ON SECURELY SCHEDULING A MEETING

Thomas Herlea, Joris Claessens, Bart Preneel
COmputer Security and Industrial Cryptography (COSIC)
Dept. of Electrical Engineering – ESAT
Katholieke Universiteit Leuven
Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
firstname.lastname@esat.kuleuven.ac.be
http://www.esat.kuleuven.ac.be/cosic/

Gregory Neven, Frank Piessens, Bart De Decker
Dept. of Computer Science
Katholieke Universiteit Leuven
Celestijnenlaan 200A, B-3001 Leuven-Heverlee, Belgium
firstname.lastname@cs.kuleuven.ac.be
http://www.cs.kuleuven.ac.be/cwis/research/distrinet/

Abstract When people want to schedule a meeting, their agendas must be compared to find a time suitable for all participants. At the same time, people want to keep their agendas private. This paper presents several approaches which intend to solve this contradiction. A custom-made protocol for secure meeting scheduling and a protocol based on secure distributed computing are discussed. The security properties and complexity of these protocols are compared. A trade-off between trust and bandwidth requirements is shown to be possible by implementing the protocols using mobile agents.

Keywords: mobile agents, secure distributed computation, meeting scheduling

1. INTRODUCTION

When negotiating meetings, the participants look up, communicate and process information about each other's agendas trying to find a moment when they are all free to attend the meeting. Due to the private nature of a person's schedule, as little as possible should be revealed to any other party during that negotiation. Ideally, only the result of the negotiation should be known to the participants (and to the participants only), and any other information about the users' agendas should remain secret.

An easy solution for scheduling a meeting is to broadcast the schedules to all participants, but this totally neglects the privacy of the participants' agendas. Another solution is to send all schedules to a trusted third party, but finding one such single third party trusted by every participant, will be very difficult in practice.

Some existing meeting scheduling applications, like for example "Yahoo! Calendar", define access levels for viewing and modifying agenda entries, and define user groups to which these access levels are assigned. This is only necessary because the comparison between schedules must be done by the users themselves. Our approaches eliminate the need for managing access control, as they are not based on users directly accessing each other's agenda.

This paper presents more secure solutions. Their goal is for participants to be able to negotiate a meeting whereby parties have no direct access to each other's agenda, whereby parties do not rely on another party for telling the final result, and whereby no information about the agendas is revealed, but the final result, i.e., the particular time the meeting can be scheduled, or the fact that the meeting cannot be scheduled.

This paper builds on the work done in [6] and [3] and shows the tradeoffs that can be made in security, level of trust, and efficiency, when choosing a particular negotiation protocol and a specific implementation approach.

The paper is organized as follows. Section 2 presents a custom-made negotiation protocol. Section 3 presents an alternative approach based on secure distributed computing. Both approaches are analyzed from a security and complexity point of view. Section 4 discusses the use of mobile agents for secure meeting scheduling, and presents the "agenTa" prototype implementation. We conclude in Sect. 5.

2. USING A CUSTOM-MADE NEGOTIATION PROTOCOL

2.1. DATA REPRESENTATION

There exists a representation which reduces the problem of deciding if the meeting can be scheduled at a certain moment to a logical AND operation.

As shown in Fig. 1, an agenda will be represented as a bit string in the following way: for each time slot in the schedule, there is one bit indicating whether the negotiator can (1) or cannot (0) attend a

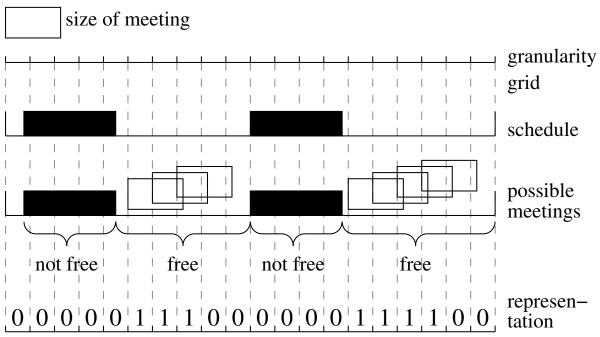

Figure 1 Conversion from agenda to representation

meeting of the specified length which would start at that time. The finer the granularity and the longer the negotiation window, the more bits there will be in the representation.

2.2. SCHEDULING MODEL

In our model, a meeting scheduling starts with an invitation phase. The initiator broadcasts to the invitees a set of negotiation parameters such as meeting length, negotiation window (limited time span in which to attempt the meeting scheduling) and a complete list of invitees. Each invitee broadcasts to all others a reply indicating whether it will accept or decline the negotiation invitation. Because broadcasts are used, no invitee can be mislead as to the set of negotiators it will encounter in the second phase.

In the second phase, called negotiation, the negotiators try the time slots one by one and attempt to schedule the meeting. For each time slot the negotiation takes place according to the protocol outlined below. If the meeting was successfully scheduled the negotiators move on to the third phase, otherwise the next time slot is tried. After independently arriving to a result concerning a certain time slot, each participant broadcasts the result to the others and checks whether all results coincide. This allows for detection of partial failures and attacks which try to mislead a subset of the negotiators.

In the third phase either the common result is presented to the users, or the users are informed that no meeting can take place. If there is a common result, users might confirm their commitment to the scheduled time on a separate channel (e-mail, telephone), independently of the scheduling process.

2.3. SCHEDULING A MEETING

For the purpose of this subsection we will refer to the representation of an agenda according to the description in the previous subsection as "schedule."

Instead of comparing schedules, the negotiation should be based on comparing protected forms of the schedules. The schedules are protected in a way which still allows scheduling to be performed by broadcasting the protected forms to all negotiators and letting them process the data without fear of the unprotected form to be revealed.

The binary XOR operation between the schedule and a mask is a transformation which still allows scheduling to be performed in the sense that the (in)equality of two or more bits is preserved when they are all XORed with the same mask.

If all negotiators know the mask, they are able to retrieve the original schedules easily, by unmasking the broadcasted data. The solution is to let the mask be a shared secret, that is, all negotiators will contribute when building it, but it will not be revealed to any of them.

The negotiation protocol then goes as follows:

1 In step one of the negotiation protocol, each negotiator chooses a random mask, and XORs it with its schedule. This random mask is actually a partial mask. The shared secret will be the XOR of all partial masks, and is called global mask. Even if only one negotiator keeps its partial mask secret, the others cannot find the global mask solely using their partial masks.

2 In step two of the protocol, all schedules visit all negotiators exactly one time. At each visit they are masked with the partial mask of that particular negotiator. In the end, all original schedules are thus masked with the global mask, without the need for the negotiators to disclose their partial mask. Since the schedule is first masked with its owner's partial mask it remains secret during its visits.

  A negotiator must be unable to identify a protected schedule as representing its own schedule: otherwise performing XOR between the original and the protected schedule reveals the global mask, allowing the negotiator to retrieve all original schedules. Therefore during the trip to all negotiators, the schedule must be forwarded randomly between the negotiators in order to make it impossible to trace. The schedule must have attached a list of negotiators it hasn't visited yet, decremented at each forwarding, in order to prevent multiple maskings with the same partial mask.

Note that for countering attempts to trace a schedule by attackers who have a global view on the network, all communications should be encrypted.

3 In step three, all protected schedules are broadcasted. Each negotiator looks independently for a time slot when all protected schedules have the same value. That implies that the original schedules are identical, too, for that time slot but does not provide any clue whether the negotiators are free or busy for that time slot. The clue is provided by each negotiator's schedule for that time slot. If the negotiator is free then, it means all negotiators are free then and the meeting can be scheduled. For time slots when some are busy and some are free, it is not possible to figure out who are the busy ones and who are the free ones.

Note that our scheduling protocol does not specify any form of negotiator authentication. This is however needed for linking the protocol messages to their originators. Depending on the meeting application, the desired form of authentication can be added to the protocol.

Figure 2 shows the negotiation protocol as performed by three parties. For easy understanding of the protocol the schedules in the simulation are following the same route and the maskings appear to be performed simultaneously by the three negotiators. In reality the process is asynchronous (some negotiators may be idle while others are masking) and routing is random (in the end some negotiators may have nothing to broadcast while others may broadcast several protected schedules). Another difference is that in reality only one bit is processed at a time (otherwise an attack is possible, see following section). If the meeting can be scheduled in the corresponding time slot the protocol stops, otherwise the next time slot is processed.

2.4. SECURITY ANALYSIS

Our custom-made protocol does not require one single entity to be trusted. It however does not completely protect the privacy of the participants' agenda, as attacks by both passive and active adversaries are possible.

Bad slots. There may be time slots for which all users are busy and therefore all protected slots will be equal. By checking against the original schedule each negotiator will avoid scheduling a meeting in that slot but it will also know everybody else's schedule for that slot (i.e., everybody is busy). Because they constitute an infringement on all users' privacy we call these slots *bad slots*.

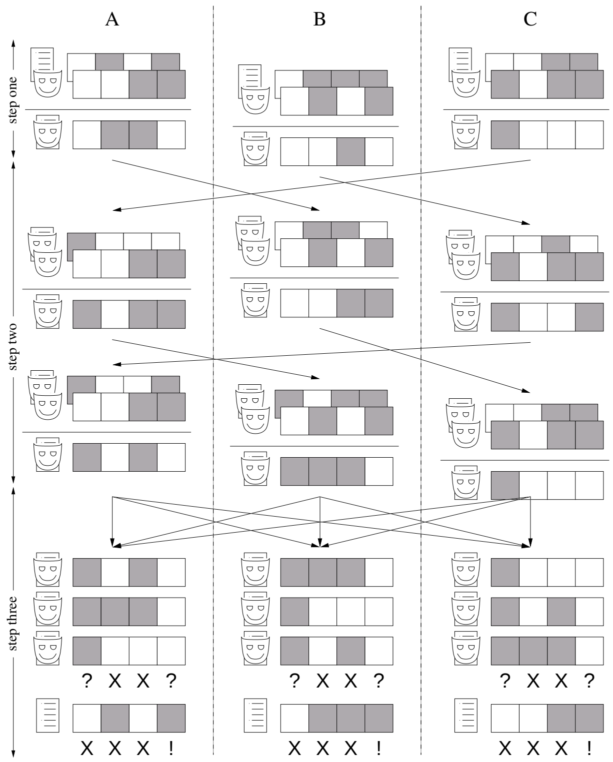

Figure 2 Simulation for three negotiators

Entropy attack. The reason for performing the negotiation one slot at a time is to prevent the following attack. If the negotiation is done on sequences of slots, when all the broadcasted masked schedules are received, it still is possible for a party to recognize its original schedule. It can be done by testing all the masks which transform the original schedule into one of the protected forms. The correct global mask can be recognized by the fact that by unmasking the other protected schedules with it, bit strings are obtained which have the entropy expected from a schedule.

Negotiating one bit at a time, with fresh partial masks for each bit and stopping when a meeting is scheduled counters this attack because each mask bit and schedule bit have maximal entropy.

Number of parties. When only two parties are negotiating, each can deduce the schedule of the other based on their own schedule and the comparison between the protected forms of the schedules. Besides that, the global mask is straightforward to find because the original schedule can be linked to its protected form. Also when only three or four parties are negotiating it is sometimes possible to find out the global mask by tracing back schedules. For five or more participants the ability to trace a schedule along its route decreases as the number of participants increases.

Dummy negotiators could be introduced to artificially increase the number of parties, and thus to alleviate this problem. In a broadcasting communication environment encrypted dummy messages could also be sent to make the real schedules untraceable.

Rogue negotiators. Active adversaries could attack the protocol in various ways.

A simple denial of service attack can be mounted by negotiating based on a fully busy schedule instead of declining the invitation. Since the protocol relies on the negotiators consistently using their partial mask, the protocol has unpredictable outcomes if a negotiator randomly changes its partial mask during the negotiation of a time slot.

Goal-oriented misbehavior is also possible. A negotiator can wait to be the last to broadcast the protected schedule(s) it has. This way it is able to detect first when a meeting could take place. In that case it can broadcast a false protected form, preventing the meeting from being scheduled. It knows everybody else's schedule for that time slot, while the others do not.

2.5. COMPLEXITY

For analyzing the complexity of the scheduling we count the messages that are sent between the negotiators. In a distributed environment it is expected that sending messages will be much more resource consuming than masking or a comparison between bits. Since much of the processing is done in parallel, bandwidth is more important. Remember that the negotiation protocol is performed bit by bit.

Note that for n negotiators a broadcast is of complexity $n-1$. When an all-to-all broadcast is needed it has complexity $n(n-1)$.

The scheduling starts with a simple broadcast of the invitation. $C_1 = n-1$. All negotiators (except for the initiator) must announce their position towards the invitation. These broadcasts adds complexity $C_2 = (n-1)(n-1)$. For getting masked, one bit must visit all negotiators and then be broadcasted: $2(n-1)$. This happens to each negotiator's bit in a round: $C_3 = 2n(n-1)$. If the number of bits in a schedule is l, after at most l rounds the protocol will end. In the check phase of the scheduling, all negotiators broadcast their result or the fact that no meeting could be scheduled to all others: $C_4 = n(n-1)$. Note that only positive results (i.e., a meeting is possible) are broadcasted. If the result is negative, the agents automatically go to the next bit. If the result is still negative after the last bit, it was not possible to schedule a meeting.

Therefore at most $C = C_1 + C_2 + lC_3 + C_4 = (1+n-1+2nl+n)(n-1) = (2+2l)n(n-1)$ messages are sent. For example, for a scheduling window of 3 eight-hour working days, granularity 1 hour ($l = 24$) and 5 participants ($n = 5$) this amounts to at most 1000 messages; for 10 participants in the same conditions, there will be up to 4500 messages sent.

3. USING SECURE DISTRIBUTED COMPUTING

3.1. THE PROBLEM OF SECURE DISTRIBUTED COMPUTING

Usually, the problem of Secure Distributed Computing (SDC) is stated as follows. Let f be a publicly known function taking n inputs, and suppose there are n different parties, each holding their own private input x_i $(i = 1 \dots n)$. The n parties want to compute the value $f(x_1, \dots, x_n)$ without leaking any information about their private inputs to the other parties (except of course the information about x_i that is implicitly present in the function result). In descriptions of solutions to the Secure Distributed Computing problem, the function f is usually encoded as a boolean circuit, and therefore Secure Distributed Computing is also often referred to as *secure circuit evaluation*.

Over the past two decades, a fairly large variety of solutions (other than the trivial one using a trusted third party) to the problem has been proposed. An overview is given by Franklin [4] and more recently by Cramer [2].

3.2. HOW TO PERFORM GENERAL SECURE DISTRIBUTED COMPUTING

The core problem of SDC is that we want to perform computations on hidden data (using encryption, secret sharing or other techniques) without revealing the data. One class of techniques to compute with encrypted data is based on *homomorphic probabilistic encryption*. An encryption technique is *probabilistic* if the same cleartext can encrypt to many different ciphertexts under the same encryption key. To work with encrypted bits, probabilistic encryption is essential, otherwise only two ciphertexts (the encryption of a zero and the encryption of a one) would be possible, and cryptanalysis would be fairly simple. An encryption technique is *homomorphic* if it satisfies at least one equation of the form $E(x \quad \mathbf{op} \quad y) = E(x) \quad \mathbf{op'} \quad E(y)$ for some operations $\mathbf{op}$ and $\mathbf{op'}$. A homomorphic encryption scheme allows operations to be performed on encrypted data, and hence is suitable for secure circuit evaluation.

In [5], Franklin and Haber present a protocol that evaluates a boolean circuit on data encrypted with such a homomorphic probabilistic encryption scheme. In order to support any number of participants, they use a *group oriented* encryption scheme, i.e., an encryption scheme that allows anyone to encrypt, but that needs the cooperation of all participants to decrypt. In the group oriented encryption scheme used by Franklin and Haber, a bit b is encrypted for a group of participants $S \subseteq \{1 \ldots n\}$ as

$$E_S(b) = \left[g^r \mod N,\ (-1)^b \Big(\prod_{j \in S} g^{K_j} \Big)^r \mod N \right]$$

where $N = pq$, p and q are two primes such that $p \equiv q \mod 4$, and $r \in_R Z_N$. The public key is given by $[N, g, g^{K_1} \mod N, \ldots, g^{K_n} \mod N]$, while K_i is the private key of the ith participant. This scheme has some additional properties that are used in the protocol:

- *XOR-Homomorphic.* Anyone can compute a joint encryption of the XOR of two jointly encrypted bits. Indeed, if $E_S(b) = [\alpha, \beta]$ and $E_S(b') = [\alpha', \beta']$, then $E_S(b \oplus b') = [\alpha\alpha' \mod N, \beta\beta' \mod N]$.
- *Blindable.* Given an encrypted bit, anyone can create a random ciphertext that decrypts to the same bit. Indeed, if $E_S(b) = [\alpha, \beta]$ and $r \in_R Z_N$, then $\left[\alpha g^r \mod N,\ \beta \big(\prod_{j \in S} g^{K_j}\big)^r \mod N\right]$ is a joint encryption of the same bit.
- *Witnessable.* Any participant can withdraw from a joint encryption by providing the other participants with a single value. Indeed, if $E_S(b) = [\alpha, \beta]$, it is easy to compute $D_i(E_S(b))$ from $W_i([\alpha,\ \beta]) = \alpha^{-K_i} \mod N$

First of all, the participants must agree on a value for N and g, choose a secret key K_i and broadcast $g^{K_i} \mod N$ to form the public key. To start the actual protocol, each participant broadcasts a joint encryption of his own input bits. To evaluate an XOR-gate, everyone simply applies the XOR-homomorphism. The encrypted output of a NOT-gate can be found by applying the XOR-homomorphism with a default encryption of a one, e.g. $[1, -1 \mod N]$.

The encryption scheme is not AND-homomorphic, so the evaluation of an AND-gate will be more troublesome. Suppose the encrypted input bits for the AND-gate are $\hat{u} = E(u)$ and $\hat{v} = E(v)$. To compute a joint encryption $\hat{w} = E(w) = E(u \wedge v)$, they proceed as follows:

1 Each participant i chooses random bits b_i and c_i and broadcasts $\hat{b}_i = E(b_i)$ and $\hat{c}_i = E(c_i)$.

2 Each participant repeatedly applies the XOR-homomorphism to calculate $\hat{u}' = E(u') = E(u \oplus b_1 \oplus \ldots \oplus b_n)$ and $\hat{v}' = E(v') = E(v \oplus c_1 \oplus \ldots \oplus c_n)$. Each participant broadcasts decryption witnesses $W_i(\hat{u}')$ and $W_i(\hat{v}')$.

3 Everyone can now decrypt $\hat{u}'$ and $\hat{v}'$. By repeatedly applying the fact that $(a \oplus b) \wedge c = (a \wedge c) \oplus (b \wedge c)$, one can prove that $w' = u' \wedge v' = (u \wedge v) \oplus w_1 \oplus \cdots \oplus w_n$ where $w_i = (u \wedge c_i) \oplus (b_i \wedge c_1) \oplus \cdots \oplus (b_i \wedge c_n) \oplus (b_i \wedge v)$.

   Each participant is able to compute a joint encryption of w_i: he knows b_i and c_i (he chose them himself) and he received encryptions $\hat{c}_j$ from the other participants, so he can compute $E(b_i \wedge c_j)$ as follows: if $b_i = 0$, then $b_i \wedge c_j = 0$, so any default encryption for a zero will do, e.g. $[1, 1]$. Otherwise, if $b_i = 1$, then $b_i \wedge c_j = c_j$, so $\hat{c}_j$ is a valid substitution for $E(b_i \wedge c_j)$.

   $E(u \wedge c_i)$ and $E(v \wedge b_i)$ can be computed in an analogous way. He uses the XOR-homomorphism to combine all these terms, blinds the result and broadcasts this as $\hat{w}_i$.

4 Each participant combines $\hat{w}'$ and $\hat{w}_j$ $(j = 1 \ldots n)$, again using the XOR-homomorphism, to form $\hat{w} = E(w)$.

When all gates in the circuit have been evaluated, every participant has a joint encryption of the output bits. Finally, the participants broadcast decryption witnesses for the output bits to reveal them.

3.3. SECURE MEETING SCHEDULING USING SDC

We already showed how to reduce the problem of scheduling a meeting for n secret agendas to a series of logical AND operations on n secret bits. For every time slot in the schedule, each negotiator has one secret input bit: a one if he is available to start the meeting at that time, a zero if he isn't. Because the Secure Distributed Computing protocol we just discussed can only handle binary gates, we implement the n-ary AND operation as a $\log_2(n)$-depth tree of binary AND-gates. The output bit of the circuit indicates if this slot is an appropriate starting time for the meeting (1) or not (0).

3.4. SECURITY ANALYSIS

Franklin and Haber show that their protocol is provably secure against passive adversaries (i.e., adversaries who follow the rules of the protocol, but who try to learn as much information from the communication as possible), given that ElGamal encryption with a composite modulus is secure. This means that under the assumption of passive adversaries, complete privacy of all agendas is guaranteed (except of course for the fact that everybody is available at the time the meeting is scheduled). However, the proof Franklin and Haber give uses a more complicated encryption scheme and they mention the one we used here as an alternative. To the best of our knowledge, the security of this encryption scheme is still an open problem.

The protocol is not provably secure against active adversaries (who can deviate from the protocol). For example, a malicious participant can flip the output of an AND gate by XORing his $\hat{m}_i$ with the encryption of a one. For this particular application however, the most obvious attacks don't seem to give rise to substantial information leaks. The SDC protocol presented by Chaum, Damgård and van de Graaf in [1] provides provable security against active adversaries at the cost of higher bandwidth requirements.

3.5. COMPLEXITY

Let's have a closer look at the message complexity of this protocol. The same public and private keys can be used for every evaluation. This means that the initiator's invitation message can contain N and g ($C_1 = n-1$ messages), while g^{K_i} can be wrapped together with the message that announces each participant's position towards the invitation ($C_2 = (n-1)(n-1)$ messages).

The evaluation of a single AND gate consists of four phases, of which the first three need an all-to-all broadcast (consuming $n(n-1)$ messages each) while the last one doesn't need any communication. Since the AND gates within one level of the tree can be evaluated in parallel, the evaluation of the entire circuit takes $C_3 = \lceil \log_2(n) \rceil \cdot 3n(n-1)$ messages. The broadcast of the encrypted input bits of the circuit and the broadcast of decryption witnesses for the output bit both take another $C_4 = n(n-1)$ messages.

If l slot evaluations are needed before a suitable meeting time is found, the total message complexity is given by $C_1 + C_2 + l(C_3 + 2C_4) = n(n-1)(1 + l(2 + 3\lceil \log_2(n) \rceil))$. If we consider the same example as we did in the previous section ($l = 24$), this amounts to 5300 messages for 5 participants and 30330 messages for 10 participants.

Before comparing this result to that of the custom-made protocol in the previous section, we should notice that only the number of messages is taken into account, not their size. As $|N|$ should be about 1024 bits to be secure, the messages in the SDC protocol will be larger than the messages in the custom-made protocol. However, since the maximum message length for 10 participants is only 2.5 KB (which easily fits into a single IP packet), we considered the number of transmitted messages more relevant than the number of bits that are strictly needed.

It should also be noted that we do not take into account computation or memory overhead for the protocols. The amount of computation and storage needed for the SDC protocol is considerably higher than for the custom-made protocol.

4. USING MOBILE AGENTS

In this section, it will be shown how mobile agents can be used to reduce the communication overhead of the two solutions for the agenda scheduling problem. The basic idea is to use mobility to bring agents of the participants closer together. Of course, a mobile agent needs to trust his execution platform but we will show that the trust requirements are less strong than for a classical trusted third party (TTP) solution for the meeting scheduling problem.

To compare the trust requirements of the different approaches, we use the following simple trust model. We say a participant *trusts* an execution site if it believes that: (1) the execution site will correctly execute any code sent to it by the participant; (2) the execution site will correctly (i.e., as expected by the participant) handle any data sent to it by the participant. It also implies that the execution site will maintain the privacy of the data or the code if this is expected by the participant. If p trusts E, we denote this as shown in Fig. 3.

Figure 3 Notation for "p trusts E"

To compare bandwidth requirements (for communication overhead), we make the following simple distinction. *High* bandwidth is required to execute one of the discussed protocols. *Low* bandwidth suffices to transmit data or agent code. Also intermittent connections (e.g. for devices that are sometimes disconnected from the network) are considered low bandwidth. We assume low bandwidth communication is available between any two parties. If high bandwidth communication is possible between E_i and E_j, we denote this as shown in Fig. 4.

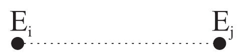

Figure 4 Notation for high bandwidth connection between E_i and E_j

Based on these simple models of communication and trust, we compare three options for implementing secure meeting scheduling.

4.1. A TRUSTED THIRD PARTY

The first, perhaps most straightforward option, is to use a globally trusted third party. Every participant sends its agenda to the TTP who will compute an appropriate meeting time and disseminate the result to the participants. Of course, data must be sent to the TTP, through an authenticated and safe channel. This can be accomplished via conventional cryptographic techniques.

It is clear that this approach has a very low communication overhead: the data is only sent once to the TTP; later, every participant receives the result of the computation. However, every participant should unconditionally trust the TTP. For the case of 4 participants, the situation is as shown in Fig. 5.

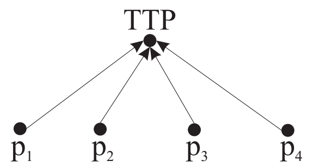

Figure 5 Situation with 4 participants and a TTP.

It is not clear whether n distrustful participants will easily agree on one single trustworthy third party. This requirement of one single globally trusted execution site is the main disadvantage of this approach.

4.2. CRYPTOGRAPHIC SECURE MEETING SCHEDULING

The second option is the use of cryptographic techniques (as discussed in previous sections) that make the use of a TTP superfluous.

The trust requirements are really minimal: every participant only trusts its own execution site.

Although this option is very attractive, it should be clear from the previous sections that the communication overhead might be too high to be practically useful in a general networked environment. High bandwidth is required between all of the participants. For the case of 4 participants, the situation can be summarized as shown in Fig. 6.

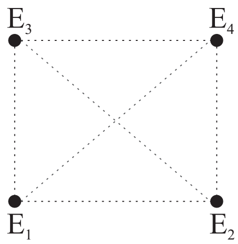

Figure 6 Situation with 4 participants without a TTP.

4.3. USING MOBILE AGENTS

Finally, a third solution tries to combine the two previous options: the communication overhead is remedied by introducing semi-trusted execution sites and mobile agents.

In this approach, every participant p_i sends its representative, agent a_i, to a trusted execution site E_j. The agent contains a copy of the agenda and is capable of running a secure meeting scheduling protocol.

It is allowed that different participants send their agents to different sites. The only restriction being that the sites should be located closely to each other, i.e., should have high bandwidth communication between them.

The amount of long distance communication is moderate: every participant sends its agent to a remote site, and receives the result from its agent. The agents use a cryptographic protocol, which unfortunately involves a high communication overhead. However, since the agents are executing on sites that are near each other, the overhead of the protocol is acceptable. For a situation with 4 participants, we could have the situation as depicted in Fig. 7.

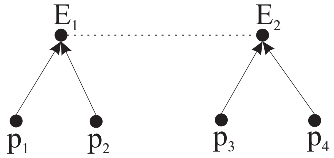

Figure 7 Situation with 4 participants using mobile agents

No high bandwidth communication between the participants is necessary, and there is no longer a need for one single trusted execution site.

4.4. CURRENT IMPLEMENTATION WITH AGLETS

"agenTa" is the name of our prototype implementation of a secure meeting scheduling system. Currently it uses the custom-made protocol described in this paper.

We have used the Aglets SDK 1.1 beta 3 [7], a mobile agents system development kit which was released to the open source community by its creator, IBM. The SDK contains an agent server, the API needed to write agents in Java (called aglets), examples and documentation. The prototype implementation of agenTa has around 3500 lines of Java code.

For the inter-agent communication KQML (Knowledge Query and Manipulation Language) was chosen. KQML was developed at the University of Baltimore Maryland County, and enhanced with security capabilities in [8].

In our implementation, each user's scheduling application is modular, the user interface, the agenda management and the negotiation being performed by distinct intercommunicating aglets. Only the negotiator aglets of all users take advantage of their mobility to gather on a host where they carry out the negotiation protocol by local communication.

There are no language limitations for implementing the custom-made protocol. Communication relies on transmitting character strings. Therefore, agents implemented with other agent platforms and in other programming languages can take part in the negotiation, provided the platforms can interoperate.

5. CONCLUSION

This paper has shown that there exist several techniques for secure meeting scheduling. Moreover, a trade-off can be made between the level of security that can be obtained, the degree of trust that is required, and the amount of overhead that is caused by the protocol.

When a TTP is used, a meeting can be scheduled very efficiently. The custom-made protocol has more overhead, but does not require trust in a third party. An SDC protocol is more secure than our custom-made protocol, but it is also much less efficient.

Using mobile agents when implementing any protocol can improve the efficiency, while still avoiding the need for one single trusted entity.

Acknowledgements

This work was supported in part by the FWO-Vlaanderen project G.0358.99 and by the Concerted Research Action (GOA) Mefisto-666. Joris Claessens is funded by a research grant of the Flemish Institute for the Promotion of Industrial Scientific and Technological Research (IWT), Gregory Neven is an FWO-Vlaanderen Aspirant, and Frank Piessens is an FWO-Vlaanderen Postdoctoral fellow.

References

[1] D. Chaum, I. Damgård, J. van de Graaf, "Multiparty computations ensuring privacy of each party's input and correctness of the result." In C. Pomerance, ed., *Advances in Cryptology—CRYPTO '87 Proceedings*, Lecture Notes in Computer Science, LNCS 293, pp. 87–119, Springer-Verlag, New York, 1988.

[2] R. Cramer, "An introduction to secure computation." In I. Damgård, ed., *Lectures on Data Security*, Lecture Notes in Computer Science, LNCS 1561, pp. 16–62, 1999.

[3] B. De Decker, F. Piessens, E. Van Hoeymissen, G. Neven, "Semi-trusted Hosts and Mobile Agents: Enabling Secure Distributed Computations." In E. Horlait, ed., *Mobile Agents for Telecommunication Applications*, Lecture Notes in Computer Science, LNCS 1931, pp. 219–232, Springer-Verlag, 2000.

[4] M. Franklin, "Complexity and security of distributed protocols." Ph.D. thesis, Computer Science Department of Columbia University, New York, 1993.

[5] M. Franklin and S. Haber, "Joint encryption and message-efficient secure computation." Journal of Cryptology, 9(4), pp. 217–232, Autumn 1996.

[6] T. Herlea, J. Claessens, D. De Cock, B. Preneel, J. Vandewalle, "Secure Meeting Scheduling with agenTa." *Proceedings of IFIP Communications and Multimedia Security*, 2001.

[7] Danny B. Lange, Mitsuru Oshima, "Mobile agents with Java: The Aglet API." http://www.genmagic.com/asa/danny/Wwwj.pdf.

[8] Chelliah Thirunavukkarasu, Tim Finin, James Mayfield, "Secret Agents - A Security Architecture for the KQML Agent Communication Language." http://www.cs.umbc.edu/kqml/papers/secret.ps.