Possible Attacks on and Countermeasures for Secure Multi-Agent Computation

ABSTRACT

In this paper we improve the model for secure multi-agent computation proposed by Endsuleit and Mie in [7]. Instead of Canetti [6] we apply a recent protocol from Hirt and Maurer [9] for secure multi-party computation to build Alliances of n agents that solve a common task. [9] is very efficient with a communication load of $O(n^2 \cdot m)$ (where m is the number of multiplications). All computations within the Alliance are robust as long as not more than $t_{max} := \lceil \frac{n}{3} \rceil - 1$ agents are corrupted at the same time.

The main contribution of this paper is an analysis of the t_{max}–limit under realistic network assumptions. We use an attack tree to identify possible attacks and discuss countermeasures. We also analyse concrete examples for framework and Alliance sizes, depending on the number of malicious hosts in the framework. In addition, we discuss different possibilities to improve Alliance security and to mitigate Denial-of-Service (DoS) attacks.

1. INTRODUCTION

Today the number of Internet users, hosts and servers is growing rapidly. As a consequence data gathering becomes increasingly difficult and time consuming. Often, manual search through results delivered by search machines like google[1] is needed but remains unsatisfactory as a solution. For this reason the idea of using mobile agents that roam the Internet and act in the user's name has found a considerable number of friends recently. Unfortunately, the security problems arising in connection with such agents are immense and mostly unsolved. During the last decade a lot of research has been done on the protection of single mobile agents. Sander and Tschudin (e.g. [18]) as well as Loureiro and Molva [12] worked on a technique called *function hiding* for confidential computations. To ensure code privacy, Hohl [10] tried *code obfuscation* by creative use of variable identifiers and unstructured implementation. In [21] Wilhelm et al. propose to defend active mobile code (like mobile agents) against malicious hosts by running it exclusively inside trusted hardware. At the moment the idea of trusted hardware is questionable at best. A lot more work has been done, but all approaches cover only parts of the security problems and/or suffer from unrealistic requirements.

Since it is hard to protect a single autonomous mobile agent without a trusted third party, the use of agent 'Alliances' with mutual protection might become the means of choice, even though the communication complexity is significant. In [7] Endsuleit and Mie describe Alliances of n agents whose private data consist of t-shares which require collaboration of at least $t+1$ agents to reconstruct the data. Most computations are done distributedly based on a protocol for secure multi-party computation. To handle concurrent execution of such protocols on the hosting servers the model of Canetti [6] is used. Public computations which are closely connected to the agents' migration are secured by majority decisions.

This paper is an extension of [7]. We will describe an implementable protocol outline of a an improved model in Section 2. In Section 3 we analyse possible concrete attacks using an attack tree method and discuss countermeasures. Section 4 looks at Denial-of-Service (DoS) attacks. Section 5 discusses the question of certificates for hosts that run an agent execution framework. Finally, Section 6 gives practical considerations on the Alliances' size. The paper ends with a brief survey of related work and a conclusion in Sections 7 and 8.

2. HOW ALLIANCES WORK

In the original model of Endsuleit and Mie [7] the n members of an Alliance share their computational state using any protocol for secure multi-party computation that is compliant with Canetti's model [6]. Per definition, such a protocol is intended for n fixed (possibly malicious[2]) players that implement a common functionality like playing a game. The common computation are required to be robust against up to t malicious players. To this end the function to be implemented is translated into a t-robust variant by adding redundancy. For the execution of this new (distributed) function it is necessary to split all data input (i.e. function arguments) into so-called *t-shares*. These are nothing else than redundant data slivers, with the additional property that reconstruction of the original data requires knowledge of at least $t+1$ of the shares. In this paper we do not use

[1] `www.google.com`

[2] A *malicious* player may be active or passive. Malicious players may work together.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright 2002 ACM X-XXXXX-XX-X/XX/XX ...$5.00.

Canetti, because we assume that a server hosting agents of different Alliances at the same time provides sand-boxes for each of them without any shared memory. This enables us to include an arbitrary protocol for secure multi-party computation. We chose Hirt and Maurer [9] because it is likely to be optimal with regard to communication complexity. The cost per round is $O(n^2 \cdot m)$ (m is the number of distributed multiplications) which is linear in the number of participating parties. Since [9] uses the verifiable secret sharing scheme of Ben-Or, Goldwasser and Wigderson [4] and the slightly modified protocol of Beaver [3] for the distributed computations, we get the same condition for the robustness of the distributed computations as in [7], namely that the model can tolerate up to $t_{max} = \lceil \frac{n}{3} \rceil - 1$ malicious players.

With the use of secure multi-party computation in an agent scenario we get the following additional aspects:

- The hosting servers are the participating parties, since they get complete control over the agents executed on them.
- Since the used protocols for distributed communication assume a synchronous network, timeouts have to be used in an asynchronous network like the Internet. Failure to answer within the timeout in the asynchronous model is mapped to failure to answer at all in the synchronous model.
- The agents' transfer from one host to another is completely outside of the protocols for secure multi-party computation. Migration introduces new problems, for example migration target selection and actual agent transfer, that may require computations on clear-text data like location lists. In addition, one must prevent an adversary from accumulating valid shares over the time. Therefore, *mobile* agent Alliances need additional security measures besides secure multi-party computation.

In order to analyse the security issues, we first need to review some important aspects of the Alliance Model:

Structure of an Alliance member

Each agent has 3 different types of data:

1. *Static data* like the code and the originator's name is the same for all agents and secured by a digital signature of the originator who is assumed to be trustworthy.
2. *Dynamic non-secret data* (e.g. a location list) is readable and changeable by the agents without restriction.
3. *Dynamic secret data* is stored in t_{max}-shares created by the verifiable secret sharing scheme from [4]. The data itself is not necessarily dynamic, but the way it is stored changes whenever the shares are re-created. The idea is that data stored in shares is only accessible when at least $t_{max} + 1$ agents collaborate. Possible static data in this class is a secret ID that enables the originator to verify Alliance identity on result return or a private key for digital signatures.

Computations

We differentiate 2 types of computations depending on their data input:

1. *Distributed computations on shares.* They are performed jointly by the Alliance following [9]. These computations are robust against corruption of up to t_{max} Alliance members at any given time.
2. *Local computations of dynamic non-secret data and/or computations using static data.* These computations are executed locally by all agents and are not secure. The whole Alliance guarantees integrity of the results by a final majority decisions. This type of computation is robust but not private and produces data known by each agent.

Life-cycle of an Alliance

The life-cycle of an Alliance is shown in Figure 1. Because of the use of a protocol for secure multi-party computation we differentiate two phases: First, *static phases* in which all agents are hosted on different servers an no migration takes place. A static phase is divided into 3 parts:

1. The *migration post-processing phase* where the agents reconstruct their shares and start operating.
2. The *computation phase* in which all distributed computations for the fulfilment of the Alliance's task are done, secured by the protocol for secure multi-party computation.
3. The *migration preprocessing phase* in which n migration targets are determined and asked for their consent to host an Alliance member. In addition, a re-sharing takes place to prevent an adversary from collecting valid shares.

As indicated above, all computations during the static phase are t_{max}-robust, i.e. remain secure as long as not more than t_{max} participants are malicious. But what happens in the *dynamic phases*, when the agents migrate? All agents are transfered as follows: First, a new location list is determined (migration preprocessing phase). Then, the old hosts execute a re-sharing (this could be done by the method of Ostrovsky and Yung in [14]) to disable the old shares. After the re-sharing process each old host owns shares of the new shares for the new hosts. Next, each old host sends the static data of his agent together with the new location list to each new host. The new hosts then perform a majority decision about the static data. When all potential new hosts agree to the execution of their agent, each old host sends his shares of shares to the appropriate new hosts. If at most t_{max} dishonest messages arrive at a new host, the new agent instance there is able to reconstruct the new shares assigned to it.

Since static data includes the agent's code we have a self-repairing system in which a corrupted agent cannot survive migration. This property helps significantly to prevent the number of malicious Alliance from exceeding t_{max}.

Is the Effort Worth it?

Since all computations are done in a robust way, most of the security problems 'disappear'. In detail, these are

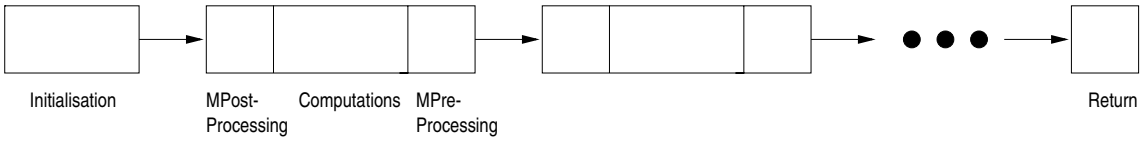

Figure 1: Life-cycle of an Alliance

1. malicious routing,
2. violation of execution integrity,
3. violation of data integrity,
4. long-lasting and accumulating corruption of agents,
5. spying on sensitive data
6. manipulation of the computed results.

What remains is the 'problem' that all those nice properties are closely connected to the upper limit on the number of malicious Alliance members. As soon as it is violated, an adversary can get full control over an Alliance.

3. BAD THINGS TO DO TO AN ALLIANCE

As mentioned in the last Section, Alliance compromise always requires the compromise of at least $t_{max} + 1$ of the agents at any one time. In practical deployment there also exists the possibility of Denial-of-Service (DoS) attacks, with various purposes. DoS attacks can aim at or help with the violation of the t_{max}–limit. They can also be targeted directly at Alliance sabotage. DoS Attacks are discussed in more detail in Section 4.

Attack trees

Attack trees [19] are a specialised version of general decision trees. They are used to organise complex reasoning that starts with abstract concepts and sub-sequentially divides these concepts into more concrete sub-concepts. Our variant of attack trees represents a more abstract attack by the incoming edge of a node and the more concrete sub-cases by the outgoing edges. Hidden assumptions and additional reasoning determining the number and nature of the outgoing edges is attached to the nodes. This additional information is given in textual form. Node-numbers are used as reference. A leaf node indicates that the specific sub-attack it corresponds to is deemed unsuitable for further graphical subdivision and is discussed in textual form instead.

It is important to note that attack trees are not a method for formal reasoning in the mathematical sense. Still, they are a very useful tool to organise informal argumentation, make it more transparent and facilitate identification of special cases. However the analysis itself still relies on human ingenuity and is contained in the text and not in the tree. The tree just binds the argumentation together.

Analysis

Our attack tree for secure mobile agent Alliances can be found in Figure 2. The following list gives hidden assumptions and possible attacks and countermeasures on a node-by-node basis. Many node-comments will contain forward-references to later parts of the paper.

1. Attacks can only happen during the life-cycle of an Alliance. The originator is trusted, therefore the initial migration can be treated like a later migration.
2. The originator is trusted, so no attack is possible.
3. The Alliance is required to follow the life-cycle of Figure 1. The subdivisions are named a little different, but represent the same steps. From a security point of view, the possible vulnerabilities in migration preparation are centred on migration target selection. Other preparation steps are just distributed computations.
4. Basically two attacks are possible in result return: To send fake data to the originator or to impersonate the originator to obtain the result the Alliance is reporting. (If both takes place simultaneously, no harm might be done.)
5. Since the distributed computation method used is believed to be secure, no compromise is possible during the computation phase, as long as there are not more than t_{max} malicious agents in the Alliance. However there is not only one computation phase, but one after each migration. See section 6 for a discussion of the security implications.

   A second concern during the computation phase is Denial of Service. See Section 4 for a detailed discussion.
6. The target selection can either be bad because the computation was compromised or because of random factors. We do not see an other possibility at the moment.
7. Possible Attack here: Migration target impersonation. Whether this is possible depends on the host authentication mechanism used. See Section 5 for a detailed discussion. If enough malicious hosts can impersonate non-malicious hosts, this attack succeeds.

   Migration source impersonation is not possible. It would just create a new Alliance. Message manipulation in transit is not possible, since the transmissions are secured end-to-end. As long as both end hosts are not impersonated and not malicious, no undetected message corruption is possible. If too many hosts are malicious the scheme breaks down in other places anyway.
8. The computations done here are mainly triple generation for the distributed computations and re-sharing. The same discussion as for tree node 5 applies.
9. The originator can be impersonated. However, if the Alliance encrypts the result data in such a way that only the originator can decrypt it, originator impersonation degenerates to conventional DoS and does not compromise confidentiality.

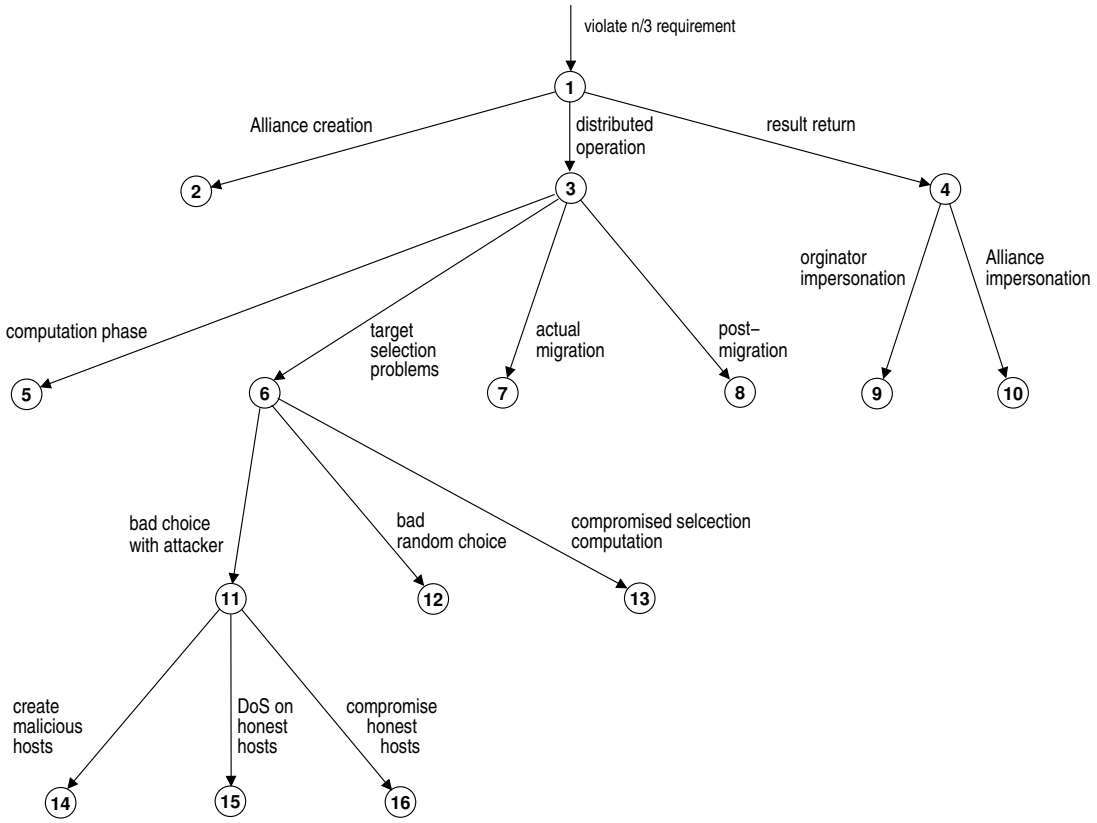

Figure 2: Attack tree

10. Alliance impersonation is feasible. A possible effective countermeasure is to embed a unique, random ID in the shared (secret, dynamic) data of the Alliance. An impersonator cannot get this ID and hence Alliance impersonation fails.

11. The aim is to increase the relative number of malicious hosts in order to make a *bad choice* more likely. By *bad choice* we mean that more than t_{max} malicious migration targets are selected. The possibilities are to have more malicious hosts, less honest hosts (general DoS) or convert honest hosts into malicious ones.

12. Bad random selection of the migration targets is a problem, see Section 6.

13. Since the distributed computation are secure, this attack is not possible.

14. This attack is possible, depending on the host authentication scheme used, i.e. the condition a host has to fulfill in order to be a valid migration target. See Section 5 for a detailed discussion of some options.

15. This type of attack is similar to general DoS attacks. It can succeed, yet it can be made harder, as explained in Section 4.

16. The possibility of host compromise depends on host security and is out of the scope of this paper. Note however that host compromise can lead to a retroactive *bad choice* where a host was honest before and during migration and only became malicious after an agent had been migrated to it or possibly even later when agent data is not deleted securely the old hosts after migration.

We see that there are several types of specific security problems that can cause Alliance compromise. Most are caused by agent and Alliance migration and are discussed in the following sections. In addition, there are some not-quite obvious ways to use DoS attacks as part in an Alliance compromise attempt.

4. DENIAL OF SERVICE

DoS attacks can be used to facilitate Alliance compromise. Of course, DoS can also be used as a direct sabotage-type attack on an Alliance. DoS attacks on one particular Alliance could be used to prevent it from obtaining a specific piece of information, reporting something back to the originator or executing some action. As an application of this type of DoS, think e.g. of a vendor that does not want to honour an offer made to an Alliance earlier.

Generally DoS attacks are very hard to defend against. Still, in many cases something can be done to make them harder or less worthwhile. In the case of mobile Alliances one option is to detect DoS-like conditions and report them back to the originator of the Alliance. Care needs to be taken to distinguish network problems and attacks. A second option is to increase the effort needed for DoS attacks.

Detect and Report

Detection of network problems or insufficient local resources is easy. Timeouts on computations and communication attempts are sufficient in most cases. Resource exhaustion attacks on hosts executing the agents can also be detected in this way. An agent can assume that the environment is too hostile for further operation when the number of accumulated timeouts exceeds a certain threshold. In this case the agent could notify the originator. When the originator gets this type of notification from at least $t_{max} + 1$ honest agents, the originator knows that the Alliance has stopped to be functional. Since the number of malicious agents in an Alliance is not larger than t_{max}, DoS attacks where the agents running on malicious hosts claim an error condition are only possible when Alliance compromise is feasible anyway.

Increase DoS Tolerance

If a DoS attack directly targets individual hosts with agents on them, these hosts are rendered unusable and unreachable in many cases. In the strict model, DoS on even a single honest host allows Alliance compromise, if the maximum permitted number of hosts is already malicious. This is due to the fact that non-responding agents need to be treated as malicious, since they cannot contribute any correct data to computations and majority-votes. One way to deal with this problem is to decrease the number of responding (and thereby not readily identifiable) malicious host that are allowed in an Alliance. If, e.g., the number of allowed responding malicious hosts is decreased to $\frac{n}{6}$, then DoS attacks on up to $\frac{n}{6}$ of the hosts executing an Alliance can be tolerated. See Section 6 for more details.

A connectivity disruption DoS attack can be made more difficult. There are two variants of this attack: Disrupt connectivity between the agents in a computation phase and disrupt connectivity to target hosts in agent migration. In the simple model every agent has to have direct connectivity of every other agent and migration target. An Alliance can employ internal dynamic routing of messages that is hard to understand from the outside. It then forms a *dark-net* [5]. Cutting off an agent's connectivity to the Alliance now requires cutting off its connectivity to all other agents. For the case of migration it requires cutting off the connectivity from all agents to a migration target. Depending on the concrete network infrastructure this may or may not be harder to do than just cutting of some connections.

Alliance-internal routing does not cause additional security problems. Messages are routed over an untrusted network anyway. Adding some possible untrusted members of the alliance in the chain of routers causes no new risks. An added benefit of Alliance-internal routing is that it increases the robustness against unreliable networks.

5. WHAT ABOUT CERTIFICATES?

Agents of an Alliance are executed on hosts that supply a special execution environment with standardised functionality. An important degree of freedom in the design of such frameworks is the question of certificates, i.e. how hosts running an agent framework can demonstrate their identity. Several alternatives exist. Each has a different impact on practicability of the overall system, effort needed and security level achieved.

The main use for host identification is the prevention of an attack where the attacker creates a massive number of (possibly virtual) malicious hosts that run the agent framework. Identity is also beneficial in the creation of secure channels between hosts.

The main possibilities are the following:

1. *Use Certificates.* Each participating host gets a certificate from one of possibly several well known authorities that allows the host to prove its identity and also allows to prevent impersonation attacks when establishing secure connections to other hosts running the framework.

   In addition, this option gives some possibility to detect the 'malicious host creation'-attack in the certifying authorities. Still, if one or more of the certifying authorities are malicious, this scheme breaks completely.

   The most serious drawback of this option is the need for non-distributed, trusted infrastructure. The cost and effort needed to establish and operate this infrastructure may well be prohibitive.

2. *Web-of-trust.* A PGP-like web-of-trust (see e.g. [8]) can be used. This could be done by having certificates that are signed by other hosts running the framework. Whether this will prevent the creation of significant numbers of malicious hosts depends strongly on the concrete details of the scheme used.

3. *Do not use certificates.* This is a least-effort least-security solution. There may be still some weak authentication, namely by IP address, i.e. by reachability. If an attacker cannot intercept most/all packets sent between two specific IP addresses, there is the possibility to establish secure end-to-end communication between two specific IP addresses.

   While this alternative offers little protection against the creation of large numbers of (possibly virtual) malicious hosts, such an attack can be made more difficult. One possibility is to disallow the selection of migration targets with IP addresses that are close together or from the same subnet. In [16] Rennhard and Plattner describe a *collusion detection* mechanism along these lines. The reason this is effective to some degree is that physical subnets on the Internet can only be allocated in larger portions. The underlying reason is that routing in the Internet is not done on individual addresses but on target address prefixes which correspond to subnets of groups of subnets that are physically close together.

   With the IP address based countermeasures an attacker can not simply take a class B subnet [15] and install 65,533 malicious hosts[3]. Instead, a large number of smaller subnets has to be obtained or addresses in many subnets have to be connected to malicious hosts. This is very expensive, since either hosts physically connected to the individual subnets are needed or virtual channels have to be created from the individual IP addresses to a central pool of (virtual) malicious hosts.

[3]At least two addresses are needed for network purposes and cannot be used as host addresses.

Still, attacks that create large numbers of malicious hosts are feasible if the attacker does not care about legality or about doing a huge amount of damage. A worm could be used to compromise a large number of well distributed hosts that could then serve as malicious hosts within an agent framework. 100,000 and more compromised hosts in worm attacks are feasible today.

Whether this type of attack is visible enough to be detected remains to be seen. The first attempts at 'stealth worms' that do not significantly impact host functionality or performance, and thereby limit the motivation of host operators to deal with them, have already been observed.

6. NUMBERS MATTER

In the absence of agent migration a secure distributed computation needs only be concerned with the number of malicious hosts in the set of hosts carrying out the computation. A limit on the number of malicious hosts is sufficient to model the requirement for a secure computation.

With agent migration the problem becomes more difficult. If there are enough malicious hosts in the set of migration target hosts to choose from, a *bad choice* can happen that selects more malicious targets than the scheme for secure distributed computation can tolerate. The question of Alliance compromise by malicious hosts becomes a matter of the target selection process and if it is random or has random components, a question of probability.

Bad Choice

We now analyse the chances of alliance compromise by bad random selection of migration targets. The problem exists in a scenario where the number of malicious hosts is constant ('attacker-less bad choice') as well as in the cases where the relative number of malicious hosts has been increased by an attacker ('bad-choice with attacker'). For the analysis there is no real difference. It is just important to keep in mind that the number of malicious hosts does not need to be stable during an Alliance's lifetime.

The following analysis assumes a static number of malicious hosts. In a scenario with dynamically changing malicious host numbers the following limits can still be used as upper bounds. Special scenarios might need more specialised models.

To solve the problem for random selection, the overall fraction of malicious hosts needs to be lower than the fraction of malicious hosts permitted in the Alliance. In addition the number of migrations m has to be limited, since the risk of selecting too many malicious hosts exists in each migration.

Let now k_m be the number of malicious hosts in the set of k participating hosts (i.e. hosts that operate an agent framework) and n the size of the Alliances as before. Then, the probability of sending exactly i agents to malicious hosts and $n-i$ agents to honest hosts is (for $i \leq n, k-k_m \geq n$):

$$P(i \text{ mal. hosts selected}) = \frac{\binom{k_m}{i} \cdot \binom{k-k_m}{n-i}}{\binom{k}{n}}$$

With the selected protocols the number of malicious hosts needs to be smaller than $t_{max} = \lceil \frac{n}{3} \rceil - 1$. Hence the probability of having selected up to the maximum number of allowed malicious hosts is now:

$$P(\text{max. } t_{max} \text{ mal. hosts selected}) = \frac{\sum_{i=0}^{t_{max}} \binom{k_m}{i} \cdot \binom{k-k_m}{n-i}}{\binom{k}{n}}$$

For m migrations we get the probability P_t of having not more than t_{max} corrupted agents in the Alliance after each migration step as follows:

$$P_t = \left(\frac{\sum_{i=0}^{t_{max}} \binom{k_m}{i} \cdot \binom{k-k_m}{n-i}}{\binom{k}{n}} \right)^m$$

This means P_t is the overall probability that the alliance is not compromised because of bad host choices in migration. In each migration the agents are securely re-created and consequentially corrupted agents can not move from one host to the other, hence the argument is valid. For the case $k_m \leq t_{max}$ we get $P_t = 1$ because there are not enough malicious hosts available to break the system.

Bad Choice by Example

Bad choice is a real problem, as will be illustrated now with some concrete examples. In the following figures we used Maple [11] to plot P_t for $k = 100$ and $k = 10,000$ participating hosts and Alliances of sizes of $n = 10$ (Figures 3, 5) and $n = 50$ (Figures 4, 6). The range of the number of malicious hosts k_m was limited to $\frac{k}{2}$, since for more malicious hosts, P_t is always very close to 0 in our examples.

Choosing $k = 100$ and $n = 10$ we get $P_t \approx 0.921$ for having a non-compromised Alliance after 10 migrations if up to 10 hosts (10%) are malicious. For $n = 50$ we improve the result to $P_t \approx 0.996$ for $k_m \leq 20$ (20% malicious hosts). Even with 25% malicious hosts we still have $P_t \approx 0.7245$.

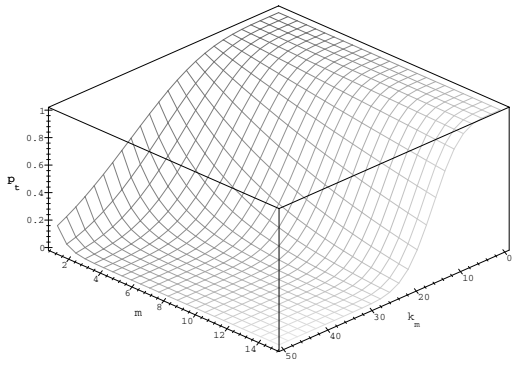

Figure 3: $k = 100$ **and** $n = 10$

As one can see in Figures 5 and 6 the results for $k = 10,000$, are a bit worse. For instance in case of $n = 10$, $k_m = 1000$ and 10 migrations we get $P_t \approx 0.8796$. And, as expected for $n = 50$ and $k_m = 2000$ we have $P_t \approx 0.8665$. If we reduce k_m to 15% we get $P_t \approx 0.9936$.

The following tables 1,2 and 3 demonstrate the necessity of choosing n big enough. Since Maple has precision issues when the values approach 0 or 1, the tables where calculated using the GNU MP [1] library with a floating point precision of 1024 bits. Entries are rounded in the last digit.

As a general observation, larger Alliances are more secure, which is not surprising. It can also be seen that pretty

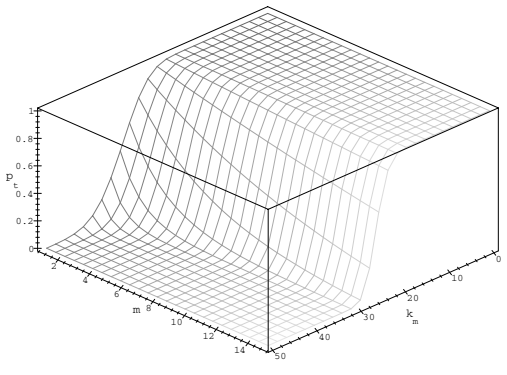

Figure 4: $k = 100$ **and** $n = 50$

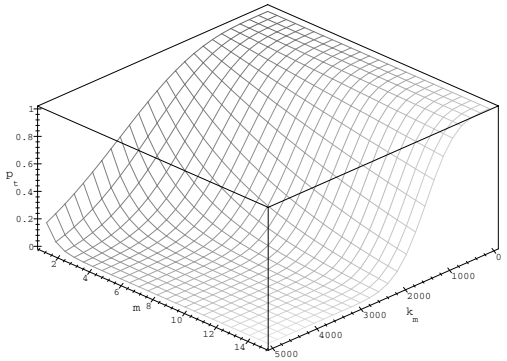

Figure 5: $k = 10,000$ **and** $n = 10$

large numbers of malicious hosts can be tolerated. These results demonstrate the practical feasibility of the Alliance approach.

7. RELATED WORK

Recently, some publications considered agent groups. In [22] Yee uses redundant groups whose members visit the same servers but not at the same time. The groups of Minsky et al. [13] visit servers at the same time with the same resources. Both use majority decisions as security mechanism. Roth [17] utilises a group of two co-operating agents with redundantly distributed data. Each of them informs

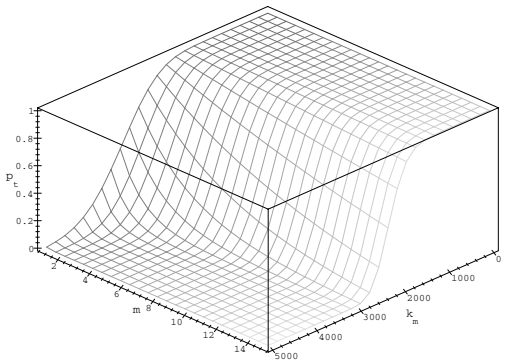

Figure 6: $k = 10,000$ **and** $n = 50$

$k = 100$			
$n = 10 \quad (t_{max} = 3)$			
$m \backslash k_m$	10	20	30
10	0.9207	0.3133	0.0143
30	0.7805	0.0308	≈ 0
100	0.4378	≈ 0	≈ 0
$n = 30 \quad (t_{max} = 9)$			
10	$1-2{\cdot}10^{-5}$	0.7322	0.006
30	$1-5{\cdot}10^{-5}$	0.3925	≈ 0
100	0.9998	0.0443	≈ 0

Table 1: $k = 100$

$k = 1,000$				
$m \backslash k_m$	10	30	100	300
$n = 10 \quad (t_{max} = 3)$				
10	$1-1{\cdot}10^{-5}$	0.9988	0.8833	0.0135
30	$1-3{\cdot}10^{-5}$	0.9963	0.6893	≈ 0
100	$1-1{\cdot}10^{-4}$	0.9878	0.2893	≈ 0
$n = 30 \quad (t_{max} = 9)$				
10	$1-1{\cdot}10^{-15}$	$1-2{\cdot}10^{-8}$	0.9965	0.005
30	$1-3{\cdot}10^{-15}$	$1-7{\cdot}10^{-8}$	0.9895	≈ 0
100	$1-1{\cdot}10^{-14}$	$1-2{\cdot}10^{-7}$	0.9655	≈ 0
$n = 100 \quad (t_{max} = 33)$				
10	≈ 1	≈ 1	$1-2{\cdot}10^{-11}$	0.0957
30	≈ 1	≈ 1	$1-5{\cdot}10^{-11}$	0.0009
100	≈ 1	≈ 1	$1-2{\cdot}10^{-10}$	≈ 0

Table 2: $k = 1,000$

the originator and terminates if there are data discrepancies. Finally, Tate and Xu [20] propose agents that own a secret key which is distributed to the agents by a dealer using a secret sharing scheme. The shares are signed by the (trusted) dealer. Computations are based on Algesheimer et al. [2] using encrypted circuits.

Although the use of cryptographic protocols for multi-party computation as a possible solution to the security problems in mobile multi-agent systems has been suggested repeatedly (e.g. [18, 22, 2]), Endsuleit and Mie [7] are the first pursuing this idea in depth. A possible reason is the absence of an efficient secure multi-party protocol until 2001.

8. CONCLUSION

We have done a detailed security analysis of mobile agent Alliances. We have then shown how such Alliances can be secured and which level of security can be expected for different parameter choices. Our results imply that secure mobile agent Alliances are feasible and secure enough to be used in practice.

What remains to be seen is the actual communication effort needed for computations and migrations in real deployment. While the amount of data send over the network

$k = 10,000$			
$m \backslash k_m$	30	100	300
$n = 10$ $(t_{max} = 3)$			
10	$1-1{\cdot}10^{-7}$	$1-2{\cdot}10^{-5}$	0.9986
30	$1-4{\cdot}10^{-7}$	$1-6{\cdot}10^{-5}$	0.9957
100	$1-1{\cdot}10^{-6}$	0.9998	0.9857
$n = 30$ $(t_{max} = 9)$			
10	$1-3{\cdot}10^{-18}$	$1-2{\cdot}10^{-12}$	$1-9{\cdot}10^{-8}$
30	$1-1{\cdot}10^{-17}$	$1-5{\cdot}10^{-12}$	$1-3{\cdot}10^{-7}$
100	$1-3{\cdot}10^{-17}$	$1-2{\cdot}10^{-11}$	$1-9{\cdot}10^{-7}$

Table 3: $k = 10,000$

can be determined analytically, practically important parameters like computation latency and time needed for a full migration will likely have to be determined using simulation and test bed deployment, since they depend strongly on the characteristics of the underlying network infrastructure. Test bed deployment can also help to determine real world resource consumption on hosts that execute the individual agents.

9. REFERENCES

[1] GNU MP - Library for arithmetic on arbitrary precision numbers. `http://www.gnu.org/directory/libs/gnump.html`.

[2] Joy Algesheimer, Christian Cachin, Jan Camenisch, and Günter Karjoth. Cryptographic security for mobile code. In *Proc. IEEE Symposium on Security and Privacy*, pages 2–11, 2001.

[3] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Advances of Cryptology – Crypto '91*, volume 576 of *LNCS*, pages 420–432. Springer Verlag, 1991.

[4] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proc. 20th ACM Symposium on the Theory of Computing (STOC)*, pages 1–10, 1988.

[5] Peter Biddle, Paul England, Marcus Peinado, and Bryan Willman. The darknet and the future of content distribution. In *Proceedings of the 2002 ACM Workshop on Digital Rights Management*, 2002.

[6] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067.

[7] Regine Endsuleit and Thilo Mie. Secure multi-agent computations. In *Proc. of Int. Conference on Security and Management*, volume 1, pages 149–155. CSREA, 2003.

[8] Patrick Feisthammel. Explanation of the web of trust of pgp. `http://www.rubin.ch/pgp/weboftrust.en.html`, 2002.

[9] Martin Hirt and Ueli Maurer. Robustness for free in unconditional multi-party computation. In *Proc. of Advances in Cryptology - CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 101–118. Springer, 2001.

[10] Fritz Hohl. An approach to solve the problem of malicious hosts. Technical Report 1997/03, Universität Stuttgart, 1997. Fakultätsbericht.

[11] Waterloo Maple Inc. Maple V, Release 4.

[12] Sergio Loureiro and Refik Molva. Function hiding based on error correcting codes. In *Proc. of Cryptec '99 — International Workshop on Cryptographic Techniques and Electronic Commerce*, pages 92–98, 1999.

[13] Yaron Minsky, Robbert van Renesse, Fred Schneider, and Scott Stoller. Cryptographic support for fault-tolerant distributed computing. In *Proc. of the 7th ACM SIGOPS European Workshop*, pages 109–114, 1996.

[14] Rafail Ostrovsky and Moti Yung. How to withstand mobile virus attacks. In *Proc. of the 10th Ann. ACM Symp. on Principles of Distributed Computing*, pages 51–59, 1991.

[15] Jon Postel. Rfc790. `http://www.ietf.org/rfc/rfc0790.txt`, September 1981.

[16] Marc Rennhard and Bernhard Plattner. Introducing MorphMix: Peer-to-Peer based Anonymous Internet Usage with Collusion Detection. In *Proc. of the Workshop on Privacy in the Electronic Society (in association with 9th ACM Conference on Computer and Communications Security)*, pages 91–102, 2002.

[17] Volker Roth. Mutual protection of co-operating agents. In J. Vitek and Ch. Jensen, editors, *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, volume 1603 of *Lecture Notes in Computer Science*, pages 275–285. Springer Verlag, 1999.

[18] Tomas Sander and Christian F. Tschudin. Protecting mobile agents against malicious hosts. In G. Vigna, editor, *Mobile Agents and Security*, volume 1419 of *Lecture Notes in Computer Science*, pages 44–60. Springer Verlag, 1998.

[19] Bruce Schneier. Attack trees. Dr. Dobb's Journal, December 1999.

[20] Stephen R. Tate and Ke Xu. Mobile agent security through multi-agent cryptographic protocols. In *Proc. of 4th Intern. Conf. on Internet Computing*, pages 462–468, 2003.

[21] Uwe G. Wilhelm, Sebastian Staamann, and Levente Buttyán. Introducing trusted third parties to the mobile agent paradigm. In J. Vitek and C.D. Jensen, editors, *Secure Internet Programming*, volume 1603 of *Lecture Notes in Computer Science*, pages 469–489. Springer Verlag, 1999.

[22] Bennet S. Yee. A sanctuary for mobile agents. In *Secure Internet Programming*, pages 261–273, 1999.