ciphertexts can be combined by applying $\texttt{Comb}(c^{(1)}, \ldots, c^{(n)}) = c^{(1)} \oplus \ldots \oplus c^{(n)}$. Finally, since the decryption phase $\mathcal{TD}$ of the cryptosystem must be performed by all users, we can define $\texttt{Unmask}$ to be precisely this decryption: $\texttt{Unmask}(c) = \mathcal{TD}(c, \{sk_i\}_{i=1,\ldots,n})$.

When using an (n, n)-threshold secret sharing scheme, a user a_i in the group of n users can mask a secret input x by computing a sharing $\{x_1, x_2, \ldots, x_n\}$ of x, keeping secret his own share x_i and sending (publicly) to each other user a_j the share x_j. The protocol $\texttt{Comb}$, with inputs $c^{(j)} = \texttt{Mask}(x^{(j)})$ for $j = 1, \ldots, n$, can be run individually by each user a_ℓ, by adding locally $x_\ell^{(1)} + \ldots + x_\ell^{(n)}$ his shares of the values $x^{(j)}$. Finally, the protocol $\texttt{Unmask}$ consists in all the users publishing at the same time their shares of the secret, and then adding all these shares to obtain the corresponding secret.

With both solutions, we have that no information about a secret x is leaked from an execution of $c = \texttt{Mask}(x)$. Furthermore, since both solutions work for secret values which are in a finite field $\mathbb{K}$, a masked version $c = \texttt{Mask}(x)$ of a value $x \in \mathbb{K}$ can be randomized by combining it with a masked version $c' = \texttt{Mask}(0)$ of the secret 0.

Besides these three basic protocols, our cryptographic solutions to the problem of distributed meeting scheduling also invoke two other cryptographic sub-protocols, that we summarize right now.

3.1.1. Shuffling Masked Values

The input of this sub-protocol is a list $\{c_1, \ldots, c_k\}$ of k masked values, where $c_j = \texttt{Mask}(x_j)$. The output of this protocol is a different list $\{c'_1, \ldots, c'_k\}$ of masked values, which mask exactly the same set of secrets $\{x_1, \ldots, x_k\}$ but in a completely random order, unknown to any of the users. In other words, there exists a random and unknown permutation $\pi : \{1, 2, \ldots, k\} \rightarrow \{1, 2, \ldots, k\}$ such that $\texttt{Unmask}(c'_j) = x_{\pi(j)}$, for all $j = 1, \ldots, k$.

This sub-protocol can be easily implemented by an iterative process: each of the agents in A takes the previous list, applies to it a randomization of all the masked values (by combining them with masked versions of 0) and also a random permutation, and sends the resulting list to the following agent. The list resulting from the last agent is the output of the protocol.

We denote an execution of this protocol as $\{c'_1, \ldots, c'_k\} \leftarrow \texttt{Shuffle}(\{c_1, \ldots, c_k\})$.

3.1.2. Comparing Two Masked Values

The input of this sub-protocol consists of two masked values $c_1 = \texttt{Mask}(x_1)$ and $c_2 = \texttt{Mask}(x_2)$. The output is 1 if $x_1 < x_2$, and is 0 if $x_1 \geq x_2$. Nothing about the values x_1 and x_2 is revealed in an execution of this protocol, besides which value is greater.

This problem is known as the *millionaires problem*: two millionaires want to know who is richer without revealing the amount of their fortunes. It can be solved for the two scenarios: see [5,11] for the scenario based on homomorphic encryption with threshold decryption, and see [6] for the scenario based on (n, n)-threshold secret sharing.

The known solutions to this problem are quite inefficient, since they involve a lot interaction and computation by the agents: they have to compute masked versions of all the bits of x_1 and x_2, they have to jointly unmask some intermediate values, etc. Still, the computation and communication cost of this algorithm is polynomial on the number n of agents and the number of bits of x_1 and x_2. We will denote an execution of such a protocol as $b \leftarrow \texttt{Compare}(c_1, c_2)$, where b is a bit, $b \in \{0, 1\}$.

3.2. A Perfectly Private Solution

In this section we describe a solution that provides *perfect* privacy: if an attacker corrupts $n - 1$ agents, the only information that he obtains from an execution of the protocol is exactly what can be inferred from the private inputs of the corrupted agents and the final (public) output of the protocol. For example, if the output is that a unique meeting will take place in time s_3, then the attacker does not know if the non-corrupted agent is free in the rest of times or not. The different steps of the protocol are described below.

1. Setup phase: recall that the goal is to schedule a set $M = \{m_1\}$ of one meeting (attended by the n agents) inside a set $S = \{s_1, s_2, ..., s_r\}$ of r slots of time in the calendar. We will use index $i = 1, \ldots, r$ for the slots of time s_i. These sets M and S are public. Note that the granularity of the values s_i is not important: for example, slot s_2 can be 'Monday, May 11, from 12.00 to 13.00', slot s_4 can be 'Saturday, May 9, afternoon', etc.
2. The public cryptographic parameters are generated. In the case that (n, n)-threshold secret sharing is used, the finite field $\mathbb{K}$ must be published. In the case that Paillier's homomorphic public key encryption scheme is used, the public key

pk is made public, and each agent a_j receives as private input a share sk_j of the secret key.

3. For each time slot s_i, each agent a_j masks the value $m_{j,i} = 0$ if this option is NOT valid for him, and the value $m_{j,i} = 1$ if this option is valid for him. The resulting ciphertext is $c_{j,i} = \texttt{Mask}(m_{j,i})$. Agent a_j broadcasts the pairs $(i, c_{j,i})$, for $i = 1, ..., r$.
4. Once all the agents have done this, the ciphertexts of each option i are combined to result in a masked version of the sum of all the $m_{j,i}$, for all users j (if all the users agree in some option i, the resulting masked value will be n). The output of this phase are pairs (i, c_i), for $i = 1, \ldots, r$, where $c_i = \texttt{Comb}(c_{1,i}, \ldots, c_{n,i})$ is the combination of all the $c_{j,i}$.
5. One of the users masks the indices i for all the pairs. The output of this phase are pairs $\delta_i = (\texttt{Mask}(i), c_i)$, for $i = 1, \ldots, r$.
6. All the users cooperate to shuffle these pairs: $\{\delta'_1, \ldots, \delta'_r\} \leftarrow \texttt{Shuffle}(\{\delta_1, \ldots, \delta_r\})$.
7. By making calls to the protocol $b \leftarrow \texttt{Compare}(c_{i_1}, c_{i_2})$, the r pairs δ'_i are decreasingly ordered, according to the values masked in c_i. The output is an ordered list of pairs $\delta_i = (\texttt{Mask}(i), c_i)$.
8. Recall that the goal was to schedule 1 meeting. The first pair $\delta_{i_1} = (\texttt{Mask}(i_1), c_{i_1})$ is taken, and all the agents cooperate to synchronously unmask c_{i_1}, obtaining $m_{i_1} = \texttt{Unmask}(c_{i_1})$. If $m_{i_1} < n$, the output of the protocol is 'no solution'. Otherwise, if $m_{i_1} = n$, the corresponding index $\texttt{Mask}(i_1)$ is unmasked, and the resulting index is the output of the protocol.

Steps 3, 6, 7, 8 require the cooperation of all the agents. Step 5 is executed by one of the agents (randomly chosen). Finally, regarding Step 4, it depends on whether homomorphic threshold encryption or (n, n)-threshold secret sharing is being used. In the first case, a single agent (randomly chosen, as well) can perform the `Comb` routine, which is public in this case. In the case of (n, n)-threshold secret sharing, each agent must execute individually his part of the `Comb` routine, adding his private shares of the considered masked values.

3.3. A More Efficient, Non-Perfectly Private, Alternative

The protocol proposed in the previous section offers perfect privacy, but it is quite inefficient, especially because of Step 7, where the expensive sub-protocol `Compare` must be executed many times ($r \log(r)$ times in the worst case). Therefore, if efficiency is the main concern, one can consider an alternative protocol which avoids Step 7:

- Steps 1 - 6 are identical.
- Step 7. Agents take at random a pair $\delta'_i = (\texttt{Mask}(i)', c'_i)$ and jointly unmask c'_i, obtaining $m'_i = \texttt{Unmask}(c'_i)$. If $m'_i < n$, this pair is discarded. If $m'_i = n$, jointly unmask the associated $\texttt{Mask}(i)'$, and associate the meeting to this slot of time.

This second solution is of course much more efficient than the first one, but it can potentially reveal more information, as well. For example, imagine a toy example with $n = 4$ agents, $r = 3$ time options and $q = 1$ meeting to be scheduled. Suppose that the only option that is valid for the 4 users is the first one, s_1. Suppose that the values corresponding to the masked pairs $(\texttt{Mask}(i), c_i)$ are, for example: (1,4) for the time option s_1, (2,0) for the time option s_2, and (3,1) for the last time option, s_3. With this

alternative protocol, the agents maybe unmask all the values c_i, and they obtain a 1, then a 0 and finally a 4. Therefore, each of them can know something more about the availability of the others. For instance, if the first user voted NO for the option s_2 but yes for the two other options, as the count is 4, he immediately knows that everybody voted for the first (winning) option s_1, but nobody else voted for any of the two other options (option s_3 has count 1 and he voted for it, option s_2 has count 0 and nobody voted for it). In other words, none of the remaining agents could (or wanted) to meet in the slots of time s_2, s_3.

In practical situations where the number n of agents and the number r of time slots are quite large, this kind of situation is very unlikely to happen, and so this alternative (and much more efficient) protocol provides a high level of privacy at a reasonable cost (previous approach, although offers complete privacy, requires a high cost with could be prohibitive for many applications).

3.4. Extension To p Meetings

Trying to extend the proposed solutions to case (iii) of section 2, when there are p meetings to be scheduled in one location, we differentiate between two cases: (a) when every agent is involved in every meeting, and (b) when not every agent is involved in every meeting. In case (a), the proposed approaches (either with perfect privacy or the alternative solution) can be easily extended to include this case. Basically, you keep decoding pairs $(\texttt{Mask}(i), c_i)$ in the decreasing order of the list (in the perfect private solution) or randomly (in the alternative solution). When you have p slots with unmasked ciphertexts equal to n, you have a complete solution. If the p meetings cannot be scheduled, perhaps it is of interest to schedule as many meetings as possible, instead of stopping with the 'no solution' output. In the perfect privacy solution, agents should take the first element in the ordered list, unmask c_{i_1}, test if it is equal to n and if so, place a meeting in the associated slot of time i_1, and move to the following element. This process would stop when some value masked at some c_{i_s} is less than n. Alternatively, you have to scan the list of pairs until p meetings have been scheduled or the whole list has been exhausted.

In case (b), the proposed solutions do not work: each pair contains the number of agents free for that slot, but not the names of those agents. To guarantee privacy, a different approach should be taken.

4. Algorithmic Performance

Solving methods for the distributing simple meeting scheduling are polynomial, and the cryptographic formulation is also polynomial. So the proposed solution remains polynomial. However, the addition of cryptography includes some overhead which could not be neglected from the practical point of view. In this section we want to address the practical efficiency of hypothetical implementations of this approach.

As we have explained before, the most costly part of the cryptographic algorithms presented in this paper is the comparison of two masked values. The second non-perfectly private algorithm is designed to remove this operation from the algorithm, of course, at cost of compromise the privacy of the agents.

Another important issue to be considered in the algorithms is the complexity added by the cryptosystem. In general, cryptographical operations of public key systems have

both a large message expansion and a large CPU computational cost, causing a significant increment in communication cost and in computation effort. However, apart from the comparison operation, all the operations presented here have a polynomial time cost, making the proposed algorithms suitable for real scenarios. Moreover, in [1] some improvements were proposed in order to reduce the message expansion of the Paillier from $O(N^2)$ to $O(N)$ where N is the product of two prime numbers p and q needed in the generation key algorithm. These prime numbers are large because the security of the Paillier method is proportional to the length of those prime numbers (typically, the size of prime numbers used in cryptography range from 128 bits, offering low security, to 1024 bits, offering high security). So the suggested reduction allows us to use the cryposystem with large keys without adding communication complexity quadratic in the message size.

5. Conclusions

In this paper we have presented a new algorithm for solving the private distributed meeting scheduling problem for the case where there is only one possible location for doing the meetings and all the agents participate in all the scheduled meetings. We have showed that a perfectly private solution is this scenario is possible but inefficient. For this reason, we have also presented another non-completely private solution but very efficient.

We have also showed that designing a perfectly private solution for more general cases, with more than one location or where not all the agents must attend all the meetings, is left as an interesting (although apparently hard) open problem.

References

[1] D. Catalano, R. Gennaro, N. Howgrave-Graham and P.Q. Nguyen, Paillier's Cryptosystem Revisited, *Proc. of the 8th ACM Conf. on Computer and Communications Security*, 206-214, 2001.

[2] G.R. Blakley. Safeguarding cryptographic keys. *Proceedings of the National Computer Conference, American Federation of Information*, Processing Societies Proceedings **48**, 313–317, 1979.

[3] I. Brito and P. Meseguer. Distributed Meeting Scheduling. *Proc. CCIA-07*, 2007.

[4] I. Brito and P. Meseguer. Privacy in Distributed Meeting Scheduling. *Proc. CCIA-08*, 2008.

[5] R. Cramer, I. Damgård and J.B. Nielsen. Multiparty computation from threshold homomorphic encryption. *Proceedings of Eurocrypt'01*, LNCS **2045**, Springer-Verlag, 280–299, 2001.

[6] I. Damgård, M. Fitzi, E. Kiltz, J.B. Nielsen and T. Toft. Unconditionally secure constant-rounds multiparty computation for equality, comparison, bits and exponentiation. *Proceedings of TCC'06*, LNCS **3876**, Springer-Verlag, 285–304, 2006.

[7] P.A. Fouque, G. Poupard and J. Stern. Sharing decryption in the context of voting or lotteries. *Proceedings of Financial Cryptography'00*, LNCS **1962**, Springer-Verlag, 90–104, 2001.

[8] M. S. Franzin, F. Rossi, E. C. Freuder and R. Wallace. Multi-Agent Constraint Systems with Preferences: Efficiency, Solution Quality, and Privacy Loss. *Computational Intelligence*, **20**, 264–286, 2004.

[9] E.C. Freuder, M. Minca and R.J. Wallace. Privacy/efficiency trade-offs in distributed meeting scheduling by constraint-based agents. *Proc. of DCR Workshop at IJCAI-01*, 63–71, USA, 2001.

[10] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. *Proceedings of Eurocrypt'99*, LNCS **1592**, Springer-Verlag, 223–238 1999.

[11] B. Schoenmakers and P. Tuyls. Efficient binary conversion for Paillier encrypted values. *Proceedings of Eurocrypt'06*, LNCS **4004**, Springer-Verlag, 522–537, 2006.

[12] A. Shamir. How to share a secret. *Communications of the ACM*, vol. **22**, 612–613, 1979.

[13] M. C. Silaghi. Meeting Scheduling Guaranteeing n/2-Privacy and Resistant to Statistical Analysis (Applicable to any DisCSP). *Proc. of the 3th Conference on Web Intelligence*, 711–715, 2004.