

Agent Alliances: A Means for Practical Threshold Signature

Regine Endsuleit and Christoph Amma
Universität Karlsruhe (TH)
IAKS
Germany

E-mail: endsuleit@ira.uka.de

Abstract

In [7] we have proposed a model for the robust and private evaluation of a function within a group of cooperating mobile agents, called an agent Alliance. The model has been given abstractly and is based on a cryptographic protocol for secure multi-party computation. This paper aims on the investigation of the complexity of a threshold signature performed within the Alliance model. The goal is to achieve a practical security model in which agents can be allowed to act fully autonomously in their users name without relying on a trusted authority.

1 Introduction

Over the last decade, the concept of mobile agents has been given more and more attention from the side of research as well as from industry. Yet, scientific publications, commercial implementations and the users wishes dramatically diverge. What is the reason for this gap? Our crucial experience happened during an important multi-agent conference, taking place a few years ago, where, at a panel discussion, the participating researchers came to the conclusion that this whole research area was not more than a theoretical playground. The unanimous opinion was, that no-one ever would be as “insane” to use mobile agents for any sensitive application. In an unknown and untrusted environment like the Internet the threats to an agent are too severe as to allow an agent to act autonomously in its users name. The problem is mainly caused by the necessity of sending executable code to a host. Thus, the host has full control over the agent; it may read, analyse and change its code as well as collected (private) data. Besides, execution integrity cannot be granted nor is it possible to guarantee that the agent

can follow its itinerary (regardless of being predefined or computed in real-time). At that time we fully agreed with this opinion. This changed when we got confronted with cryptographic protocols for secure multi-party computation (SMPC). Although these protocols can be generally considered as non-practical it was suddenly possible to design a model for secure multi-agent computation [7] in which a user-defined functionality can be evaluated in the presence of malicious hosts. It was possible to define strong guarantees for a robust and private computation as long as at no point in time more than t (usually one third) of the current hosts are corrupted. First, this idea of taking a look on secure multi-party computation as means for the security of agents has been mentioned in [16] in a concluding remark. But no-one ever pursued this idea. Most probably, the reason for this lies in the high communication and computation complexity of those protocols. In addition, SMPC assume a kind of static model, in which not only the number n of parties but also the identities of the parties are fixed. In a mobile agent scenario this presumption cannot be held. In [7] we took these aspects into account and handled them with suitable methods like resharing techniques. Anyhow, the complexity of the resulting model was too high to consider a practical implementation. But, it was the first approach proving that by co-operation and distributed computations of agents it is possible to assign sensitive tasks to agents without the need of a trusted third party or regular communication with the originator. Thus, it was an important result. While following this line of research the protocol of Hirt and Maurer from [12] found our attention. It was and still is the first protocol for SMPC that can be considered practical causing a computation complexity of $O(m \cdot n^2)$ ¹. Moreover, it offers unconditional security. As most often in life,

¹With n being the number of parties and m the number of multiplications the function requires.

there were profound disadvantages connected to the overwhelming complexity gain. The most important one was the presumption of a synchronous network. We have discussed the problems arising when using a synchronous protocol in an asynchronous network in [8]. There, also a risk analysis for the model in real networks is given.

Still, it was a theoretical model, just saying that any function that could be computed by a Boolean circuit can also be realised by secure multi-agent computation (SMAC). Since the aim of the research was to give exemplary applications, we started an implementation as well as the investigation of specific functions. It turned out very quickly, that the communication complexity of secure multi-agent computations is still too high as to use it for each and every function. Therefore, we took a look at some applications that are extremely risky for mobile agents to compute. The most important example is a digital signature. Since the user would like to have an agent that can act in his name it must be able to sign contracts. Thus, it must be provided with a private key. As mentioned before, a host is able to read the data an agent carries. This also enables it to read a private key. Encryption does not solve the problem for two reasons:

1. Either the key must somehow be decrypted before being used, which implies on the one hand another key that must be hidden and on the other hand the key is in clear-text when used.
2. In case the key stays encrypted and the encryption function allows to perform a signature with an encrypted key (see e.g. [14]), the host could just use the encrypted key together with the signature method from the agent in order to sign other contracts.

It seems as if the research community came to the conclusion that a digital signature can not be performed securely by single agents. Instead, there are several approaches for distributed signatures (some of them robust, others not). The non-robust ones are quite efficient, using the homomorphic properties of RSA while the robust ones are connected with a high communication complexity.

In this paper it is shown, that while SMAC are much more powerful than the pure threshold signature schemes, a digital signature can be performed at least as efficient as those schemes. To achieve this efficiency, a mathematical trick is used that is not envisaged by the protocols for SMPC. The resulting threshold signature scheme offers cryptographic security.

The remainder of this paper is structured as follows: First, the protocol for secure multi-agent computation

from [7, 8] is reviewed in some depth in order to provide the reader with the basics ideas of this model. Section 5 deals with considerations on the simulation of control structures by arithmetic circuits which lead to a practical implementation of the protocol. Thereafter, it is discussed how to efficiently implement a digital threshold signature within the SMAC model. A detailed complexity analysis is given. The paper ends with some concluding remarks including a short discussion on the complexity of other distributed signature schemes.

2 Secure Multi-Agent Computations

The protocol for secure multi-agent computation in [8] is based on the protocol for secure multi-party computation of Hirt and Maurer [12]. The latter allows the efficient and robust evaluation of a function within a group of n untrusted and possibly malicious parties. In the SMAC model it is used for computations during static phases, in which all Alliance members are executed on different hosts and no migration takes place. Migration itself cannot be covered by SMPC as the number of hosts temporarily diverges from n . Other means must be taken. This is discussed in depth in section 3. First, the protocol of Hirt and Maurer is shortly reviewed in order to impart a better understanding for the Alliance model presented in section 3.

2.1 Secure Multi-Party Computation

The protocol for secure multi-party computation (SMPC) from [12] presumes synchronous networks and secure channels and offers unconditional security. The protocol execution is robust against an unbounded and active adversary that may corrupt up to $t = \lfloor \frac{n-1}{3} \rfloor$ parties. The function that is to be evaluated is given in the form of an arithmetic circuit over a finite field \mathbb{F} known to all parties. Thus, the possible operations are limited to addition and multiplication and scalar multiplication. This does not represent a restriction for the class of functions that can be evaluated as shown in [11]. Semi-bounded arithmetic circuits of polynomial depth can simulate any semi-bounded Boolean circuit of polynomial depth. In practice all gates have finite fan-in which implies that each function that can be evaluated by a Boolean circuit can also be evaluated by an arithmetic one.

The input of the function is t -shared² among the parties using the verifiable secret-sharing scheme (VSS)

² t -shared means to use a (t, n) secret-sharing scheme which allows any set of $t+1$ parties to reconstruct the secret and which does not leak any information to any subset of up to t parties.

from [2]. This scheme is based on Shamir’s well-known secret-sharing scheme [15] and allows to check the consistency of shares and a secret reconstruction as long as not more than t shares are inconsistent or missing. Since those shares represent the input of the circuit, one has to look at the local computability of the allowed operations. In case of addition it is obvious that the sum of two t -shares $q(\alpha_i)$, $q(\beta_i)$ of party p_i is a t -share of $\alpha + \beta$. Analogously, a publicly known scalar u can be multiplied to a secret α by computing $u \cdot p(\alpha_i)$ at any Alliance member. Both operations can therefore be done locally by each party without communication with other parties. Multiplication of two shared secrets x, y requires one shared random triple (a, b, c) with $c = a \cdot b$ for each multiplication. The t -shared result is then computed as follows

$$\begin{aligned} x \cdot y &= ((x - a) + a) \cdot ((y - b) + b) \\ &= (x - a)(y - b) + (x - a)b + (y - b)a + c \\ &= d_x d_y + d_x b + d_y a + c \end{aligned}$$

The differences d_x and d_y are publicly reconstructed and do not leak information on x, y or $x \cdot y$ since a, b, x, y are not publicly known. Thus a multiplication requires $O(n^2)$, or more precisely, $2n^2$ messages to be communicated.

The protocol is divided into two phases: The preparation phase in which m random triples are generated (one for each multiplication) and the actual computation phase. In the beginning of the computation phase each party shares its input with the other parties by using the slightly modified verifiable secret-sharing scheme from [2]. Thereafter, each party evaluates the circuit gate by gate where for each multiplication one of the random triples from the preparation phase is spent.

The communication complexity of the protocol is in $O(m \cdot n^2)$, where m is the number of multiplication gates and n the number of parties. Hirt and Maurer state that this result may be optimal since it seems unavoidable to cause less costs than those proportional to a broadcast for each multiplication gate. Thus it is the first and, till now, the only protocol for secure multi-party computation that is practical.

3 Agent Alliances

While it is quite obvious that an SMPC protocol can be used for agents which do not migrate, the situation must be carefully analysed in case of *mobile* agents. The life-cycle of a mobile agent is illustrated in figure 1. The single phases consist of static and dynamic parts. Each phase, in which an agent is statically located on one host is split up into three sub-phases:

1. The *migration post-processing phase* where the agents and their shares are reconstructed and start operating.
2. The *computation phase* in which all distributed computations for the fulfilment of the Alliance’s task are performed, secured by the protocol for secure multi-party computation.
3. The *migration pre-processing phase* in which n migration targets are determined and asked for their consent to host the Alliance. In addition, new shares of the Alliance’s knowledge are computed and the old ones disabled. This guarantees that the former hosts have no relevant information on the Alliance, anymore.

Migration post-processing and migration pre-processing phases are dynamic phases; computation phases are static ones. While all computations performed within static phases are secure, dynamic phases require special handling. There are new threats to be considered.

- Once an agent is corrupted and no additional means are taken we have to assume a corrupted agent to stay in this condition for the rest of its life. In a naive approach, corrupted agents migrate to other hosts and cause their infection. Thus, the number of corrupted hosts in the network as well as the number of corrupted agents increases over the time. Consequently, the probability that an honest agent migrates to a malicious host and the upper bound t is exceeded, increases over the time.
- As it is possible that a server hosts several Alliance members over the time, the adversary must be modelled as a mobile one that could gather information about more than t shares although it controls at most t hosts at any point of time. In this case, the security of the SMAC protocol would be broken although the presumptions of the SMPC protocol are kept. The model for secure multi-agent computation is built upon secure multi-party computation and should be secure under the same presumptions. Therefore, there must be adequate tools like a regular re-sharing to prevent from this contradiction.
- The migration process cannot be realised within the SMPC protocol as it cannot handle the secure agent transfer from the old hosts to the new ones. Number as well as identity of the parties are changing which must be covered by a separate sub-protocol.

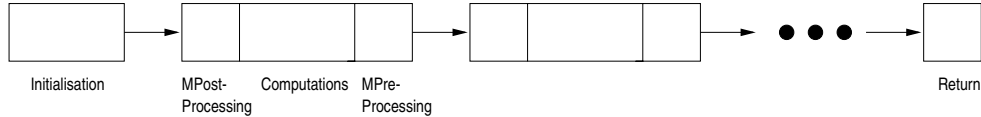


Figure 1. Life cycle of an Alliance

The security flaws arising with dynamic phases are solved mainly by two means: majority decisions within the Alliance and re-sharing techniques [13]. For more detail and the actual protocol, the reader be referred to [7]. It is only exemplarily reviewed in the next section.

4 Alliance Example

Assuming an environment in which never more than $t = 1$ Alliance member is corrupted at any point in time, the size of an Alliance should be at least $3t + 1 = 4$ and the agents data must be 1-shared. There is a subset of at most 2 agents that contains pairs of agents accusing each other for corruption and since one cannot determine who tells the truth both are excluded from computations. The remaining set of 2 honest agents is co-operating and able to perform the computations correctly. The construction of an Alliance of size $n = 4$ is illustrated in Figure 2. Due to the user interface the user still defines one (virtual) agent that is capable of fulfilling a special task. Internally, a protocol compiler transforms the agent’s program into one that is capable to perform distributed computations on shares. Additionally, all data is split into four 1-shares. Finally, an Alliance of four agents is generated and launched.

Figure 3 continues the example of an Alliance of size 4 which has been sent to make a common computation on 4 fixed nodes. One of them, node 4, is malicious and fakes his agents shares. Since $t = \lceil n/3 \rceil - 1 = 1$ the computations of the Alliance are robust against one adversary. Consequently, the originator is able to re-construct a correct result upon result return.

5 Simulation of Control Structures

Aiming on an implementation the problem of simulating programs with arithmetic circuits arises. This requires a specific machine model and is part of the field of compiler construction and optimisation. Nonetheless, to be able to demonstrate the practicability of SMAC, solutions to some basic problems are given. We do not claim these solutions to be optimal, but for a first approach they are sufficient.

5.1 Conditional Branches

There is no possibility to implement a conditional branch in an arithmetic circuit as in any evaluation process all gates are evaluated. Each gate gets input and produces output. This cannot be suppressed. In normal programming languages there are control structures which use control variables. Thus, by analysing the control flow of a program, information, e.g. about the control variables, gets lost. This must be prevented.

As an example, take a look at the following simple example:

`if b then $x \leftarrow A_1$`

By analysing the control flow one can determine whether the statement $x \leftarrow A_1$ has been executed or not. Thus, the condition b is revealed. In case of a modular exponentiation using the square-and-multiply algorithm the consequences can be crucial. Computing an RSA digital signature as presented in section 6, a binary representation of the private key serves as exponent for a modular exponentiation. Each bit of the private key serves as condition for an `if-then-else` statement. Thus, by an analysis of the control flow, the whole private key is revealed. To prevent from this, it is necessary to *complete* conditional statements, which means that missing branches are anyhow created.

But how can this be done? A completion is reached by statements that do not change the state of the variables. In the example above, the solution is to introduce an `else` branch with the command $x \leftarrow x$. Thinking of a normal program, this still allows a side-channel attack as the computation of a regular command will take much more time as the assignment $x \leftarrow x$. A modern compiler will even recognise and delete such meaningless statements. However, simulating the branching and having distributed variables the situation is different as one can see in example 1.

Example 1 *The conditional statement*

`if b then $x \leftarrow A_1$`

is computed by

$$x = b \cdot A_1 + (1 - b) \cdot x.$$

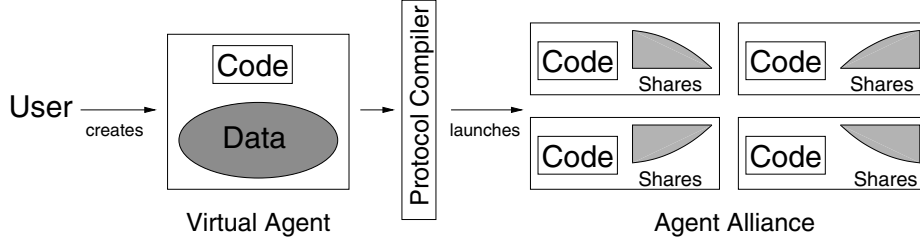


Figure 2. From agents to agent Alliances

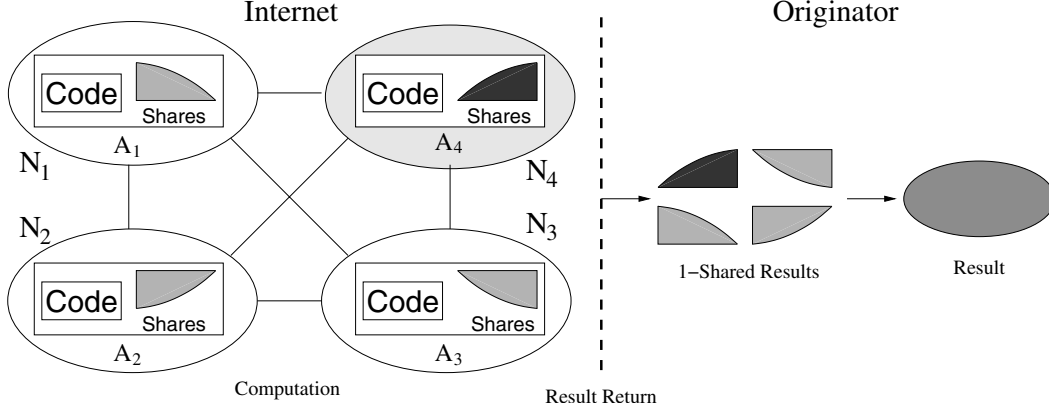


Figure 3. Result return with one malicious node

As x and b are distributed, their shares Mx, Mb are field elements. Thus, the product $(1 - Mb) \cdot Mx$ that is to be locally computed in the simulation of the `else` branch, is not trivial and can be expected to cost as much time as the `if` branch.

In general, the simulation of a conditional branch is done as in definition 1.

Definition 1 Let b be a Boolean value and A_1, A_2 arithmetic instructions. Then, the branching

`if b then $x \leftarrow A_1$ else $x \leftarrow A_2$`

can be locally computed as follows:

$$Mx = Mb \cdot MA_1 + (1 - Mb) \cdot MA_2$$

The computations costs 2 distributed multiplications.

5.2 Loops

A loop consists of a block of code and a condition. Again, the latter does not exist in an arithmetic circuit. Therefore, the simulation requires a trick. The while loop can be thought of as a repeating `if` statement without an `else` branch. If one knows an upper bound k of the number of executions a simulation of

```
while B do
  x ← A1
end
```

is given by the sequence

$$\begin{aligned} Mx &= Mb_1MA_1 + (1 - Mb_1)Mx, \\ Mx &= Mb_2MA_2 + (1 - Mb_2)Mx, \\ &\vdots \\ Mx &= Mb_kMA_k + (1 - Mb_k)Mx. \end{aligned}$$

The condition b_i contains the result of the evaluation of B in the i th iteration. An arithmetic representation of the result x is

$$x = b_k A_k + (1 - b_k)(b_{k-1} A_{k-1} + (1 - b_{k-1})(\dots \cdot (b_2 A_2 + (1 - b_2)(b_1 A_1 + (1 - b_1) A_1)) \dots))$$

Obviously, the number of required distributed multiplications is

$$\mathcal{M}(\text{while}_k) \leq 2k + k \cdot \mathcal{M}(B) + \sum_{i=1}^n \mathcal{M}(A_i),$$

where $\mathcal{M}(B)$, $\mathcal{M}(A_i)$ defines the number of distributed multiplications necessary to evaluate the arithmetic expressions b and A_i .

6 RSA

In this section, we analyse the accomplishment, security and exact communication complexity of a digital signature using SMAC. The section starts with a detailed description of the algorithm that is used as well as possible optimisations. The second part of this section consists of a mathematical discussion about the security risks arising when an arithmetic circuit over a finite ring is used instead of a field. Finally, the actual complexity analysis and an overview on the time necessary for one signature presuming Internet connections that are usual today is given.

6.1 Modular Exponentiation with Square-and-Multiply

An RSA signature requires the computation of $h^d \bmod n$ for a given message h . Besides its efficiency³ the so-called *Square-and-Multiply Algorithm* (S&M) is a kind of natural solution to the exponentiation problem using SMAC. We will see this after a short review of the algorithm. S&M uses the unique representation

$$h^d \bmod n = h^{1 \cdot d_0} \cdot h^{2 \cdot d_1} \cdot h^{4 \cdot d_2} \cdot \dots \cdot h^{2^k \cdot d_k} \bmod n$$

of h^d with $d = \sum_{i=0}^n d_i 2^i$, $d_i \in \{0, 1\}$:

Input: $n \in \mathbb{N}, d, m \in \mathbb{Z}_n$

Output: $m^d \bmod n$

```

res := 1 for i ← k to 0 do
  res = res2 mod n if di = 1 then
    res = (res * m) mod n
end

```

end

Algorithm 1: Modular Exponentiation with Square-and-Multiply

As discussed in section 5 an implementation in traditional programming languages could open the possibility of side-channel attacks by analysing the control flow of the program. Especially in case a private key is used as conditional argument for a control structure, important information could get lost. Here, we have two problems:

1. By using **if** depending on the key bits, the whole key is revealed.
2. Because of the **for** loop the number of bits of the private key d is revealed.

As explained in section 5.1 we solve the first problem by simulating both branches of the conditional statement using the following construction: Be res_i the result after the i th loop. Then, res_{i+1} is computed by

$$res_{i+1} := res_i^2(d_{i+1} \cdot h(m) + (1 - d_{i+1})).$$

The multiplication of d_{i+1} and $h(m)$ is a local scalar multiplication as the hash value is publicly known. The resulting product is a shared value. Thus, there are two shared multiplications, namely the computation of res_i^2 and multiplying the result with the result of the shared scalar multiplication. The resulting number of multiplication steps is in $\Theta(2k) = \Theta(2 \log_2 d)$. As the d_i are binary, they can be directly used as conditional statements for the computation of both branches. No further evaluation of arithmetic expressions is necessary. This is why S&M offers an efficient solution to the exponentiation problem in the SMAC model.

Additional measures are necessary in order to hide the key's actual size. For smaller d a padding is difficult as S&M would deliver wrong results. It is much easier to demand the private key to be of a specific binary length as to say 1024 bit. Doing so, an attacker knows the exact length of the key and thus the highest valued bit to be 1. Consequently, the key space an attacker had to search through is reduced to $2^{1024} - 1$ possible keys. This is approx. one half of the original search space. For practice this is not relevant, but the problem could easily be circumvented by enlarging the key to 1025 bit. A last point to discuss in this context is the creation of the private key. Is there any problem to find an invertible number of exactly k bits in a given residue class ring \mathbb{Z}_n with $n = pq$, with p, q prime numbers, and is there a serious reduction of the key space? The answer is "no" as the multiplicative group \mathbb{Z}_n^\times has $(p-1)(q-1) = pq - p - q + 1$ elements. This means, the number of zero divisors is only $p + q - 1$. As all zero divisors are multiples of either p or q , the invertible elements are profoundly uniformly distributed over the whole ring. Consequently, there are as many k -bit invertible numbers in \mathbb{Z}_n as smaller ones, and only one bit of the search space gets lost.

6.2 SMAC Digital Signature over the Ring \mathbb{Z}_n

SMPC from [12] is defined over an arithmetic circuit over the finite field \mathbb{F}_p . We have investigated in [1] the

³The number of multiplications needed to raise a number h to the power of d lies in $O(\log d)$.

implementation of a long number arithmetic over \mathbb{F}_p and the resulting complexity of a SMAC signature. As expected, the result was depressing and forbids practical usage. But, how to avoid long numbers if one wants to apply public key cryptography? An intuitive solution would be to abandon the field property and to work in the residue class ring \mathbb{Z}_n , instead. This way, the modulo operations in the S&M-algorithm are automatically done by the circuit gates and do not require communication. In general, substituting a field with a ring is problematic as there are zero divisors in a ring that do not exist in a field. Therefore, one has to analyse the SMAC protocol as well as the application carefully.

There are two problem cases requiring the inversion of numbers.

1. As shown in section 2.1 each multiplication requires a random triple. The triple generation phase in the beginning of the protocol from [12] uses Lagrange polynomials:

$$L_i(X) = \prod_{j=0, j \neq i}^t \frac{(X - x_j)}{(x_i - x_j)}$$

where x_i is the value that is uniquely assigned to agent A_i .

Those values x_1, \dots, x_n are assigned only once and can be arbitrarily chosen before the protocol starts. Thus, it is no problem to check whether the differences $(x_i - x_j)$ for $i \neq j$ are invertible in \mathbb{Z}_n . There is no limitation for the security parameters by doing so.

2. Secret reconstruction uses the method from Berlekamp and Welch [3]. This entails the necessity of solving a linear equation system using the algorithm of Gauss. Here, one cannot eliminate the risk of having to invert a zero divisor a . In this case, the protocol would terminate with an error and by computing $\gcd(a, n)$ either p or q would be revealed. But, there is one argument saying that this risk can be taken:

Breaking RSA is equivalent to factorising the modulus n . Still, the problem of factorising long numbers is assumed to be hard enough to be used for cryptographic purposes. Consequently, it is very unlikely that a factorisation appears from nowhere just using SMAC. Looking at probabilities, we get

$$P(x \notin \mathbb{Z}_n^\times) = \frac{p+q-1}{pq} = \frac{1}{q} + \frac{1}{p} - \frac{1}{pq} \approx \frac{1}{\sqrt{n}}.$$

This is exactly the same probability breaking RSA by chance when guessing possible factorisations.

Ergo, from the security point of view there is no reason not to use an arithmetic circuit over the ring \mathbb{Z}_n for a digital signature. But what about the complexity of such a signature?

6.3 Communication Complexity

Let d be a 1024 bit private key and n a 1024 bit modulus. The message m that is to be signed is initially a hash value that is most of length 160 bit. This length changes during the S&M algorithm by potentiation and modulo reductions. So, we assume the worst case which is 1024 bit in each step. We know from chapter 2.1 that each distributed multiplication requires two message exchanges between all agents.

$$1024 \text{ S\&M steps} \times 2 \text{ distr. multiplications} = 2048 \\ \text{message exchanges per agent}$$

The data volume to be transferred by one agent is

$$2048 \text{ message exchanges} \times 1024 \text{ bit} = 0.25 \text{ MB} = 2.1 \\ \text{MBit}$$

With a realistic number of 10 agents in an Alliance (which gives us a redundancy of $t = 3$) the data volume that is to be communicated for one signature is 25 MB. Table 1 shows the time needed to perform one signature for different bandwidths currently in use. It is assumed that the communication of the single agents is in parallel which means the total time is assessed by the time needed to transfer 2.1 MBit. As most private Internet connections are asynchronous, the upload rate (which is lower than the download rate) is used. One can easily see that all results are practical.

7 Conclusion

The overall evaluation of this approach to a threshold signature is very positive. A threshold signature using SMAC is cryptographically secure and practical. Using the ring \mathbb{Z}_n no modulo operations are necessary. Doing so, one loses unconditional security, but the model still stays cryptographically secure which can be considered absolutely satisfactory for real applications. There are other theoretical approaches as for example [10, 6, 5, 4, 9] which offer similar security properties (i.e. robustness), but it may be doubted that they reach the same efficiency as the approach presented here. While those approaches make use of costly broadcast simulation, this is not necessary when using SMAC for a digital signature. The reason for this is that the preparation phase of the SMPC protocol is done by the originator of an Alliance who can be

Type	T1	ADSL	SDSL	Cable	Cable	Sky-DSL
Users	prof.	private	business	business	private	private
Downstream/s	1 GBit	16 MBit	2 MBit	20 MBit	6 MBit	1 MBit
Upstream/s	1 GBit	1 MBit	2 MBit	10 MBit	600 KBit	64 KBit
s/signature	0.0021	2.1	1.04	0.21	3.5	32.77

Table 1. Time required for one signature

assumed to have no interest in manipulation his own agents. Unfortunately, none of publications mentioned above discusses the complexity in detail. But it seems that the approach presented here, offers a very good constant factor when using a finite ring. In addition, the competing methods have been designed for signatures, only. In contrast, SMAC can be used for every boolean function with finite fan-in gates. Non-robust distributed approaches are much more efficient but produce incorrect results in case of wrong input shares. This is not satisfactory in open and unauthenticated networks like the Internet.

References

- [1] Christoph Amma. Über die Implementierung kryptographischer Primitive mittels sicherer Multiagentenberechnungen. Master's thesis, Universität Karlsruhe (TH), 2006.
- [2] 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.
- [3] E.R. Berlekamp and L. Welch. Error correction of algebraic block codes. US Patent Number 4,633,470, 1986.
- [4] Christian Cachin. An asynchronous protocol for distributed computation of rsa inverses and its applications, 2003.
- [5] R. Cramer and V. Shoup. Signature schemes based on the strong rsa problem. *ACM Transactions on Information and System Security*, 3(3):161–185, 2000.
- [6] Y. Desmedt and Y. Frankel. Shared generation of authenticators and signatures. In *Proc. of CRYPTO '91*, pages 457–469. Springer Verlag, 1992.
- [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] Regine Endsuleit and Arno Wagner. Possible attacks on and countermeasures for secure multi-agent computation. In *Proc. of Int. Conf. on Security and Management*, pages 221–227. CSREA, 2004.
- [9] Marc Fischlin. The cramer-shoup strong-rsa signature scheme revisited. In *Public Key Cryptography — PKC '03*, volume 2567, pages 116–129. Springer, 2003.
- [10] Yair Frankel, Philip D. MacKenzie, and Moti Yung. Robust efficient distributed RSA-key generation. In *The Thirtieth Annual ACM Symposium on Theory of Computing – STOC '98*, pages 663–672. ACM Press, New York, 1998.
- [11] Anna G' al. Semi-unbounded fan-in circuits: Boolean vs. arithmetic. In *Proceedings of the 10th Annual Symposium on Structure in Complexity Theory*, pages 82–87, 1995.
- [12] 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 Verlag, 2001.
- [13] 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.
- [14] 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.
- [15] Adi Shamir. How to share a secret. *Communications of the ACM*, 22:612–613, 1979.
- [16] Bennet S. Yee. A sanctuary for mobile agents. In *Secure Internet Programming*, pages 261–273, 1999.