Available online at www.sciencedirect.com

SCIENCE DIRECT®

Information and Software Technology 48 (2006) 726–744

INFORMATION AND SOFTWARE TECHNOLOGY

www.elsevier.com/locate/infsof

A framework for anonymous but accountable self-organizing communities

Gábor Ziegler [a], Csilla Farkas [b,*], András Lőrincz [c]

[a] High Speed Networks Laboratory, Department of Telecommunication & Media Informatics, Budapest University of Technology and Economics, Hungary
[b] Information Security Laboratory, Computer Science and Engineering Department, University of South Carolina, Columbia, SC 29208, USA
[c] Department of Information Systems, Eötvös Loránd University, Budapest, Hungary

Received 21 July 2004; revised 23 August 2005; accepted 24 August 2005
Available online 28 November 2005

Abstract

In this paper we propose a novel architecture and approach to provide accountability for Web communities that require a high-level of privacy. A two-layered privacy protection architecture is proposed, that supports (i) registration of participants and enforcement of community rules, called *internal accountability*, and (ii) rule-based interaction with real world organizations, called *external accountability*. Our security protocols build upon community-based trust and limit the exposure of private data on trusted third parties.

The two-layered architecture protects the mappings between real users and their virtual identities, and among the virtual users, while guaranteeing internal and external accountability. We target Web communities that are dynamic and self-organizing, i.e. roles and contributions of participants may change over time. The proposed concepts and protocols are implemented in our SyllabNet project that supports anonymous course evaluations by university students.

Keywords: Accountability; Anonymity; Privacy; Trust; Self-organizing

1. Introduction

One of the most fascinating possibilities of the World Wide Web and its underlying communication infrastructure, the Internet, is their support of (electronic) collaboration beyond the traditional geographical limits. A key factor of these collaborations is their 'virtuality', that is, instead of real users only their *virtual identities* are observable. The need to provide privacy in this virtual world led to theoretical research as well to several prototype implementations (see Section 6 for an overview). These technologies provide data, communications, or users anonymity. Our claim is that full anonymity poses a serious practical security problem, allowing malicious users to avoid accountability for their actions.

This paper addresses the trade-offs among the concepts of anonymity, accountability, and evolution (self-organization). Our aim is to enable systems that (i) support free and rule-based collaborations among their members—*self-organization*, (ii) hide user identity data—*anonymity*, and (iii) make users responsible for their actions—*accountability*.

Our aim is to develop a tool for communities that (i) supports handling information requests from the external world, (ii) provides means for investigating inconsistencies of the internal decision making, (iii) guarantees the anonymity of the users and investigators, and (iv) supports dynamic decision making process. Our solution also provides means to the community members to look at the data stored about them and—upon condition—terminate data. We denote this collaborative framework as *Anonymous Accountable Self-Organizing Community*, or shortly A2SOC. The research challenge is how to establish such communities and how to safeguard their operations. Our preliminary results were given in [12].

We present a framework supporting anonymous collaborations of virtual users while providing means to hold users responsible for their activities. We use the term *accountability* for this latter requirement throughout the paper. We use *pseudonymity* to hide the identity of a user but allow to reveal this identity under specific circumstances. The main focus of this work is to develop methods that guarantee that the mapping between a user's real identity and the corresponding pseudonym cannot be disclosed, unless this disclosure is justified. This justification may be given by the *virtual community* (e.g. to enforce internal requirements) or by the external *society* (e.g. to enforce legal restrictions). Existing solutions to safeguard user identity while providing accountability [8,10,14,22,31] require a trusted authority or authorities.

[*] Corresponding author. Tel.: +1803 576 5762, fax: 1 803 777 3767.
E-mail address: farkas@cse.sc.edu (C. Farkas).

0950-5849/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2005.08.007

A limitation of these approaches is that it is not always possible to find such authorities. We propose a new method that distributes trust among the members of the community without the need of the existence of a single trusted entity. The disclosure of a user's identity requires the collaboration of a quorum of users, thus our protocols are applicable in peer-to-peer and self-organizing virtual communities.

Our solution is a two-layered anonymity model. The first layer provides mapping among the different virtual identities of the same real user without revealing the identity of the real user. The second layer provides mapping between a real user and the user's base virtual identity. The support of the community is required to reveal the mapping at each layer. Disclosure of the mapping between a real user and his/her virtual identities also requires external justification, like court order. We develop the concepts of *internal* and *external* accountability:

(1) *Internal accountability* is when the virtual (pseudonym) member of a group is identifiable within the group and can be held responsible for his/her actions according to the 'ethic' or policy of the group.
(2) *External accountability* is when the real entity behind the virtual member of a group is identifiable and can be held responsible for his/her actions according to the 'ethic' or the law of the external environment hosting the group.

We develop five protocols to handle (1) verified member registration, (2) non-verified member registration, (3) registration of a new virtual identity, (4) release of real user identity, and (5) verification of personal data. These protocols use public- and secret-key encryption methods to guarantee message confidentiality and authenticity. We use threshold cryptography to safeguard sensitive data. The protocols are discussed in the paper; their step-by-step descriptions are given in Appendix.

Finally, we present our implementation, a 'practical choice' of the above concepts for a real world application. In our example, students can evaluate course materials, can design their own course maps using a graphical map editor, have a map publishing web application, and can keep their anonymity under A2SOC principles. In this particular case, internal accountability means that individual maps, their annotations, forum entries, etc. can be evaluated by the community and the *value* can be associated or mapped to virtual user identities in a non-ambiguous fashion. External accountability concerns cases when the real identity of the user needs to be revealed. For example if the activities of a virtual user are not allowed by the rules of the external community (e.g. legal requirements) the real person should be penalized. Clearly, appropriate justification is required to reveal the identity of the real user.

Novel aspects of our paper include the formalization of A2SOC systems, the development of the protocols to maintain anonymity and accountability and the detailed description of our implementation.

The rest of the paper is organized as follows. In Section 2 we define our problem domain, Section 3 deals with the general solution. Section 4 gives a more formal definition of our accountable anonymity model and the formal analysis of the assurance of the protocols. Our implementation details are given in Section 5, together with the SyllabNet [2] project, our practical prototype system. Related works are reviewed in Section 6.

We conclude and recommend future research directions in Section 7. We close the paper with the details of our accountable anonymity security protocols (Appendix A).

2. Problem domain

In this section we present our problem domain and the technical challenges of our work. We begin by describing some real-world problems, explanatory examples, where the application of A2SOC principles could help. Then we continue by defining *abstractions* such as anonymity, accountability, and self-organization. We discuss conflicts between these abstractions and introduce the concept of anonymous, accountable, and self-organizing community.

2.1. A2SOC application examples

We illustrate our problem domain with examples from different domains. First, consider an open-source, collaborative software development environment, like SourceForge [30]. In such developments there are two opposing roles, the software developer and the quality assurer. Depending on the application, anonymity of either, or both of these roles might be required.[1] Furthermore, the same real person can be either a developer for one project and a quality assurer for another. However, these roles are incompatible within one project project, because there is a conflict of interest between these roles. Therefore, coincidence between developer and quality assurer needs to be avoided. That leads to the problem of internal accountability. Also, in general, there is no warranty that software developers are using their own names and are not malicious. This leads to issue of accountability that such communities could take advantage.

Our second example is taken from the domain of medical applications. There are special medical problems, e.g. Parkinson disease, where novel hi-tech solutions, such as Deep Brain Stimulator, are entering medical practice. It is typical that newsgroups are formed by the patients or those, who are considering undergoing such operations. Participants of these newsgroups may be using their own names, or may be using nicknames. This is contradictory, because from the point of view of their own personal rights, they should use nicknames, whereas from the point of view of the community they should be accountable.

From the point of view of the patient it would be also important to find the optimal medical product on the market. Since there might be a competition between manufacturers, they must be accountable for both their products and their advertisements. Patients may participate in A2SOC newsgroups. It is necessary to hide the identities of the patients. Consider that doctors have different surgical practice concerning the particular nature of the

[1] Note, these roles correspond almost exactly to the roles of authors and referees in scientific journals.

surgical problem and the instrument they apply in the surgery. It is important for the patient to find the optimal solution: i.e. a good surgeon, who has high success rate for his particular operation and is close for the possibly frequent check-ups. Taking the example of Parkinson disease, the failure rates differ considerably for different problems, operation methods, and hospitals [28]. Such data are uncovered directly or indirectly by patient communities. However, subjective judgements may require adjustments, which corresponds to the external accountability.

2.2. A2SOC concepts

In this subsection we clarify our terminology.

Prototype in artificial intelligence is seen as the single best example of a class. Other members of that class are distorted versions of the prototype. *Abstraction*, in our model, is a special prototype, which has an additional feature: *it does not exist*.

Many members may belong to the class defined by an abstraction. These members are the approximate instantiations of the abstraction; these are *distorted versions* of the non-existent prototype. Deviations from the abstraction provide the measure about the quality (the distortion) of an instantiation.

We use five basic abstractions: community, interaction, self-organization, compatibility, and accountability.

Community: is defined recursively as follows:

Community The trivial community has a single member. A community can be formed by a set of members, or by a set of communities. A member can be a human, or a robotic agent. A community is characterized by its *members* and by the *rules* of the community. A community becomes an abstraction by *separating* the community from its environment in which the community is embedded.

A member, or a participant of the community is described by the role played by the member and rules concerning that role. The concept of the member is also an abstraction; it separates the member from the rest of the community.

Interaction: Interaction is the effect of two communities up to which they cannot be separated, i.e. up to which they can influence each other. One particular form of interaction is information exchange, or, *communication*. Two forms of interaction are distinguished here:

Tightly coupled interaction is rule-based, or *algorithmic* interaction.

Loosely coupled interaction has cost (reward, or punishment) based components and can be subject to optimization.

A telecommunication network with its signaling protocols is a good example of the tightly coupled interaction. Protocols that permit negotiations are loosely coupled. In most practical cases, the interaction of a community is a mixture of the two components, because, cost can be the result of an algorithm, whereas the execution of an algorithm may involve certain costs. An agent using reinforcement learning (see, e.g. [32] and references therein) is an example to such mixtures. Note also that algorithmic interaction assumes that all possible interactions are known. This assumption is an abstraction if the environment is uncertain, which is the case, e.g. for telecommunication. Also, rules may become obsolete or incomplete, because of self-organization.

Self-organization: Self-organization is the abstraction that describes the emergence of new features that cannot be (has not been) foreseen from the known properties. In turn, using the word of self-organization, we try to cover the emergence of new qualities, the emergence of higher order functions, and so on. Taking a broader view, a process is a self-organizing process if a higher level of organization may emerge from it. In this sense, synonyms of self-organization include development and evolution.

Compatibility and incompatibility: If we have an interaction scheme, then the issue of *compatibility* arises. A tightly coupled interaction is compatible if the interaction rules are always satisfied. Note, that if one of the interacting community is a black-box to the other then—even in theory—it is impossible to establish if the interaction is tightly coupled or not. On the practical side, one might consider the validation problem of the communication protocols [17].

Corruption: When a tightly coupled interaction proves to be (partially) incompatible, it is called *corrupted*.

Negotiation: Non-rule based (i.e. loosely coupled) interactions between communities are called negotiations. Such interactions may aim to find satisfactory solutions or to remove inconsistencies.

Voting: A particular form of non-rule based interactions aims to make decisions in the absence of rules. For example, such interactions may serve to fix corrupted interactions. Although many different types of such interactions may be envisioned, here, it is satisfactory to consider thresholded decision-making, or voting.

Anonymity: The term *anonymity* is used as a general term in this paper for any private information (i.e. 'secrets') that an individual would like to save from being known to others. An example could be the name of the individual, hence the word, anonymous. Secrets do not have to be abstractions. They may become abstraction when they need to be considered together with accountability. For our purpose, anonymity means some of the secrets, which are necessary for accountability, but should be hidden, from anyone within or outside of the community. A special form of anonymity is *pseudonymity*, where the identity of a member is protected, but all individual interactions of the same member can be non-ambiguously attributed to this member

Accountability: We consider two different types of *responsibilities* with respect to the results of interactions:

Internal accountability: This is the cost-based part of accountability. Members engaged in interactions may be *responsible* for the outcomes of the interaction within the community, that is, their state or their status within the community may be changed. Enforcement of this type of accountability is based on punishment and reward.

External accountability: External accountability is the rule-based part of accountability. Occasionally some members, or the community as a whole may be subject to rules between communities. These rules serve to maintain conformance and, unlike to cost-based interactions, may concern the disclosure of secrets, e.g. the disclosure of certain part of private information. For example, performing legal actions may be necessary against a pseudonymous member of the community. Another example would be to promote a pseudonymous author of the A2SOC journal into the Editorial Board of that journal, provided if he/she is not a member of the Board under a different virtual name.

2.3. Discussion of the problem domain

Clearly, accountability and anonymity are conflicting concepts. In addition, the very nature of self-organizing processes is subject to extensive research and is not fully understood yet. They might create an ever-expanding list of obstacles to enforce certain rules of the community.

The protection of the personal data, i.e. anonymity, should not be without limits. The precise delineation of those limits is a philosophical question, which is outside of the scope of this paper. However, we can say, that anonymity should be guaranteed only as long as the individual is not breaking the rules of the (internal or external) community. If a member violates these rules, then the member should be identified, prosecuted, and—if found guilty—be punished. We support the safeguarding of the personal data, identification of members violating the (internal or external) rules, and some level of prosecution by guaranteeing the correctness of our techniques. This led us to the concept of *external accountability*, which is the dual of internal accountability. External accountability deals with the justifications necessary to reveal the protected secrets of the members (e.g. to uncover anonymity).

Architectures to safeguard private data may vary from centralized solutions, where a single trusted entity protects and knows all secrets; to distributed, democratic decision making, such as voting, where no single entity is able to make a critical decision alone, like revealing a secret.

Moreover, voting can be seen as the tool of decision making in cases when decisions are needed, but there are no rules to apply. Corruption of rules or rules that need to modified, deleted, established may be typical in self-organizing communities. Voting may also concern changes of the rules about secrets.

Our particular engineering solution concerns the voting procedure. The voting procedure determines if a member should be excluded from the community or not. We require that a minimum number of members, say k members have to agree to reach a decision.

3. Building the A2SOC model

3.1. Considerations on anonymity in self-organizing community

Our approach to provide accountable anonymity is based on (1) the requirement to be able to track activities of a user under a single or multiple pseudonyms, and (2) the requirement that associations between real users and their virtual identities should not be disclosed unnecessarily. These requirements are guaranteed by the use of partially trusted third parties (TTP).

Our solution is inspired by the layered communication model of OSI [1]. Any application server (AS) hosting a virtual community can be enhanced by wrapping it into an anonymizer layer (TTP) that enforces the concept of pseudonymity. The general architecture of the system is depicted in Fig. 1. The TTP acts as a 'firewall' between the users and the AS. That is, TTP realizes the access control function and maintains the book keeping of internal accountability.

Two layers of associations are maintained:

Layer 1 (External accountability) associates a real user with a 'base' virtual identity.
Layer 2 (Internal accountability) associates virtual identities, based on their relations to their 'base' identity.

Both associations can be confidential. They are distinguished here: Internal pseudonymity concerns the disclosure or matching of

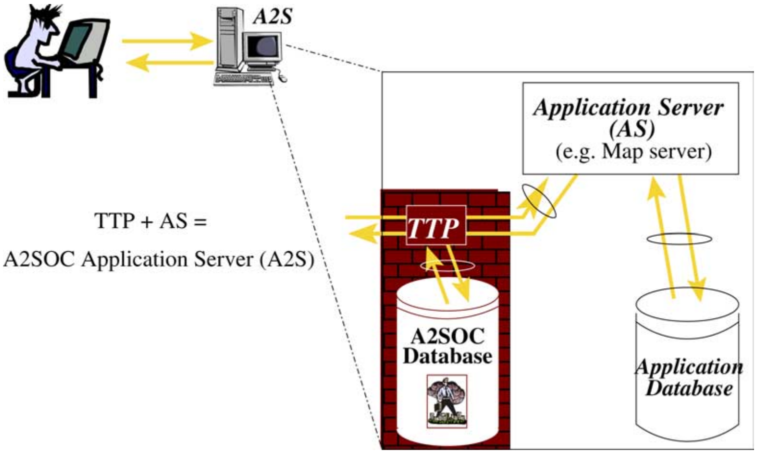


Fig. 1. A partially Trusted Third Party (TTP) together with a general purpose Application Server (AS) can form a so-called A2SOC Application Server (A2S).

certain virtual identities to fulfil the rules of the community. External anonymity is also subject to rules and concerns a disclosure procedure, with the support from the TTP. In our model, external accountability—called also as external pseudonimity—invokes the cooperation of a group of distinguished users.

First, we define our assumptions:

- **A Public-Key Infrastructure** is assumed to be available, including the existence of a trusted Certificate Authority. The TTP also has limited certification authority; that is used within the virtual domain only. Each party is assumed to have public-key encryption/decryption and secret-key encryption/decryption capabilities.
- **Trusted Third Parties** (TTP-s) are responsible for managing the real and virtual user identities and the corresponding databases.
  - **A2SOC Application Server (A2S)**: One TTP (say *TTP*$_1$) is responsible for the internal pseudonymity, allocation of virtual identities, and book-keeping of the virtual identity trees. Note that if one uses only a single TTP then the registration procedure of new users might pose either privacy, or registration data authenticity problems (see Section 4.3 below).
  - **Trusted A2SOC Authority (TAA)**: If an another, separated, second TTP (say *TTP*$_2$) is responsible for verifying the users' identities, then the A2S does not necessarily have to see it in unprotected (i.e. 'cleartext') form.

    Authentication of registration data is an important building block of external accountability). The TAA acts similarly to public notaries during registration: as an unbiased third party it certifies the validity of registration data without ever revealing its actual content to the A2S.

    We assume that they operate correctly, i.e. upon registering a (virtual) user; they discard any sensitive data as required by the protocols. Compromise of a single TTP, after the initial user registration, should not effect *external accountability*.
- **Users** are all entities who wish to register for pseudonyms. All users must possess a set of required proofs to register for base-identity. In the simplified form, this is a digital certificate issued by a mutually trusted Certificate Authority (CA). Each user also has a pair of public and private keys certified by the trusted certification authority. Any users can get at most one base-identity; otherwise internal accountability could not be enforced.
- **Virtual users** are the pseudonyms acquired by the users or other virtual users. The number of virtual users registered by a (virtual) user maybe bounded. The first pseudonym registered for a user is called base identity. Any new virtual identity, registered to a real or virtual user, will receive a signed, digital certificate of the registering TTP. This certificate carries the virtual identity and a corresponding public key.

We consider the protection of internal pseudonymity information less sensitive; therefore it is guarded only by the partially trusted A2S. This separation has practical and not theoretical considerations: The internal accountability algorithms may frequently need to traverse the identity tree. Therefore, protecting this tree by a voting requirement is often a too heavy burden. The lack of voting means that internal accountability, which might concern evaluations and costs, is a tightly coupled interaction. This solution is satisfactory, because the community has a voting procedure and can change tightly coupled interactions.

The protection of the data related to external accountability is highly sensitive and we do not trust a single TTP to safeguard this information. We use threshold cryptographic techniques to distribute the decryption key, needed to reveal this secret, among a distinguished group of users. We call this group the *Board Members* (BMs for short). None of the Board Members holds the decryption key alone, since they are only partially trusted, like the TTP. The disclosure of a secret constitutes a democratic voting process.

A given minimum number of BM-s (which corresponds to the threshold of the cryptosystem) should agree on the decryption to reach a quorum, i.e. to be able to decrypt. We use the TTP only to perform some of the computations and coordinate the generation of the fragments of the shared private key.

3.2. Accountable Anonymity Model

Definition 3.1. (*Personalization/Identity Tree*). Let I_0 denote a base virtual identity of a real user u, and the set V denote the set of all virtual identities v_i. We say, that a personalization tree is a node-labelled tree, where the root is labelled with $v_0=I_0$ and for every other node the following holds:

(1) If there is an edge from node v_i to node v_j, where $v_i, v_j \in \mathcal{V}$, such that v_i is the parent of v_j, then v_i registered the virtual identity v_j.
(2) Each personalization tree (unique I_0) is mapped to a unique user identity u.

Definition 3.2 (*Internal Accountability*). Let v_i and v_j be any two virtual identities of our model, such that there is a path p in the personalization tree from v_i to v_j. We say, that the model preserves internal accountability if it is impossible for any user or group to see p without proper justification.

Definition 3.3 (*External Accountability*). Let u be a real user and v_i a virtual user such that v_i is in the personalization tree $\mathcal{T}$ with root I_0 such that $\nu(u)=I_0$; that is, the root of the tree containing the v_i is the base identity of u. We say, that the model preserves external accountability if it is impossible for any user or group to see $\nu(u)=I_0$ and the path p from I_0 to v_i without proper justification.

Definition 3.4 (*Justification*). Let T denote the number of votes needed (i.e. the threshold) from the community to disclose a mapping between a real user and a base virtual user. Let there be N users of the community who have voting rights, each has vote $v_i=1$ ('yes'), or $v_i=0$ ('no'). Let there be $0<l\leq N$ collected votes at a given time. The request of disclosure is considered valid if $\sum_{i=1}^{l} v_i \geq T$.

Definition 3.5 (*Rule Based Accountability, or Compatibility*). Let v_i and v_j be any two virtual entities of our model. We say, that accountability over v_i and v_j can be enforced, if

(1) it can be established that v_i and v_j are two different virtual identities of the same real user, but the identity of the real

user is not revealed. (Internal compatibility),

(2) it can be established that v_i is a virtual identity of real user u. (External compatibility, or external accountability)

The need to reveal one or both of the layers of associations depends on the seriousness of the misuse. Minor misuse, say offensive posting on the web site, may result in penalizing the virtual identity. For example, it might be satisfactory to decrease the internal accountability value (e.g. rank, role, rights, etc.). Offensive posting initiated by different virtual identities may be subject to further investigations if it is suspected that the different virtual identities belong to the same real user. Some misuses may involve legal actions. We assume that appropriate policy to describe community-based and external requirements is available, all users agreed to the policy, and they are aware of the possible consequences of their actions. Our work focuses on the enforcement of these policies.

3.3. Providing accountability

We use threshold cryptography for encrypting the associations between real and virtual entities, i.e. for providing external accountability. A TTP encrypts the mappings between real and virtual users, performs the book-keeping of the internal accountability, and handles user requests for new virtual identities. However, in contrast to most of the works based on trusted entities, like ticket granting service or to a certificate authority, after the mappings are encrypted, the TTP is unable to decrypt these mappings. To disclose the association between a real user and its virtual entity, a voting process has to be initiated. Only the Board Members may take part in the voting procedure.

This feature limits the power of TTP and ensures confidentiality of earlier associations even in the presence of a compromised TTP. The identity of an Anonymous Accountable Community (A2C) is defined by the key-pair $K_{AC} = \left(K_{AC}^{Pr}, K_{AC}^{Pu}\right)$. The private key K_{AC}^{Pr} is shared among the Board Members and the public key K_{AC}^{Pu} represents the community for the outside world. Note, that there exist threshold cryptosystems, where the private key is never (re) constructed at a single place [35], which might provide extra security.

In Section 4.3 we provide protocols for real and virtual users to obtain new virtual identities. The confidential information kept by the TTP-s ensures that accountability can be maintained. The developed model permits the followings:

- Without revealing the identity of the real user it can be determined whether two different virtual entities belong to the same real user (internal accountability).
- The identity of the real user corresponding to a virtual identity (cf. external accountability) can be revealed.

As we mentioned earlier, internal accountability is used to enforce community-based restrictions. For example, the same user cannot be the author and reviewer of the same document. This kind of accountability is maintained by the TTP in the current model. The external layer supports real world-based restrictions. In addition, users are permitted to observe the data stored about them. This consideration originated from the observation that users feel more confident about a system if they have the power to monitor its collection of their own personal data. Access control for personal information is supported by cryptographic and database techniques.

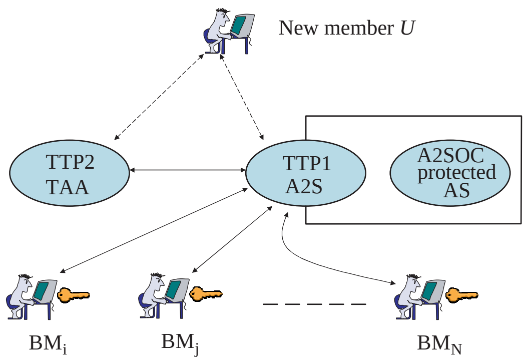
We assume that proper mechanisms to mask the users' IP addresses and to protect against other web navigation-based identification methods are available. Our main focus is to hide the mapping between the virtual entities as well as between the real users and their virtual entities in a way, that these mappings can be revealed in justified cases. In the current model, a decision by the TTP to reveal a mapping between virtual entities is considered justified. The request to reveal the mapping between a virtual entity and a real user is considered justified only if a group of users agrees on the request and if the TTP supports this request (similar to the requirement of a court order in real life).

For protecting the external accountability we use the above mentioned threshold cryptosystem with an asymmetric key. The community uses the well-known public key for encryption and the distributed private key for decryption. The protection of the external accountability is done by encrypting the sensitive data with the public key of the community.

The *j*th fragment of the shared private key of the community is held and known only by the *j*th Board Member (BM_j) and is denoted as $K_{A2C,j}^{Pr}$. The public key of the cryptosystem can be used as a traditional public key, while using the private key requires at least T (the 'threshold') number of BM members to cooperate. No group of BM-s that consists of less than T members shall be able to successfully apply the private key. Any group of BM-s that contains at-least T cooperating members shall be able to apply the private key.

For example, a cryptosystem fulfilling the above requirements has been developed by Boneh et-al. within the ITTC project [6] at Stanford University, see also [5,21,35]. Their cryptosystem is compatible with the RSA cryptosystem in the sense that an ITTC public key can be used as an RSA public key. This means that applications using RSA cryptosystems can transparently use ITTC cryptosystems provided that appropriate shared-key servers are available to serve as a distributed private key repository.

As we will show in Section 5, the ITTC cryptosystem can be adapted to provide an A2SOC cryptosystem to fulfill the requirements stated in this section. In this adaptation 'A2SOC board members' act as the 'ITTC shared key servers'. There is a small (but easily solvable) difference between the A2SOC and the ITTC model in the sense that the ITTC cryptosystem always uses *exactly T* (instead of *at-least T*) number of shared-key servers. For example, among any group of $m > T$ number of *willfully cooperating* BM-s there can always be made a (randomly generated) agreement to choose exactly T members who performs a particular decryption, then the mapping between the two cryptosystems becomes straightforward.



Legend: **TTP**=Trusted Third Party,
A2S=Accountable Anonymizer Server,
TAA=Trusted A2SOC Authority,
AS=Application Server,
A2SOC=Accountable Anonimity for Self-Organizing Communities

Fig. 2. The general architecture with two TTPs.

3.4. The need for two TTPs

As we will see in Section 4.3 if we would like to only partially trust the TTP, then we need to separate the role of TTP into two (refer to Fig. 2). The main problem is that we would like to eliminate the need for trusting a single entity (even if the need for trust is just temporary). We do not want to trust any single entity that it handles our personal data correctly, which means that we cannot let it to see sensitive data in cleartext form, since it would be a 'full trust'. However, to perform the verification of real users' identities a TTP must see their personal data in cleartext. Consequently, we must trust this TTP that it will immediately and voluntarily drop the sensitive cleartext data after encryption. Furthermore, if we consider to participate in several A2C communities then we have to trust even more TTPs, one for each community.

As a solution we can split the responsibilities into two disjoint fields and divide them between to kind of TTPs (resulting in a partial-trust model): one entity (say TTP_1, or Accountable Anonymity Server—*A2S*) is responsible for the internal accountability and allocation of base virtual identities; and an other entity (say TTP_2 or Trusted A2SOC Authority—TAA) is responsible for external accountability, that is, verifying user identities.

The TAA can be a community-independent one, in the sense that it is not involved with the community's everyday life beyond assisting in user verification. Therefore, many communities can trust in a single *TAA*. The main responsibility of the TAA is to verify and certify users' identities by suitable means. (E.g. checking digital certificates, examining real-world paper documents, etc.) After verification, the TAA takes over the role of encrypting the privacy sensitive information from the A2S.[2]

Users trust the TAA that it verifies their identity fairly and does not reveal these identities in clear-text form to anybody. Each A2S trusts a TAA in the sense, that the TAA indeed verifies the users on the behalf of the A2C community that the A2S represents. Users present their identity to the TAA, which verifies the data, encrypts the verified data with the public key of the target community and sends the encrypted data over to the respective A2S, after signing the encrypted data with its own private key. The target A2S is not able to comprehend the encrypted data, but it can verify the signature of the TAA whom it trusts.

Based on this trust the A2S can accept the encrypted registration data as if it had been encrypted itself.

4. A2SOC Model

4.1. Notations for the model

We assume the existence of reliable public-key distribution media. We use the notation of K_A^{Pu} to represent the public key of entity A; K_A^{Pr} to represent the private key of entity A; the key-pair is denoted as $\left(K_A^{Pr}, K_Z^{Pu}\right)$.

$E[K, M]$ is the encryption of message M with key K, and $D[K, M]$ is the decryption of message M with key K. The ith fragment of the shared private part of a key K_A of a threshold cryptosystem is denoted as $K_{i,A}^{Pr}$. Decryption of a message M with this shared private key requires the cooperation of a suitable coalition $\mathcal{C}$ of shared-key holders, such that $|\mathcal{C}| > T$, where T is the threshold of the cryptosystem. Decrypting in this latter case denoted as $D\left[K^{Pr}_{\forall i\in\mathcal{C},A}, M\right]$. The shared-key holders of an A2SOC are called *Board Members*, they form a set: $BM_k \in \mathcal{BM}$, where the following property must hold: $|\mathcal{BM}| > T$.

We define signing of a message M with some key K as signing the hash $H(M)$ of the message with the private half of the key K: $E[K^{pr}, H(M)]$. We concatenate the message with the signed hash to make the pair $S_K[M] = \{M, E[K^{Pr}, H(M)]\}$.

We use U to represent the real user; *parsed*(U) to represent the identification of U as being parsed and transformed to a suitable form for machine processing and storage, e.g. an XML document. I_i is used for representing the ith virtual identity of user U. The first virtual identity (the base identity) that user U first acquired is denoted as I_0.

The pair $\{parsed(U), I_0\}$ is a two-tuple which will be stored as the identity map $IdMap(U, I_0)$ between user U and its unique base identity I_0. The identity map signed with the private key of the base virtual identity is

$$SIdMap(U, I_0) = S_{K_U^{Pr}}[\{parsed(U), I_0\}].$$

The signed identity map $SIdMap(U, I_0)$ that is encrypted with public key K_{A2C}^{Pu} of the community for storage is denoted as

$$ESIdMap(U, I_0) = E[K_{A2C}^{Pu}, SIdMap(U, I_0)].$$

There should be a 'virtual user identity' U_{I_j} attached to each virtual identity I_j. This virtual identity is fictual, defined arbitrarily by virtual user I_i who register the child-identity I_j.

[2] Note that—in principle—we have established a self-organizing hierarchy, because any TAA may have a voting procedure in its rules.

4.2. Data structures

Data structures maintained by the TTP:

DB-Real Users: contains the mapping between the real users and their 'base' virtual identities and their statuses. It stores

$$\{I_0, E[K_{A2C}^{Pu}, SIdMap(U, I_0)], Status_U\},$$

where U is the real user's identity; I_0 is the base virtual identity of U, and $SIdMap(U, I_0)$ is the signed identity map of U and I_0.

The signing of the mapping (U, I_0) provides means for user U to check that its registration data is not tampered (integrity check). Also, this provides non-repudiation, i.e. the real user belonging to I_0 cannot deny that indeed user U has acquired originally the I_0 base identity.

Note, that to decrypt $E[K_{A2C}^{Pu}, SIdMap(U, I_0)]$ the community's private key K_{A2C}^{Pr} is needed. This key is shared among the members; therefore to decrypt this formula collaboration of the BM-s is needed.

$Status_U$ is the status of user U, e.g. whether (s)he is permitted to play an active role in the community or (s)he is not, etc.

DB-Virtual Users: contains the mapping between virtual identities. It stores $\{I_i, I_j\}$, where I_i is the virtual identity that activated a new virtual identity I_j. Note, that we consider this association less sensitive than the associations between the real and virtual user. Therefore, for the sake of efficiency, this pair is only protected by the TTP. Clearly, if required, strong protection, similar to *DB-Real Users* can be applied.

Virtual Users: contains the relationship among activated virtual identities. We store these relationships as rooted, and directed trees, where the root is a base virtual identity, the nodes are activated virtual identities, and there is an edge from I_i to I_j iff I_i activated I_j. Virtual Users is a forest of exclusive trees, each tree is labeled by its root.

4.3. Protocols

Early version of the A2SOC protocols has been published in the paper [12], where different cryptographic technique was used than the one here.

In the followings we describe the A2SOC protocols in general. A more detailed explanation of them is given in the Appendix A. The following protocols are covered there:

Protocol 1: *Verified registration* describes the protocol for registering a verified user into the A2SOC community. This protocol provides the strongest level of accountability (both external and internal). The protocol contains the following eight steps, which are detailed in Appendix A. In Fig. 3 there are numbered arrows corresponding the numbered steps below:

(1) User U sends Registration Request to the TAA (TTP_2)
(2) The TAA sends Pre-Allocation Request to the A2SOC Application Server (A2S, TTP_1), in order to temporarily reserve the chosen identity until the final succesful completion of the protocol.

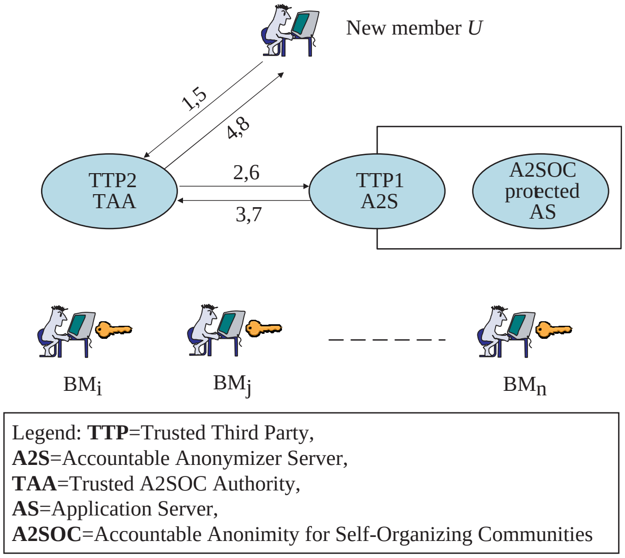

New member U
1,5
4,8
2,6
3,7
TTP2
TAA
TTP1
A2S
A2SOC
protected
AS
BMi
BMj
BMn
Legend: TTP=Trusted Third Party,
A2S=Accountable Anonymizer Server,
TAA=Trusted A2SOC Authority,
AS=Application Server,
A2SOC=Accountable Anonimity for Self-Organizing Communities

Fig. 3. Initial sign in with TAA. For message details refer to Protocol 1.

(3) The A2S replies with Pre-Allocation Acknowledge to the TAA.
(4) The TAA sends Sign-Identity-Map Request to user U
(5) User U replies with Sign-Identity-Map Acknowledge to the TAA (TTP_2)
(6) The TAA sends Store-Identity-Map Request to the A2S
(7) The A2S replies with Store-Identity-Map Acknowledge to the TAA
(8) The TAA sends the final Registration Acknowledge to user U

Protocol 2: *Non-verified registration* describes a simplified protocol. It omits the rigorous verification of the user's identity except checking the user data form U for syntactical correctness. This kind of logins might be suitable for users to obtain a base virtual id with restricted rights, e.g. users with a passive, observing-only role, such as read-only accounts. This effectively provides internal accountability only, since the real users' identities are not verified.

The protocol contains the following four main steps, which are detailed in Appendix A. In Fig. 4 there are numbered arrows corresponding the numbered steps below:

(1) User U sends Registration Request to the A2S.
(2) The A2S sends Sign-Identity-Map Request to User U
(3) User U sends Sign-Identity-Map Acknowledge to the A2S
(4) The A2SOC Application Server sends Registration Acknowledge to User U

Protocol 3: *Registration of a descendant virtual identity* describes the process of registering a new virtual identity I_j using another, already registered identity I_i. The new I_j identity becomes descendant of I_i in the identity tree. This protocol contains a sequence of four messages, therefore it is visualized in the same Fig. 4, like as Protocol 2.

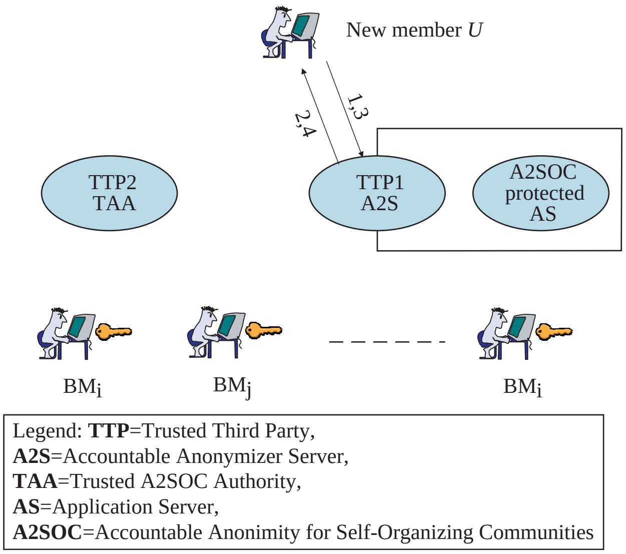

New member U
1,3
2,4
TTP2
TAA
TTP1
A2S
A2SOC
protected
AS
BMi
BMj
BMi
Legend: TTP=Trusted Third Party,
A2S=Accountable Anonymizer Server,
TAA=Trusted A2SOC Authority,
AS=Application Server,
A2SOC=Accountable Anonimity for Self-Organizing Communities

Fig. 4. Initial sign in without TAA, or registering a child-id. For message details refer to Protocol 2 and Protocol 3.

(1) The user with virtual identity I_i sends Registration Request for virtual identity I_j to the A2S server TTP_1.
(2) The A2S (TTP_1) sends Sign-Identity-Map Request to the parent user I_i.
(3) The user I_i sends Sign-Identity-Map Acknowledge to the A2S
(4) The A2S (TTP_1) sends Registration Acknowledge to the user I_i.

Protocol 4: *Voting about uncovering the real user behind an accused virtual user* aids a requester (e.g. a user) to obtain the real identity of some accused user of the community (external accountability). The protocol contains a sequence of six main steps corresponding to the numbered arrows in Fig. 5, which are detailed in Appendix A.

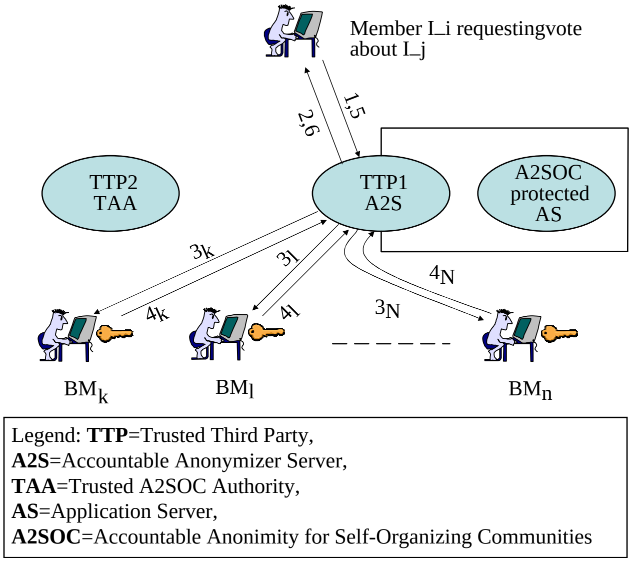

Member I_i requestingvote
about I_j
1,5
2,6
TTP2
TAA
TTP1
A2S
A2SOC
protected
AS
3k
4k
3l
4l
4N
3N
BMk
BMl
BMn
Legend: TTP=Trusted Third Party,
A2S=Accountable Anonymizer Server,
TAA=Trusted A2SOC Authority,
AS=Application Server,
A2SOC=Accountable Anonimity for Self-Organizing Communities

Fig. 5. The voting process. For message details refer to Protocol 4.

(1) User I_i wants to reveal the real identity behind user I_j, therefore sends a Vote Request to the A2S.
(2) The A2SOC Server TTP_1 sends a Vote Pending message to user I_i
(3) The A2S TTP_1 sends out Voting Indications message to each Board Member: $\forall k, BM_k \in \mathcal{BM}$
(4) Each kth Board Member BM_k responds with a Vote Respond message
(5) Optionally, user I_i can check the status of an ongoing voting by sending the message Check Vote Request to TTP_1 any time
(6) The A2SOC Server TTP_1 notifies user I_i about the status of a previously initiated vote with the message: Vote Status. This message is sent by the A2S either upon
    - receiving a Check Vote Request from the same user I_i who initiated the voting.
    - The vote has been either successfully or unsuccessfully ended due to either completing the vote or observing a timeout

Protocol 5: *'Vote-about-self'* provides means for a user who wants to check the registered data about himself/herself to retrieve his/her own registration data for checking purposes. The protocol is very similar to Protocol 4, actually it is a special case of that: the requester and the 'accused' are the same user I_i.

4.4. Assurance of security protocols

One of the major problem in dynamic, decentralized systems is the maintenance of information that is necessary for security processing. In our system, the mapping ν: *real user → base virtual identity* is encrypted with the public key of the A2C community, and the private key is distributed among the Board Members of the community by threshold secret sharing.

Theorem 4.1. The A2SOC security protocols provide

(1) internal accountability
(2) external accountability

The proof of the theorem follows naturally from the processes enforced by the protocols.

Proof: First we prove the second part, that is, that the A2SOC protocols provides external accountability (Definition 3.3). The content of the database *Real users* of the A2S (Section 4.2) is encrypted (thus protected) by the K^{Pu}_{A2C} key of the community. Revealing the mapping between the real user U and a virtual user I_i can be done by either of the following two ways:

(1) Other user(s) may want to look up the mapping $\nu = IdMap(U, I_0)$. According to Step 4 of Protocol 4 the A2S cannot reveal the mapping ν to the real user without the shared private key K^{Pr}_{A2C}. Since the private key is usable by
    - neither the A2S alone,
    - nor any subgroup $\mathcal{G} \subset \mathcal{BM}$, if $|\mathcal{G}| < T$

therefore, the mapping can neither be revealed by the A2S alone, nor by hackers breaking into the A2S. Since the private key is distributed as decomposed subkeys among the A2SOC BM-s, it cannot be reconstructed without a quorum of the A2SOC BM-s. Reaching a quorum means the ‘justification’ itself in our interpretation.

(2) The user may look up its own data (refer to Protocol 5). It trivially cannot be a violation of anonymity.

On the other hand, the A2S on behalf of the community can enforce the internal accountability onto the virtual users.

Furthermore, as it is described in Protocol 4, in ‘justified’ cases (when the A2SOC community has achieved a quorum) it is possible to reveal the real identity of the user, which establishes external accountability. □

Note, that our model does not guarantee accountability if the voting process fails. Voting may fail if there are not enough active board members $BM_K \in \mathcal{BM}$ to form a voting majority, or if communication failures occur. This problem is outside of the scope of our work. Solutions to this issue cover a very broad range including the two opposite and limiting solutions (1) *nothing happens*, which means that there is no consequence and accountability may be in danger (2) *requested data is uncovered* for the external world, which should be dangerous for the community. Nevertheless, the rules of A2SOC may include both cases, e.g. (1) suits low priority problems, whereas (2) may be necessary in the medical field. The current system has a time limit for coming to a decision. We interpret the case when a Board Member may deliberately abstain from voting (i.e. withhold his/her share) as exercising community-based trust. This approach, indeed guarantees the democratic nature of our trust management.

5. A testbed for A2SOC principles

To test our ideas in practice, the construction of a proof-of-concept testbed project was decided about. The testbed is being developed jointly at the Budapest University of Technology and Economics and the Eötvös Loránd University. For the groupware application to be extended by A2SOC principles we have chosen the Coraler Hostess (as the application server (AS)) and the Coraler MapEditor/MapViewer [2] as the client program for the Coraler Hostess). The choice of the Coraler system is justified by its map-based versatility and that it is freely available for academic use.

5.1. Motivation for SyllabNet

The main purpose of the Coraler Hostess/MapEditor/MapViewer system is to provide means for graphical organization and visualization for knowledge management through interactive maps. The original Hostess server is used for storing knowledge maps and provides access control to individual maps via user rights management. The interface to the system is either the MapViewer application, or the MapEditor application.

The former is intended to be embeded into web pages (portals) providing read-only browsing access to the stored maps. It is available as either a Java or Flash applet.

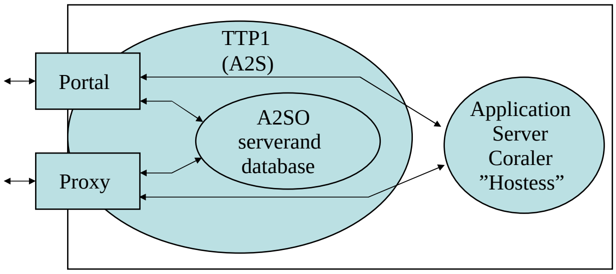
The MapEditor is a standalone application that has to be downloaded and installed at the users. The MapEditor can be used either on-, or off-line. If it is used on-line then it covers all the functionality of the MapViewer plus makes it possible to create and/or change maps (if the user has sufficient access rights for doing so). It communicates with the Hostess server using web-services [33] technology, i.e. SOAP [34] over HTTP protocol. As a summary, from the point of view of A2SOC principles it is just ‘a’ client-server application that is used for collaboration over the Internet. (It is like a web forum that runs on a server and is accessed by the users via a web-browser client).

We work on enhancing this application with A2SOC principles as a proof-of-concept project. We plan to use it for suggesting curricula maps for students who would like to get some guidance to choose among the various courses made available at their, or other universities, including course and test materials, which represent international standards and are made available on the Internet. In defining this goal we have been inspired by the MIT Open CourseWare (OCW) initiative [25], which provides a plethora of available courses that could be nicely interleaved with the curricula of our universities. This will allow students to measure their knowledge against the MIT OCW materials.

5.2. SyllabNet architecture

The project required practical compromises. The Coraler Hostess itself had a built-in virtual identity management; therefore we have decided to partially rely on the Hostess for the virtual identity tracking. A web-service proxy has been developed, which provides all interfaces (SOAP ports) of the Hostess. It selects among the web-service calls: it redirects all user management related activities and those web-service calls that has to be tracked to the A2S due to the internal accounting, otherwise it transparently proxies the calls to the Hostess.

The implementation of SyllabNet contains three main building blocks beside the Hostess server that are protected



Legend: **TTP**=Trusted Third Party,
A2S=Accountable Anonymizer Server,
A2SOC=Accountable Anonimity for Self-Organizing Communities

Fig. 6. The building blocks of SyllabNet.

by the A2SOC principles. These three blocks are the followings (refer to Fig. 6):

The A2SOC server and database plays the role of A2S (TTP_1)

The Portal provides the graphical user interface to the A2S and also embeds the read-only Coraler MapViewer

The Proxy routes the communication coming from the MapEditor either to the A2SOC Server (TTP_1) or to the application server (i.e. the Coraler Hostess), appropriately

A stripped down and modified version of the system (containing only a TAA server and database; a TAA-proxy and a TAA-portal) serves as the TAA.

Since the Hostess/MapEditor/MapViewer was also intended for portal-embedding; some A2SOC protocol steps were implemented as HTTP form submission over SSL link, some as HTTP document retrieval and some as e-mail sending via the SMTP protocol. The original Coraler Hostess has required a registration email address, which we found useful since it provides means to initiate contact to the users (e.g. sending a vote indication to BM-s).

The SMTP communication also provides means for asynchronous store-and forward communication (i.e. the sender and receiver entities do not have to perform simultaneously the communication acts). For example, it makes possible to send a message to a receiver who is currently off-line, and the message can be furnished with suitable encryption, too.

In Protocol 1 the Steps 1, 5, and 8 are implemented as sending HTML forms and retrieving HTML documents over HTTPS. Step 4 is implemented as sending an e-mail to the registration e-mail address. (Users will be instructed to choose a suitable anonymous e-mail account.) Steps 2,3,6, and 7 are realized as web-service calls.

Similarly in Protocol 2 and 3 the Steps 1,3, and 4 are implemented as sending HTML forms and retrieving HTML documents over HTTPS. Step 2 is implemented as sending an e-mail to the registration e-mail address.

In Protocols 4,5 the Steps 1,2,5 are implemented as sending HTML forms and retrieving HTML documents over HTTPS. Step 3 is realized as sending an e-mail to the registration e-mail address of the Board Member. The voting Step 4 is realized as a web-service call. The status/result of the voting (Step 6) is realized as either an asynchronous e-mail in case of completing a successful vote, or as retrieving a HTML document in case of an ongoing vote.

For the threshold cryptosystem the ITTC system [35,6] of Stanford University was adapted and slightly modified to meet our purposes, because we faced the following problems.

Off-line usage. The ITTC system was designed for continuous, synchronous and on-line operation, where the off-line status of a shared-key server is not the normal way of operation.

Asynchronous communication. The shared-key servers are supposed to answer synchronously to decryption requests. However, the human Board Members of SyllabNet are likely to be off-line and not to answer promptly on a voting-indication message.

Non-deterministic coalition selection. If the original ITTC system is about to perform a decryption via the shared-key servers then it does the followings:

(1) It chooses a random suitable coalition of shared-key servers among the supposedly on-line servers. (The ITTC system keeps track of the on-line status of the shared-key servers continuously.)
(2) Contacts all shared-key servers within the chosen coalition and attempts to ask them to decrypt. The servers are supposed to willingly take part in the process unless they are corrupted.
(3) If the request is unsuccessful and it is possible to choose another different coalition from the on-line and uncorrupted servers, then the ITTC client-library repeatedly attempts the decryption each time using a new coalition.

In the SyllabNet project the Board Members might be off-line, might choose to dissent to the voting request, might be late to answer, etc. It is also not-desirable to repeatedly contact a particular Board Member who has approved the vote (that is, generated his/her partial decrypt for a particular coalition) about the same vote just because the original coalition was not able to produce a quorum.

Therefore, in our adaptation we transformed the original ITTC model to an asynchronous A2SOC model, where Board Members might be off-line and might disapprove any decrypting request.

The A2S sends out the voting-indication message to all Board Members. Any Board Member BM_i who approves the vote will generate his/her partial decrypts for all possible coalitions $\mathcal{C}_i$ that (s)he might be member of: $\exists PD_{k,\mathcal{C}^i} \,|\, \forall i\, \mathcal{C}^i$. This makes possible to request a vote from a BM only once for any given vote-campaign, regardless the particular coalition that can be decided later.

5.3. Discussion of ITTC in SyllabNet

For A2SOC purposes the ITTC cryptosystem suffers from some bottlenecks. Two main problems became apparent during the project:

The ITTC does not scale well. For a k-out-of-N threshold scheme there are $\binom{N}{k}$ possible coalition. The ITTC cryptosystem requires a computation of different key-share per BM for each possible coalition. As a consequence our A2SOC adaptation requires $\binom{N}{k}$ partial decrypts to be calculated and sent-over for an approving vote. This does not scale well if the number of BM-s grows.

Recalculation of shared secrets. Suppose that some of the Board Members are to be replaced. Recalculating the shared secrets of the ITTC cryptosystem invalidates the old public key as well. This means that the 'Board' membership has to be predetermined and quasi-static. This is a clear obstacle of value based selection of Board members and requires further work.

6. Related works and discussion

The increased use of electronic media in every day life generates new concerns regarding users' privacy. Anonymity providing technology has emerged to protect the confidentiality of private data. Martin [23] gives an overview of the anonymizing techniques for the Internet. In general, these technologies can provide data or communications anonymity, and personalization. For example, Onion Routing [3], Crowds [27], and Hordes [29] provide connection anonymity. Systems GNUnet [4], FreeNet [11], and Napster [24] support file-sharing services while guarantee different levels of anonymity. In addition, several models have been developed to support specific applications, such as anonymous e-mail [9,16] or electronic commerce [10,20,19].

Many existing reputation systems are built upon transitive trust, whereby positive and negative observations are accumulated and aggregated to produce reputation values. Buchegger and Boudec [7] introduced a Bayesian approach for peer-to-peer reputation management. In the scheme, peers maintain reputation and trust rating for all peers with which they have interacted directly. The observations are aggregated using Bayesian estimation to address dishonest nodes. However, the method fails to adequately deal with nodes that provide false information.

Kamvar, Schlosser and Garcia-Molina [18] proposed the EigenRep schema, which uses transitive trust and efficiently propagates reputation information. Local observations are exchanged and aggregated to produce both global and local reputation values. This solution is highly susceptible to node collusion without the presence of multiple pre-existing trusted nodes. Additionally, the pre-existing trusted nodes introduce a central point of failure.

The Pinocchio network [13] rewards participation and honesty within a global, centralized distributed computing environment. The trust management system is based on using community gossip to infer third party service provider ratings. Peers are offered incentives for active network participation and punished for freeloading. This scheme does not protect against collusion or observation falsification.

Garcia-Martinez and Chuang [15] proposed a cryptographic scheme based on the notion of 'reputocracy', where a node's reputation is its most important asset. The reputation values are stored as a file randomly within the network identifier space, and verified with the use of secure voting via threshold cryptographic schemes. This scheme requires significant overhead in terms of establishing and maintaining communication. Also, the scheme utilizes a trusted authority to facilitate the cryptographic exchange, thus introducing a central point of failure.

The closest to our work is presented in [8,14,22,31]. For example, Buttyan et al. [8] present a ticket-based system, which allows anonymous accesses to the services. The paper addresses the important issue of costumers' lack of trust is the service providers, need of scalability and accountability. Their architecture is based on the existence of a customer care agency that is trusted by both client and service provider. They provide 4 types of tickets with varying bonding to the customer and service provider. However, they place full trust in the customer care agency, thus its compromise would potentially disclose all previous and future costumer information. Also, their model provides 1 level of anonymity, that is, mapping between real users and tickets. Therefore, it may unnecessarily reveal the identity of the real user even if only similarities among the virtual identities is considered.

In general, current technologies to provide anonymity or pseudonymity are either fully anonymous, thus lack accountability, or—if accountability is addressed—fully dependent on a trusted mediator (certificate authority, customer care agency, etc.). Furthermore, these systems do not provide access to the users to observe or terminate their personal data or terminate their data if they do not want to participate in a given community any longer. Finally, only one layer of anonymity is supported, where the need to validate whether two virtual entity belongs to the same real user, requires the disclosure of the real entity's identity. In this paper we provide solutions to address the above shortcomings of these models in a common framework.

6.1. The core of the problem

The problem may already arise in a two-community world, where each community has a single member, and these two communities form the whole world, that is, the first community is the environment of the other one and vice versa. There are only two possibilities for initiating identity disclosure between such interacting communities:

(i) investigation: the first member has some information and initiates queries about the secrets of the second member,
(ii) reporting: the first member has some information and requests queries about its own secrets from the other member.

An example for case (i) is some 'solid' piece of evidence that secrets of the second member have to be uncovered. As an example, consider the master-slave situation, when the joint performance—as observed by the 'master'—is not satisfactory and the activity table of the 'slave' needs to be disclosed. An example for case (ii) may occur, for example, if the master is uncertain, whether the payment has reached the 'slave' and asks the 'slave' to query and match the master's bank account about recent payments to the slave.

6.2. General A2SOCs

The example shows that our model is a particular model for A2SOCs. Different levels of secrets and different disclosing procedures may coexist. One may think of the members as databases that may ask queries concerning certain data of other members. The answers may give rise to new questions concerning the other database. That is, the task is database management, where access rights are determined by an algorithm.[3]

Any algorithm, including the interaction algorithms can be corruption free, or corrupted. However, in principle, there is no general way of proving that interaction algorithms are corruption free. For self-organizing communities, it seems better to consider that algorithms are either

- corruption sensitive (i.e. there are ways to corrupt the algorithm),
- or corrupted.

Within this framework, error free interaction becomes an abstraction and negotiations as well as changes of the rules need to be considered. The extreme case is if there is no direct interaction between the members, but it is indirect and is limited to reward and punishment generated by the environment the members are embedded into. This situation may occur in self-organizing systems, and has been studied by some of us in the form of a self-assembling Internet agent system [12,26]. This value based part of the interaction was not considered in this paper. We restricted ourselves to corruption sensitive interaction algorithms. Also, we did not consider negotiation processes that may follow some evidence of corruption.

Our most important assumption is that for compatible interaction algorithms, there is a single origin of incorrect operation and this is the lack of information. Lack of information then involves certain information exchange mechanism about the 'secrets' of the partners. Taking another view, A2SOCs can be subject to rules that minimize communication and communicate only if necessary.

7. Conclusions

In this paper we presented a framework for self-organizing electronic communities, and developed security protocols that enforce accountability, while providing anonymity. Our main focus was to reduce the possible exposure of private information, i.e. the mappings between real and virtual users, and limit the trust in any single entity or computing resource, e.g. TTP. We developed layered personalization architecture. External accountability requires higher level of confidentiality than internal accountability, because external accountability protects the mapping from the real users to their base virtual identities. Internal accountability is supported by the second layer that contains mappings between virtual identities. We propose a relaxed security for the internal accountability layer to increase efficiency, flexibility, and provide transparency for new identity activation. Our detailed protocols are given in the Appendix.

Finally, in addition to the development of the technical solutions, this paper contains our discussions on the trade-offs between security, privacy, and self-organization. Our aim is to motivate further investigations of these concepts, aid the development of theoretical models, and support practical implementations. Clearly, these tasks involve the collaboration of experts from different domains, like social science, artificial intelligence, computer science, and security.

Acknowledgements

This work was partially supported by the National Science Foundation under Grants No. INT-0304904 and IIS-0237782. Support was also provided by the Hungarian National Science Foundation (Grant OTKA 32487) and by EOARD (Grant FA8655-03-1-3084). The implementation of the SyllabNet project was partially funded by the Hungarian Ministry of Informatics and Communications (Grant ITEM-2002/222). The SyllabNet project includes software written by Mike Malkin, Tom Wu, and Dan Boneh [6]. The PI of the A2SOC project, the medical and training applications is the third author.

Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the European Office of Aerospace Research and Development, Air Force Office of Scientific Research, Air Force Research Laboratory, or the US National Science Foundation.

Appendix A. The A2SOC protocols

In this section we describe basic use-cases of our proposed system. For each use-case we have defined a corresponding *protocol* that realizes the task. These protocols contains message interactions that are numbered according to their temporal ordering. This temporal ordering is also shown by numbered arrows in the corresponding Figures along the protocol descriptions.

Protocol 1 Initial sign in with the Trusted A2SOC Authority (TAA) in order to receive a verified virtual identification I_0.

Refer to the numbered arrows in Fig. 3 for the respective steps.

Step 1: User U sends Registration Request to the TAA (TTP_2):

$$E\left[K_{TTP_2}^{Pu}, \left\{U, K_U^{Pu}, I_0, S_{K_U^{Pr}}[R], t_1\right\}\right],$$

where
U is the user's (certified) electronic identification;
I_0 is the requested base identity;
$S_{K_U^{Pr}}[R]$ is the request signed by user U.

Upon receiving the message, TTP_2 (the TAA) decrypts the message, verifies the user's identity (e.g. via the certified user data U), and checks *Real Users* that there is no virtual entity exists for U and U is not forbidden to activate a virtual identity.

If the user is permitted the activation, TAA checks and verifies if the base identity I_0 is indeed free and allocatable by querying TTP_1 (A2S) in the next step.

[3] We note that the model relies on the assumption that the answering time between databases is zero. This assumption, however, may be hard to satisfy and needs to be investigated.

Step 2: The Trusted A2SOC Authority TTP_2 sends Pre-Allocation Request to the A2SOC Application Server (A2S), TTP_1:

$$E\left[K^{Pu}_{TTP_1}, S_{K^{Pr}_{TTP_2}}[I_0]\right],$$

where
$E\left[K^{Pu}_{TTP_1}, S_{K^{Pr}_{TTP_2}}[I_0]\right]$, means that the preallocation request is encrypted for the A2S after signing by the TAA for certification.

Upon receiving the message, the A2S checks whether I_0 is free. If yes, then it pre-allocates the I_0 identifier and acknowledges the request in the next step.

Step 3: A2S sends Pre-Allocation Acknowledge to the TAA

$$E\left[K^{Pu}_{TTP_2}, S_{K^{Pr}_{TTP_1}}[I_0]\right],$$

where
$E\left[K^{Pu}_{TTP_2}, S_{K^{Pr}_{TTP_1}}[I_0]\right]$, means the I_0 identifier first signed by TTP_1, then encrypted for TTP_2.

Upon receiving the message, the TAA knows that I_0 is free and pre-allocated. (The pre-allocation will become a final allocation later, in Step 6.) As the next step, the TAA will request the user U to sign the identity-mapping between U and I_0.

Step 4: The TAA sends Sign-Identity-Map Request to user U:

$$E\left[K^{Pu}_{U}, IdMap(U, I_0)\right],$$

where
$IdMap(U, I_0) = \{parsed(U), I_0\}$ is the identity map as an ordered tuple.
$parsed(U)$ is the machine-processable personal data record of user U, as
parsed from the submitted original data, which is probably some plain text.

Upon receiving the message, user U should confirm that his/her identity is indeed correctly transformed into $parsed(U)$; then (s)he should generate the public-private key-pair $K^{Pu}_{I_0}$ and $K^{Pr}_{I_0}$ for the identifier I_0. (S)he should sign the $IdMap$ with K^{Pr}_{U}, producing $SIdMap(U, I_0) = S_{K^{Pr}_U}[IdMap(U, I_0)]$.

Its reason is twofold: On one hand User U cannot deny that user U received his/her identifier I_0. On the other hand user U can optionally check that the identity-mapping is indeed correct in Protocol 5.

Step 5: User U send Sign-Identity-Map Acknowledge to the TAA (TTP_2):

$$E\left[K^{Pu}_{TTP_2}, (K^{Pu}_{I_0}, SIdMap(U, I_0)\right]$$

where
$SIdMap(U, I_0)$ means the signed identity map $S_{K^{Pr}_U}[\{parsed(U), I_0\}]$.

Upon reception of the message the TAA checks to signature on $SIdMap(U, I_0)$ with K^{Pu}_{U}; encrypts the $SIdMap(U, I_0)$ with the well-known public key K^{Pu}_{A2C} of the community and sends it over to the A2S representing the community.

The $ESIdMap(U, I_0) = E\left[K^{Pu}_{A2SOC}, SIdMap(U, I_0)\right]$ (that is, the encrypted and signed identity map) is unintelligible for TTP_1, therefore the A2S needs to trust the sender that it indeed has verified the identity map. In order to support this, the TAA signs the encrypted map before sending it over to the A2S in the next step.

Step 6: The TAA (TTP_2) sends Store-Identity-Map Request to the A2S (TTP_1):

$$E\left[K^{Pu}_{TTP_1}, S_{K^{Pr}_{TTP_2}}[\{I_0, ESIdMap(U, I_0)\}]\right].$$

where $S_{K^{Pr}_{TTP_2}}[\{I_0, ESIdMap(U, I_0)\}]]$ is the $ESIdMap$ (U, I_0) signed by TTP_2, as mentioned at the explanation of Step 5.

Upon reception of the message the A2S verifies the signature of TTP_2, then the A2S finalizes the allocation of I_0, by storing the $ESIdMap$(U, I_0) with index I_0 in DB-Real Users and acknowledges the successful end of the registration in the next step.

Step 7: The A2S sends Store-Identity-Map Acknowledge to the TAA

$$E[K^{Pu}_{TTP_2}, S_{K^{Pr}_{TTP_1}}[I_0].$$

Upon reception of the message, the TAA records in it own version of *Real Users* that user U has obtained a base identity I_0.

In the last step it will acknowledges the registration for user U.

Step 8: The TAA sends the final Registration Acknowledge to user U:

$$E\left[K^{Pu}_{U}, I_0\right].$$

Upon reception of the message user U got notified about the succesful registration for base identity I_0.

End The verified registration ends here.

Protocol 2 Initial sign in without TAA (TTP_2) to the A2S (TTP_1), in order to receive a non-verified virtual identification I_0

Refer to the numbered arrows in Fig. 4 for the respective steps.

Step 1: User U sends Registration Request to the TTP_1:

$$E\left[K^{Pu}_{TTP_1}, \left\{U, K^{Pu}_{U}, I_0, S_{K_U}[R], t_1\right\}\right].$$

This step corresponds to the respective Step 1 of Protocol 1.

The TTP_1 (A2S) decrypts the message and verifies that the base identity I_0 is indeed free and allocatable by checking *VirtualUsers*.

Step 2: The TTP_1 sends Sign-Identity-Map Request to User U:

$$E\left[K_U^{Pu}, IdMap(U, I_0)\right].$$

The A2S request signing the identity map, for the same reasons as for Step 4 of Protocol 1. The user U generates the keypair $K_{I_0}^{Pu}$ and $K_{I_0}^{Pu}$ and signs the *IdMap*.

Step 3: User U sends Sign-Identity-Map Acknowledge to the A2SOC Application Server (TTP_2):

$$E\left[K_{TTP_1}^{Pu}, \left(K_{I_0}^{Pu}, SIdMap(U, I_0)\right)\right]$$

This step is similar to Step 5 of Protocol 1.

Upon reception the message the A2S encrypts the $SIdMap(U,\ I_0)$ with the well-known public key K_{A2SOC}^{Pu} of the community and stores the encrypted identity map

$$SMap(I_0) = E\left[K_{A2SOC}^{Pu}, SIdMap(U, I_0)\right]$$

with the index I_0 in *Real Users*. The A2S is supposed to drop the cleartext version of *parsed*(U).

Step 4: The A2SOC Application Server TTP_1 sends Registration Acknowledge to User U

$$E\left[K_{I_0}^{Pu}, I_0\right].$$

In this last step user U got notified about the successful registration for Base id I_0 (refer to Step 8 of Protocol 1).

End There non-verified registration ends here.

Note, that this protocol requires an increased level of trust in TTP_1.

Firstly TTP_1 does see the clear-text version of *parsed*(U), so we must trust it that it does not misuse the data. Secondly we must trust it that it will delete the clear-text version of *parsed*(U) from its data-storage immediately after finishing Step 4.

Until this our data is vulnerable to any third-party intruder who might break into the system. There is no means for the User U to verify this deletion act, unfortunately.

This problem does not occur with Protocol 1, since TTP_2 does not see the clear-text version of *parsed*(U) in that protocol. Although TTP_1 does see the clear-text version in Protocol 1 that is unavoidable anyway: in order to guarantee user U's identity somebody *must* read its identity-documents at the first step of the certification chain).

Protocol 1 provides a guarantee via TTP_2 that the identity which can be revealed with voting (refer to Protocol 4) is indeed not a fake one. Furthermore, it ensures that a single real user U cannot obtain multiple base virtual identity I_0. The price for this advantage is that we need an additional party beyond the A2S server: the TAA server. It must be partially trusted by both parties, i.e. the A2SOC Application Server and user U. However, if we suppose that the electronic identity U of user U is indeed a 'real identity' from the point-of-view of the law, then it means that is issued by some legally acknowledged authority. The tasks of the TAA is very similar to those tasks that is routinely performed by such an authority, who is supposed to be capable of issuing digital certificates. Vesting this authority with the additional role and power of being a TAA should not cause technical problems, nor does it mean additional trust toward this authority other than that is already required by law.

Protocol 1 provides a simplified registration procedure for the price of lesser accountability, since there is no guarantee that the registered identity of user U is not a fake one. Furthermore, since no verification is done nothing prevents a user to obtain multiple pseudonym base identities that cannot be interrelated from within the inside of the A2SOC system only.

Protocol 3 Register another (child) identity $\mathrm{I_j}$ with TTP using virtual identity $\mathrm{I_i}$

Refer to the numbered arrows in Fig. 4 for the respective steps. Note, that this protocol does not have multiple versions for verified, or non-verified case. The main difference between this protocol and Protocols 1 and 2 is that those protocols addresses the cases where a user who is *outsider* of the community register its base virtual identity I_0.

Step 1: The user with virtual identity I_i sends Registration Request for virtual identity I_j to the A2S server TTP_1.

$$E\left[K_{TTP_1}^{Pu}, \left\{I_i, I_j, S_{K_{I_i}^{Pr}}[R]\right\}\right]$$

where

I_i is the parent identity,

I_j is the requested new (descendant) identity, and

$S_{K_{I_i}^{Pr}}[R]$ is the request for identity I_j signed by the private key of user I_i as the proof of origin.

The virtual user with the identity I_i registers a child-identity I_j. The mapping between I_i and I_j will be similarly protected like the mapping between U and I_0 in Protocols 1 and 2.

The TTP_1 (A2S) decrypts the message and verifies that the new identity I_j is indeed free and allocatable by checking *Virtual Users*.

Step 2: The A2SOC Server TTP_1 sends Sign-Identity-Map Request to the parent user I_i:

$$E\left[K_{I_i}^{Pu}, IdMap(I_i, I_j)\right].$$

The A2S requests signing the identity map, for the same reasons as for Step 4 of Protocol 1.

Step 3: The user I_i sends Sign-Identity-Map Acknowledge to TTP_1:

$$E\left[K_{TTP_1}^{Pu}, (K_{I_j}^{Pu}, SIdMap(I_i, I_j)\right]$$

User I_i generates the key pair $\left(K_{I_j}^{Pr}, K_{I_j}^{Pu}\right)$ signs the identity map as virtual user I_i and sends it back to the A2S as in Step 5 of Protocol 1.

Upon reception the A2S adds to *VirtualUsers* I_j as the child of I_i.

Step 4: The A2S TTP_1 sends Registration Acknowledge to the user I_i:

$$E\left[K_{I_i}^{Pu}, I_j\right].$$

In this last step user I_i got notified about the succesful registration of descendant id I_j.

End The child-identity registration ends here

Protocol 4 A user with virtual id $\mathrm{I_i}$ requests to reveal the true identity U of a virtual id $\mathrm{I_j}$. Note that $\mathrm{I_i}$ and $\mathrm{I_j}$ may be the same identity for the purpose of checking the stored registration data. The Board Members judge the case.

Refer to the numbered arrows in Fig. 5 for the respective steps.

Step 1: User I_i wants to reveal the real identity behind user I_j, therefore sends Vote Request:

$$E\left[K^{Pu}_{TTP_1}, I_i, S_{K^{Pr}_{I_i}}[\{I_j, Cause.\}]\right]$$

where

I_i is virtual identity of the requester

$S_{K^{Pr}_{I_i}}[\{I_j, Cause.\}]$ is the voting request signed by user I_i for authenticity

I_j is the accused virtual identity

Cause is the justification cause for the voting request to be honored.

User behind I_i requests a vote about uncovering the identity of a real user U_j behind the virtual user I_j. To request the vote (s)he sends its own identity I_i. (S)he also signs and sends the identity I_j of the user who is the object of the vote and the justification *Cause* of the request to the A2S in an encrypted form.

Upon reception the message, the A2S authenticates the request based on its punishment level (*PunishLevel*[*i*] for user *i*, see Step 4 of this protocol for more about punishments), which shall be less than the community specific value *votingPunishLimit*. If the request can be granted (i.e. *PunishLevels*[*i*] $<$ *votingPunishLimit*), then A2S acknowledges the request by sending a message of type *VotePending* to user I_i. Then the A2S retrieves both the base identity I_0 behind I_j and the tree of identities *IdTree* that rooted at I_0 and contains I_j; and initiates the voting by notifying each board member BM_k, $k = 1\ldots N$.

The A2S also initializes counters pros and cons to 0; the variable terminate to 'continue', assigns a unique id V_l to the vote, and creates the empty set of votes $\mathcal{D}_{V_l}$.

Step 2: The A2S TTP_1 sends a Vote Pending message to user I_i:

$$E\left[K^{Pu}_{I_i}, \{I_j, Cause, V_l, Timeout\}\right]$$

where

Cause is the cause for the request

V_l is the reference to the initiated voting process

Timeout is the timeout value for the cause

Upon receiving the message user I_i has been notified about the pending voting process by receiving back the object of the vote with the cause and the timeout value to be expected. The voting is an asynchronous communication process with possible failures. The user I_i will be notified about either the outcome of the voting, or the timeout event at Step 6 (shown on page 40).

Step 3: The A2S TTP_1 sends out Voting Indications message to each Board Member BM_k:

$$E\left[K^{Pu}_{BM_k}, V_l, I_i, I0, IdTree, S_{K^{Pr}_{I_i}}[\{I_j, Cause\}], ESIdMap(U, I_0)\right].$$

where

V_L is the vote identifier

I_i is virtual identity of the requester

*I*0 is the base identity of the accused person

IdTree is tree of identifiers which is rooted in I_0 that I_j belongs to

$S_{K^{Pr}_{I_i}}[\{I_j, Cause\}]$ is the voting request signed by I_i

$ESIdMap(U, I_0)$ is the protected data to be decrypted if the voting request is to be granted.

After all voting requests are sent out to the BM-s, the A2SOC Application server starts a so-called 'guard timer' T_{V_L} for this V_L voting process with an expiry time of *Timeout*.

The Board Member BM_k will judge the case and make his/her decision. (S)he has three possibilities:

Voting 'Yes' Assent to the *Cause* and approves the revealment of the real identity behind I_i: $Decision_{k,V_l} :=$'Yes'. This also means that BM_k applies his shared-key fragment to the decryption process resulting in a partial decrypt:

$$PD_k = D_k\left[K^{Pr}_{A2C,k}, ESIdMap(U, I_0)\right].$$

Voting 'No' Dissent to the *Cause* and disapproves the revealment of the real identity behind I_i: $Decison_{k,V_l} :=$'No'

Voting 'No, with punishment' Dissent to the *Cause* and beside disapproving the revealment of the real identity behind I_i, (s)he decides that even the voting request was an offence. Therefore, BM_k requests the punishing of user I_i due to the seemingly unjustified voting request: $Decision_{k,V_l} :=$'No, with punishment'

Step 4: Each Board Member BM_k is supposed to respond with a Vote Respond message:

$$E\left[K^{Pu}_{TTP_1}, \{Decision_{k,V_l}, PD_k\}\right].$$

where

$Decision_{k,V_l}$ is the decision of Board Member *k* about the vote V_l, according to Step 3 above.

PD_k is optional message part, if the $Decision_{k,V_l}$='yes', then it carries the partial decrypt of the secret by Board Member *k*.

The A2S waits for the reception of the decisions of the individual Board Members until either all responses has arrived or timer T_{V_L} has expired as follows:

The outcome of the vote is stored in the variable *terminate* and continuously being updated in

```
PROCEDURE proc_conduct_vote:
    $cons_{V_L} := 0$; $pros_{V_L} := 0$;
    $terminate_{V_L}$ := 'continue';
    WHILE ($terminate_{V_L}$ == 'continue')
    DO
        call proc_eval_vote
    END_WHILE
    IF ($terminate_{V_L}$ =='approved')
    THEN call proc_succ
    ELSE IF ($terminate_{V_L}$ =='refused')
    THEN call proc_fail
    END_IF
END_PROCEDURE
```

procedure proc_eval. Note, that the WHILE loop will always terminate, since either the vote and will be finished, or a timeout will occur.

The expiry of timer T_{V_L} can be caused either by unreliable data communication channels, or by the Board Members itself. The latter means that the BM-s may fail to vote (i.e. they might abstain)—either intentionally, or not. The possibility of abstaining is an inherent feature of any democratic system. However, it makes providing accountability more difficult, refer to Section 4.4.

Upon either the reception of a particular $Decision_{k,V_l}$ from BM_k, or the expiry of timer T_{V_L} the A2S records the $Decision_k$ into $\mathcal{D}_{V_l}[k]$ according to the following procedure:

```
PROCEDURE proc_eval_vote:
    $\mathcal{D}_{V_L}[k] := Decision_{k,V_l}$
    IF ($Decision_{k,V_l}$ =='Yes') THEN
        $pros_{V_L} := pros_{V_L} + 1$;
    ELSE IF ($Decision_{k,V_l}$ =='No'
    OR $Decision_{k,V_l}$ =='No, with punishment')
    THEN $cons_{V_L} := cons_{V_L} + 1$;
    END_IF
    IF ($pros_{V_L}$ >threshold)
    THEN $terminate_{V_L}$ :='approved'
    ELSE IF ($cons_{V_L}$ >N-threshold)
    THEN $terminate_{V_L}$ := 'refused';
    ELSE IF (isExpired($T_{V_L}$))
    THEN $terminate_{V_L}$ := 'timeout'
    END_IF
END_PROCEDURE
```

where

threshold is the T threshold of the shared private key of the community, and

N is the number N of the Board Members that knows the distributed key-fragments.

After the algorithm pro_conduct_vote terminates with 'refused', 'approved' or 'timeout' the A2S calculates the detailed result according the respective procedures proc_fail, and proc_succ shown in the following paragraphs. The user I_i will be notified accordingly in the next step below (refer to Step 6), and the vote will be over.

If the result of the vote is 'refused', then the A2S has to notify the user (see Step 6) on one hand, and it has to decide on potential 'punishment' of the requester on the other hand. This latter is necessary against unjust voting requests, that is, harassment and Denial-of-Service attacks. Any Board Member who dissents the vote my indicate that further punitive actions are suggested against the requester.

Punishments are stored for each user i in the vector element $PunishLevels[i]$ and calculated within the procedure proc_fail as detailed below. Furthermore, there is a community specific constant *punishIncrement*. If the result of the vote is 'refused' then the punishment level of user i, that is, $PunishLevels[i]$ will be increased by the number of punishment suggestions times the increment:

```
PROCEDURE proc_fail:
    FOREACH $Decision_{k,V_l} \in \mathcal{D}_{V_L}[k]$ DO
        IF ($Decision_{k,V_l}$ =='No, with punishment')
        THEN $punishments_{V_L} := punishments_{V_L} + 1$;
        END_IF
    END_FOREACH
    IF ($terminate_{V_L}$ =='refused')
    THEN $PunishLevels[i] := Punishments V_L \cdot PunishIncrement$
    END_IF
END_PROCEDURE
```

If the outcome is 'approved' then it examines the set $\mathcal{D}_{V_l}$ of received votes to identify an appropriate coalition $\mathcal{C} \in \mathcal{D}_{V_l}$ of at-least T Board Members. The number of approving votes should be bigger than the T threshold of the cryptosystem ($|\mathcal{D}_{V_l}| > T$) in order to make it possible to find a suitable coalition.

The coalition $\mathcal{C}$ is defined as a group of Board Members $BM_i \in \mathcal{C}$ that contains at least T BM-s ($|\mathcal{C}| > T$), such that for each selected BM-s there is a partial decrypt received ($\exists PD_i \in \mathcal{D}_{V_l} | \forall BM_i \in \mathcal{C}$).

After selecting the suitable coalition $\mathcal{C}$ among the approvers, the A2S combines the partial decrypts $\forall_i PD_i \in \mathcal{C}$ to get the decrypted identity map

$$IdMap(U, I_0) = D\left[K^{Pr}_{\forall i \in \mathcal{C}, A2C}, ESIdMap(U, I_0)\right].$$

After this the exection continues at Step 6 of this protocol.

```
PROCEDURE proc_succ:
    $\mathcal{C} := \emptyset$
    FOREACH $Decision_{k,V_l} \in \mathcal{D}_{V_L}[k]$ DO
        IF ($Decision_{k,V_l}$ =='Yes')
        THEN $\mathcal{C} := \mathcal{C} \cup BM_k$;
        END_IF
    END_FOREACH
    IF ($terminate_{V_L}$ =='approved'
    AND $|\mathcal{C}|$ >threshold)
    THEN $IdMap(U, I_0) = D[K^{Pr}_{\forall i \in C, A2C}, ESIdMap(U, I_0)$
    END_IF
END_PROCEDURE
```

Step 5: User I_i can optionally check the status of the voting any time by sending the message Check Vote Request to TTP_1:

$$E\left[K^{Pu}_{TTP_1}, S_{K^{Pr}_{I_i}}[V_l\}]\right]$$

where

V_l is the identifier of the vote.

This step is optional, its sole purpose is to provide means for the user to asynchronously intiate the sending of the message of type *VoteStatus* from the A2S during an ongoing vote (i.e. polling for status of vote).

Step 6: The A2SOC Server TTP_1 notifies user I_i about the status of a previously initiated vote with the message: Vote Status:

$$E\left[K^{Pu}_{I_i}, \{I_j, Cause, V_l, Result(, SIdMap(I_0))\}\right].$$

where

I_j is the accused identity

V_l is the identifier of the vote

Result contains information about of the result or, progress of the vote, that is, the value of the variable $terminate_{V_L}$ corresponding to vote V_l, as described at Step 4.

$SIdMap(I_0)$ is an optional part in the message, it contains the revealed real identity behind the accused I_j of the final outcome of the vote is ’approved’.

This message is sent on the following cases:

(a) The user initiated it by sending a Vote Status query as described in Step 5

(b) The vote has been timeout without enough responses to collect

(c) The vote successfully concluded with ’refused’

(d) The vote successfully concluded with ’approved’. In this latter case it also contains the $SIdMap(I_0)$ information element

End The voting protocol ends here

Protocol 5 User $\mathrm{I_i}$ requests to check the registred data about himself/herself. The Board Members judge the case.

Refer to the numbered arrows in Fig. 5 for the respective steps.

This protocol is almost identical to Protocol 4 The user requests a vote, where the object of the voting is his own base virtual identity (I_0). The user has to state that the ’Cause’ is self-checking. There should be some published common understanding within the community (more precisely among the Board Members) about the fair use policy of this protocol. Obviously, on one hand the self-voting request should be honored and not penalized (this is a crucial factor for the user in order to trust the system). On the other hand, this is clearly a way of Denial-of-Service type of attack if a malicious user floods the board members with self-voting requests.

Upon successful reception the user can check his/her own signature on the $SIdMap(I_0)$.

References

[1] International Standard ISO 7498-1984 (E): Information Technology—Open Systems Interconnection—Reference Model—Part 1: Basic Reference Model.

[2] The SyllabNet Project, http://syllabnet.tmit.bme.hu.

[3] P.F. Syverson, D.M. Goldschlag, M.G. Reed, Anonymous connections and onion routing, in: Proceedings of the IEEE Symposium on Security and Privacy, Oakland, California, 1997.

[4] K. Bennett, C. Grothoff, T. Horozov, I. Patrascu, T. Stef, Gnunet—a truly anonymous networking infrastructure, http://citeseer.nj.nec.com/502472.html.

[5] D. Boneh, M. Franklin, Efficient generation of shared RSA keys, in: Proceedings of the Crypto’97, Lecture Notes in Computer Science, vol. 1233, 1997, pp. 425–439.

[6] Dan Boneh, Intrusion Tolerance via Threshold Cryptography (ITTC) project, http://crypto.stanford.edu/dabo/ITTC/, 1999.

[7] S. Buchegger, J.Y.L. Boudec, A robust reputation system for p2p and mobile ad-hoc networks, in: Proceedings of the Second Workshop on the Economics of Peer-to-Peer Systems, 2004.

[8] L. Buttyan, J. Hubaux, Accountable anonymous access to services in mobile communication systems, in: 18th IEEE Symposium on Reliable Distributed Systems (SRDS’99), Workshop on Electronic Commerce (WELCOM’99), Lausanne, Switzerland, 18–21st October 1999.

[9] D.L. Chaum, D.L. Chaum, Untraceable electronic mail, return addresses and digital pseudonyms, Communications of ACM 24 (2) (1981).

[10] J. Claessens, B. Preneel, J. Vandewalle, Anonymity controlled electronic payment systems, in: Proceedings of the 20th Symposium on Information Theory in the Benelux, 1999.

[11] Ian Clarke, Oskar Sandberg, Brandon Wiley, Theodore.W. Hong, Freenet: A distributed anonymous information storage and retrieval system, vol. 2009, Springer, 2001. p. 46+.

[12] C. Farkas, G. Ziegler, A. Meretei, A. Lőrincz, Anonymity and accountability in self-organizing electronic communities, in: Pierangela Samarati (Ed.), Proceedings of the Workshop on Privacy in the Electronic Society, Washington, DC, USA, November 21, ACM SIGSAC, ACM. Paper no. 136, 2002, p. 10.

[13] A. Fernandes, E. Kotsovinos, Sven Őstring, Boris Dragovic, Pinocchio: Incentives for Honest Participation in Global-scale Distributed Trust Management. http://citeseer.ist.psu.edu/fernandes03pinocchio.html.

[14] E. Gabber, P.B. Gibbons, D.M. Kristol, Y. Matias, A. Mayer, Consistent yet anonymous web access with lpwa, Commun. ACM (1999).

[15] A. Garcia-Martinez, J. Chuang, A Cryptographic Reputation Scheme for Peer-to-peer Networks. http://citeseer.ist.psu.edu/garcia-martinez02cryptographic.html.

[16] I. Goldberg, D. Wagner, E. Brewer, Privacy-enhancing technologies for the internet, in: Proceedings of the 42nd IEEE Spring COMPCON, 1997.

[17] Gerard J. Holzmann, Design and Validation of Computer Protocols, Prentice-Hall, 1991.

[18] S.D. Kamvar, M.T. Schlosser, H. Garcia-Molina, The eigentrust algorithm for reputation management in p2p networks, in: Proceedings of the Twelfth International World Wide Web Conference, 2003.

[19] D. Kugler, H. Vogt, Off-line payments with auditable tracing, in: Financial Cryptography, Lecture Notes in Computer Science, Springer, 2002.

[20] P. MacKenzie, J. Sorensen, Anonymous investing: Hiding the identities of stockholders, in: Proceedings of Financial Crypto’99, Lecture Notes in Computer Science, vol. 1648, 1999.

[21] Michael Malkin, Thomas Wu, Dan Boneh, Experimenting with shared generation of RSA keys, in: Proceedings of the Internet Society’s 1999

Symposium on Network and Distributed System Security (SNDSS), pp. 43–56.

[22] Aviel D. Rubin, Marc Waldman, Lorrie Faith Cranor, Publius: a robust, tamper-evident, censorship-resistant, web publishing system, in: Proceedings of the 9th USENIX Security Symposium, August 2000, pp. 59–72.

[23] D. Martin, A. Schulman, Deanonymizing Users of the Safeweb Anonymizing Service, Nov. 2002. .

[24] Napster, http://www.napster.com/about_us.html, 2002.

[25] Massachusets Institute of Technology, The OpenCourseWare project, http://ocw.mit.edu/index.html.

[26] Zsolt Palotai, Sándor Mandusitz, András Lőrincz, Distributed mining of the internet for novel news: evolutionary community of news foragers, in: Proceedings of the International Joint Conference on Neural Networks, July 25–29, in press.

[27] M.K. Reiter, D. Rubin, Crowds: anonymity for web transactions, ACM Transactions on Information and System Security 1 (1) (1998) 66–92.

[28] A.R. Rezai, M. Phillips, K.B. Baker, A.D. Sharan, J. Nyenhuis, J. Tkach, J. Henderson, F.G. Shellock, Neurostimulation system used for deep brain stimulation (DBS): MR safety issues and implications of failing to follow safety recommendations, Invest. Radiol. 39 (5) (2004) 300–303.

[29] C. Shields, B.N. Levine, A protocol for anonymous communication over the internet, in: Proceedings of ACM Conference on Computer and Communications Security, 2000.

[30] SourceForge.Net, the open source software development web site. http://www.sourceforge.net.

[31] S.G. Stubblebine, P.l F. Syverson, Authentic attributes with fine-grained anonymity protection, Lect. Notes Comput. Sci. 1962 (2001).

[32] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, MIT Press, 1998.

[33] The World Wide Web Consortium, Web Services activity. http://www.w3.org/2002/ws/.

[34] The World Wide Web Consortium, XML Protocol Working Group, http://www.w3.org/2000/xp/Group/.

[35] Thomas Wu, Mike Malkin, Dan Boneh, Building intrusion tolerant applications, in: Proceedings of the 8th USENIX Security Symposium, 1999, pp. 79–91.