An Infrastructure for Community Signatures and Micro-Agreements

Architecture, Android prototype implementation, and usage examples

Mitja Vardjan and Jan Porekar
Research Department
SETCCE
Ljubljana, Slovenia
{mitja.vardjan, jan.porekar}@setcce.si

***Abstract*—Digital signatures are widely used for non-repudiation and other purposes. In various cases, there is a group of two or more parties that have to agree on a common set of data and digitally sign it in order to provide the other party or parties a proof of non-repudiation. A simple and scalable infrastructure for community signatures or groups of individual party signatures is described. It allows third party applications to simultaneously digitally sign arbitrary XML documents by any number of entities, for any purpose, using high level interfaces, not having to deal with digital signatures themselves. A dedicated backend server dynamically merges received documents and signatures from all parties. When a sufficient number of entities have signed the document, a signal is triggered to announce the document finalization. Despite the simple overall design, handling security issues and user control at appropriate spots are crucial for any business application.**

Keywords-community; agreement; digital signature; mobile environment

I. INTRODUCTION

One of the most used aspects of digital signatures is non-repudiation. When electronic documents are digitally signed by one or more parties, the signatures can be used to verify the document integrity and, more importantly for this work, to prove that the parties have agreed on the document and stand behind it.

In many cases, only one valid digital signature is provided with the document at any time. The goal in such cases is usually to ensure document integrity, or to provide non-repudiation of a single entity. In case of signing contracts, agreements, and similar documents, two or more entities are to provide non-repudiation to each other. Some of these entities can be owners of internet connected pervasive services or internet connected objects. The signing process and distribution of digital signatures can easily get overly complex or even infeasible for the entities, especially if their number is large or arbitrary. This can be remedied in a business process where the document format and the order, in which it is signed by the entities, are determined by the application or protocol, such as the negotiation presented in [1].

The infrastructural service described here allows for groups and communities to reach legally binding agreements in an ad-hoc manner. Third party services can offload any documents that need to be agreed over group of participants or even whole communities. These documents range from service level agreements, meeting minutes to non-disclosure agreements or even business contracts that may have rich content embedded. The work in this paper is a continuation and complement of [1].

The functionality reuses the concepts of digital identities, certificates and digital signatures. Documents are structured with Extensible Markup Language (XML) and agreements are signed using XMLDSig [5]. Both architecture and implementation target mobile and pervasive environments by providing an asynchronous and scalable solution that limits bandwidth usage, avoids unnecessary communication, and enables all user devices to be used from arbitrary local networks that are connected to the Internet intermittently and through firewalls.

Existing group signature and concurrent signature [13] solutions, especially the improved and multi-party versions [14][15][16] fit various purposes, but may not be most suitable for use by third party application developers who prefer well known solutions and expect fast and easy integration. Some existing designs for group signature use their own custom signatures and require additional solution-specific steps to sign the data and to verify a signature [7][8][9], or allow only community members to sign [8], which is not suitable for ad-hoc communities. Such requirements can put additional burden to both implementation of third party applications that use the signature infrastructure, and to community administration. In terms of efficiency and optimization, additional network interactions are required, e.g., when the keystone is released in case of concurrent signatures. Moreover, both group signatures and concurrent signatures diverge even further from the traditional way of signing paper documents, still widely used. While the concept of fair exchange of signatures and decreased verification time are highly beneficial in some cases, the additional differences may present an obstacle for adoption of the solution. For example, if the identity of the first signers is not known to all, subsequent signers may be less likely to sign the document. This may be because in case of known identities, they trust the party or parties who already signed the document, or simply because they have a proof that the party with known identity has already signed the document, e.g., when negotiating a service-level agreement [1]. On the other hand,

 ISBN: 978-1-61208-324-7

for communities where all members are equal and do not know or trust each other, the concurrent signatures are better in terms of fairness and non-exposure, but they are not used in the presented work.

The next chapter describes the initial document creation and its distribution to other users. The chapter is followed by descriptions of document signing and finalization procedure. Afterwards, various privacy and security aspects of the whole process are explained. The paper ends with usage examples to illustrate a few implemented and suggested services that are using the presented community signature infrastructure.

II. Document Creation and Distribution

Initially, an XML document with arbitrary schema and contents is created either by one party or in a collaborative manner by multiple members of a community. The document may hold a service level agreement, meeting minutes, non-disclosure agreement, or even business contracts that may have rich content embedded such as images, video or voice recording.

Regardless of what the document represents, the community members are expected to review it once it is finalized and confirm they agree with it. Their consent is formally expressed with their digital signature, appended to the document as a detached XMLDSig [5]. Depending on the application, a member may choose to sign the whole document, only some of its parts, or nothing and leave the document intact.

The initial document is distributed to the intended signers or members by uploading it to a dedicated Representational State Transfer (REST) server in a single HTTP PUT request. The REST server stores the document under the name, supplied by the client as resource name within the URL. The name is generated as a random string of a fixed length. The concept of resource name is similar to universally unique identifier (UUID) [2] but the name is shorter because it is checked for uniqueness at the server level when the resource is initially uploaded. Unless a resource with same name already exists on the server and the HTTP PUT request has to be repeated with a new name, the upload is a single step operation. The request includes the owner's serialized X.509 certificate [4] as part of the URL. This certificate is stored by the server for later authorization to access the document by others. It is never used to sign the document, unless the user chooses to do so. Therefore, it could be anonymous or generated ad-hoc by the initial document uploader. Its corresponding private key is used to sign the resource name. This signature is not supplied with the initial upload, but with another URL, generated by the community signature infrastructure.

Whenever a document is downloaded or a new version of existing document is uploaded, digital signature of resource name is passed as a URL parameter. The same URL is used for downloading and updating documents. The URL of the uploaded document is distributed to the members as an invitation for them to agree with and digitally sign the document.

Figure 1. Document creation and distribution.

The members list is usually application specific and the URL distribution is handled in the background by an app that is using the community signature infrastructure. If this is not the case, the URL and the document can still be accessed manually within the signature infrastructure itself (Figure 4). This lightweight and easy to implement process is suitable for the uploader device and signer devices, which are usually smart phones or tablet PCs. When a user chooses to reject or ignore the invitation to sign the document before he even reads it, bandwidth usage is negligible.

III. Micro-Agreements and Document Finalization

In the process of agreeing, the canonical form [6] of agreement document is digitally signed with a private key that is stored in participant's smart phone's secure storage. The meeting participants do not need to sign the document immediately but can postpone the signing of the agreement.

After the agreement is signed by a participant it is uploaded back to the community sign service using the same URL that has been used to download it. The reasoning is that for community signatures, anyone who is authorized to download the document should be able to upload the signed version as well. If this is not the case, the concept of authorization signature in the URL can be easily expanded to include option to allow download only or both upload and download. An example solution is to sign document resource name, suffixed with an appropriate parameter, known to the service. The community sign service at the REST server verifies whether the digital signature is valid and whether the content of the agreement has not been modified in any way.

The community signature functionality allows third party services that are using it to specify the minimal number of community members that need to agree in either relative terms such as percentage of community or fixed threshold numbers. Every time the document with a new signature is

 ISBN: 978-1-61208-324-7

uploaded to the community signature service backend node, this micro-agreement is merged into the main document stored on the server. Due to the nature of detached XMLDSig, the merges originating from various signers can be performed in any given order and the signers will experience a convenient and seemingly parallel signing procedure.

The resulting document at any moment contains signatures from all parties that have signed the document and sent it back to the server so far. When number of parties that signed the document exceeds the given threshold, the community signature service backend server signals completion and participants can now download the final agreement, which now contains at least the required number of signatures (Figure 3) and represents a common and a legally valid agreement. Depending on the implementation, the document finalization can be signaled to the original document creator, e.g., meeting organizer, who can first inspect the document and the signers and then choose to signal document finalization to the other selected parties. At any point, the parties can see the current status of any document they have signed, or were invited to sign. Figure 4 shows the status of a document in the process of being signed (left) and the status of that same document at a later time, when one more party has signed it and the number of signers reached the required threshold (right). If concurrent signatures were used, full status with signers' identities could be displayed only after the keystone is released.

Figure 2. A community member receives invitation to sign a document.

Unlike a group signature [7] where multiple individual signatures are replaced with a single group signature, individual signatures are preserved and any party can verify individual signatures using a standard verification procedure. Due to the nature of XMLDSig, any party can also get the list of all signers solely from the document.

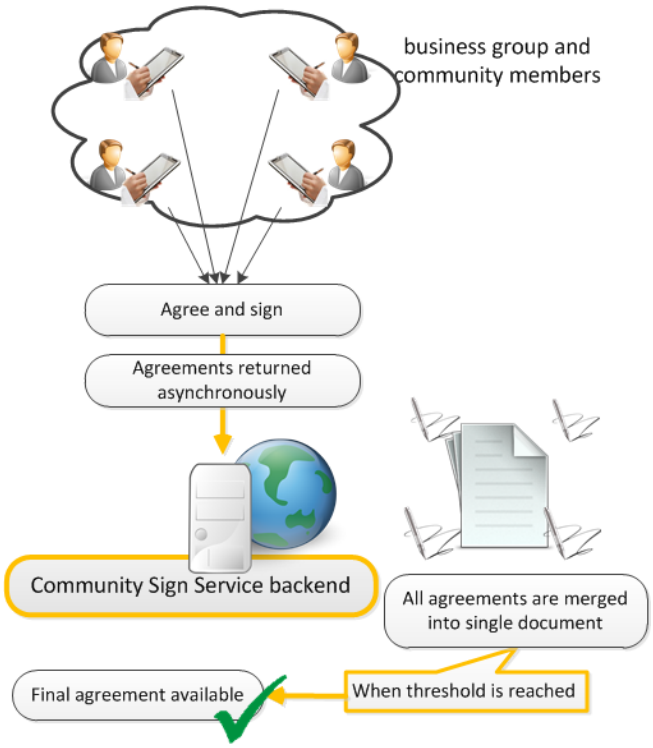

Figure 3. Community signature and document finalization.

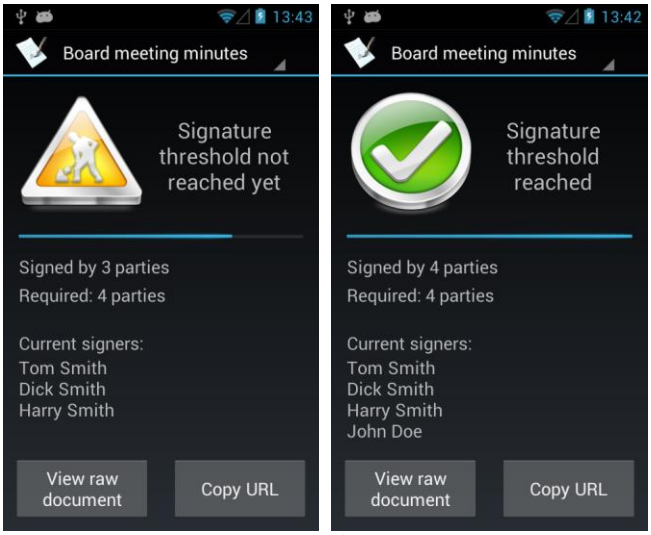

Figure 4. Viewing current status of the document signing process.

The downside of not using the concept of group signature [7] is that processing power and time to verify all signatures increase with number of signatures in the final document. As the increase is only linear, this is usually not problematic in terms of scalability. If all parties can be forced to use a specific key-pair type, then verification of multiple signatures could be sped up [10][11], although care must be taken because some such solutions have issues [12].

IV. PRIVACY AND SECURITY ASPECTS

The two main groups of information that could be treated as sensitive are the document contents and the list of entities

 ISBN: 978-1-61208-324-7

who have signed the document. The document itself has to be made fully available to all entities that are given the option to sign it. Same applies to the list of signers because they all receive the final document in the end, leaving no alternative to ultimately trusting the entities not to disclose any sensitive information they receive.

Various notifications about document finalization do not carry any personal or document data and usually do not need to be secured. A few other points where it makes sense to take security into account are described below.

A. Document Distribution

There are established protocols to encrypt the network traffic from eavesdropping. However, a custom solution described in Chapter II is used as a secure and convenient method to authorize the clients to download and upload the document. With the proposed solution, the clients (entities) are given only one URL that already contains all necessary tokens (Figure 5). As the digital signature of requested resource is part of the URL, the certificate owner can easily disable access by removing the public part of his certificate at the service backend (Figure 1 and Figure 5).

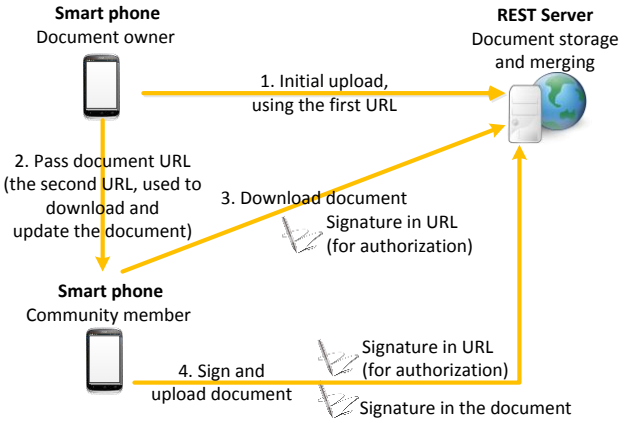

Figure 5. The two roles of signatures.

Alternatively, when the certificate is revoked, access is automatically disabled, provided that the service backend implementation does check certificate revocation lists.

In any case, the number of network operations from mobile devices is limited and the authorization is integrated into the simple and widely used HTTP methods, so third party developers are not required to implement any authorization procedures.

B. Storage of Certificates on Android

With any digital signature based system, it is vital to protect the private keys from unauthorized use. The prototype has been implemented for Android where a secure storage is provided by the operating system. This storage is used for storing user's certificates and private keys. It is accessed in two significantly different ways, depending on Android version. For Android versions up to 4.2.2, the API is not public and the operating system grants requests to the storage based on the requestor process ID. The concept is described in [1]. For Android versions 4.3 and newer, the access to the secure storage is possible only through the new and official API for storing and accessing certificates and keys. To support all versions, the app implements both strategies and chooses the appropriate one dynamically.

C. Using the Securely Stored Private Keys on Android

To sign an arbitrary XML document, our prototype app can be used directly. However, in most cases it is to be used by other apps that parse the document and show the user a human readable and application specific document representation before the user authorizes signing. The problem is to access the user's private keys, which are not available to third party apps and not even to the operating system. As a solution, the third party app can simply invoke in the background our prototype app with access to private keys to sign the given document.

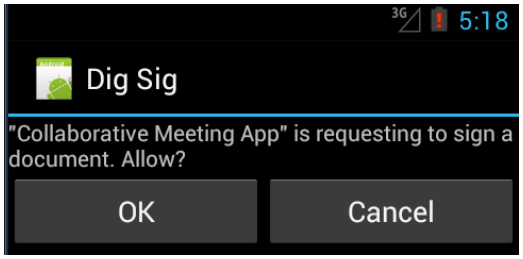

Figure 6. Third party app requests to sign a document have to be explicitly confirmed by the user.

It is vital for the prototype app to show the user which app is trying to sign the document in the background, to prompt the user to authorize signing (Figure 6) and choose the identity to use (if multiple certificates are stored). The key itself is never exposed to third party apps, so only the data explicitly approved by the user are signed.

V. USAGE EXAMPLES

Examples of usage are described below. The community micro-agreements are suited to also be used by applications and services that enable governance tools to communities.

A. Capturing Meeting Minutes

Community micro-agreements allow business communities to capture meeting minutes and other meeting agreements in a legally valid and binding manner. The meeting organizer can choose whether the consensus is reached among only participants that are physically present during the meeting or the whole community.

Existing community signature prototype implementation has been used by an example app to capture meeting minutes. After users register to the meeting through this app, they can actively participate in the meeting. Their input is recorded by their Android devices and sent to a central Android device, which has the role of the document owner. When the meeting is finalized on that central device, the minutes are uploaded to the document storage server (Figure 7) and its URL is distributed to meeting participants. The REST servers which handle distribution of document URLs and receive notifications about document finalization (Figure

 ISBN: 978-1-61208-324-7

7) are application specific, i.e., implemented as part of the meeting minutes software, not the general community signature software. Google Cloud Messaging (GCM) is used to relay the messages to Android phones of users who are to sign the minutes. At an earlier point, the meeting software automatically registers Android devices of community members with GCM to receive these messages. GCM is used by the meeting software as a convenient way to push small messages to Android devices, connected to the Internet through firewalls, with variable network addresses, etc. The community signature infrastructure does not require using neither GCM, nor the additional REST server to distribute document URL, but only to distribute the URL to community members. Therefore, any alternative distribution of the URL is valid. For example, the app on the central device embeds the URL into a Quick Response Code (QR code) and the physically present members can scan it. Again, this is only an alternative way of URL distribution and the primary way is application specific automatic distribution in the background, in this case through GCM. Arrows in Figure 7 indicate information flow for the implementation with GCM, starting with document upload by the document owner to the first REST server shown at the top center.

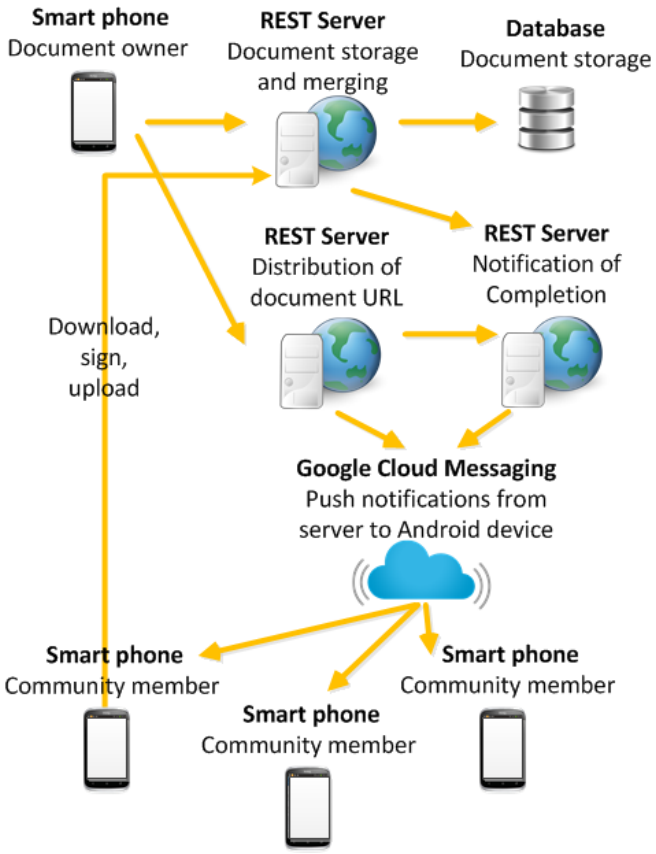

Figure 7. Process and information flow between devices in a chosen implementation for capturing meeting minutes.

Regardless of the implementation, the signatures are always in standard XMLDSig form, as in Figure 8. In the figure, XML nodes with signature and certificate values are collapsed but the highlighted text shows the signatures refer to the whole document, i.e., the whole meeting minutes. In case a participant agreed only with part of the document, his signature would refer to the relevant part only, provided that the application specific implementation allowed signing only a part of the document.

```
<meeting Id="Board001">
  <conversation>
    <user>
      <id>92001</id>
      <username>John Doe</username>
    </user>
    <minute>Hello and welcome to the meeting!</minute>
  </conversation>
  <conversation>
    <user>
      <id>97001</id>
      <username>David Smith</username>
    </user>
    <minute>Hi John. First things first: we need more drinks.</minute>
  </conversation>
  <conversation>
    <user>
      <id>92001</id>
      <username>John Doe</username>
    </user>
    <minute>Agreed.</minute>
  </conversation>
</meeting>

<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#" Id="Signature-a8
  <ds:SignedInfo>
    <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xr
    <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-s
    <ds:Reference URI="#Board001">
  </ds:SignedInfo>
  <ds:SignatureValue>
  <ds:KeyInfo>
</ds:Signature>

<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#" Id="Signature-5e
  <ds:SignedInfo>
    <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xr
    <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-s
    <ds:Reference URI="#Board001">
  </ds:SignedInfo>
  <ds:SignatureValue>
  <ds:KeyInfo>
</ds:Signature>
```

Figure 8. An example of meeting minutes signed by two parties.

In this example, the omnipresent issue of identity mapping is evident. Mapping between various identity types is essential for any legally binging document. Typical identity types relevant for community signatures are:

- Possible identities in the signed document. Figure 8 shows a case where identities are explicitly listed in the signed document. This is not always the case. The document could include only impersonal statements.
- Identities in encoded X.509 certificates, contained in the collapsed “ds:KeyInfo” nodes in Figure 8.
- Identities of the community members who signed the document.

Clearly, any implementation should check:

- mapping between the certificate filed values, e.g., common name, and the document identities, if any,
- certificate validity and whether it is issued by a trusted authority,
- mapping between certificate and real entity, e.g., by checking the entity listed in the certificate is actually a member of the community that is supposed to sign the document.

 ISBN: 978-1-61208-324-7

For large communities, this can be far from trivial, as the certificate identities can be ambiguous and also because a single entity can be listed under different names in the certificate and community members list.

B. Crowd Tasking

A service called Crowd Tasking has been developed to enable community members to create tasks (an example is shown in Figure 9), propose solutions, post comments and solve tasks. These tasks usually involve some physical presence of people or physical work, which makes it inconvenient or impossible to post either the solution, or proof of the task solution to the service or to the Internet.

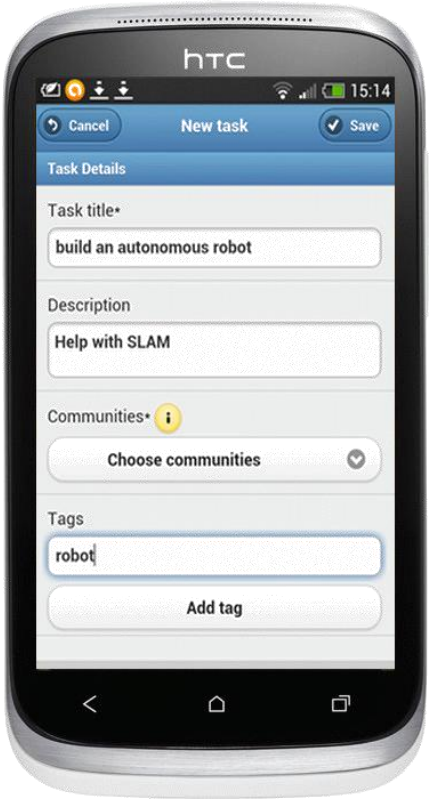

Figure 9. Crowd Tasking Service.

The service will integrate with the community signature infrastructure to enable task members to sign the agreements about the work to be done by each of them and to enable task creators to confirm the task completion by additional signature. As with any other usage of community signature, the interactions of third party service with community signature infrastructure and the document signing happen in the background, except prompting the user to confirm signing.

C. Service Sharing Within a Community

The policy negotiation described in [1] could be extended by integrating with community signatures and micro agreements presented here. A service provider would negotiate a service level agreement (SLA) with a community instead of only a single service consumer. The community members would decide if a particular SLA is compatible with community's internal rules and sign the SLA so the service could be shared within the community.

VI. Conclusion and Further Work

An infrastructure and prototype implementation of community signatures and micro-agreements has been presented, followed by usage examples. The design uses digital signatures to sign XML documents, which can serve as legally binding agreements. It is based on REST servers, a database or other storage system, and Android devices. The simple, scalable and generic main concepts allow for fast integration of various third party services with it. Network communication is optimized for mobile devices with limited and intermittent bandwidth, but at least occasionally working network connection is still required for all devices. Compared to concurrent signatures, the presented approach requires slightly less network interactions, is more similar to traditional signing process of paper documents, and as such does not exchange signatures between parties in a fair manner, which has both advantages and disadvantages. Ideally, the solution could offer both signature options to cover additional possible scenarios. Other services are planned to use the implemented community signature infrastructure in an application specific manner.

Acknowledgment

Authors would like to thank the SOCIETIES project [3] consortium and EC for sponsoring the project.

References

[1] M. Vardjan, M. Pavleski, and J. Porekar, "Securing Policy Negotiation for Socio-Pervasive Business Microinteractions", SECURWARE 2012: The Sixth International Conference on Emerging Security Information, Systems and Technologies, ISBN: 978-1-61208-209-7, Aug. 2012, pp. 142-147.

[2] ITU Recommendation X.667 (09/04), http://www.itu.int/rec/T-REC-X.667-200409-S/en [retrieved November, 2013].

[3] Self Orchestrating CommunIty ambiEnT IntelligEnce Spaces (SOCIETIES), EU FP7 project, Information and Communication Technologies, Grant Agreement Number 257493.

[4] X.509 standard recommendation, http://www.itu.int/rec/T-REC-X.509/en [retrieved November, 2013].

[5] XML-DSig, XML Signature Syntax and Processing, 2nd Edition http://www.w3.org/TR/xmldsig-core/, [retrieved April, 2012].

[6] Canonical XML 1.1, W3C recommendation, http://www.w3.org/TR/xml-c14n11/, [retrieved April, 2012].

[7] L. Harn, "Group-oriented (t, n) threshold digital signature scheme and digital multisignature", IEE Proceedings - Computers and Digital Techniques, Volume 141, Issue 5, Sep. 1994, p. 307-313, DOI: 10.1049/ip-cdt:19941293.

[8] C. M. Hsu, S. H. Twu, and H. M. Chao, "A Group Digital Signature Technique for Authentication", IEEE 37th Annual 2003 International Carnahan Conference on Security Technology, ISBN: 0-7803-7882-2, Oct. 2003, pp. 253 – 256.

[9] L. Harn, C. H. Lin, and C. W. Hu, "Contract Signatures in E-Commerce Applications", International Conference on Broadband, Wireless Computing, Communication and Applications, Nov. 2010, pp. 384-388, DOI: 10.1109/BWCCA.2010.101.

 ISBN: 978-1-61208-324-7

[10] C. Li, M. Hwang, and S. Chen, "A batch verifying and detecting the illegal signatures", International Journal of Innovative Computing, Information and Control, Dec. 2010, pp. 5311-5320.

[11] A. Atanasiu, "A New Batch Verifying Scheme for Identifying Illegal Signatures", Journal of Computer Science and Technology, Vol. 28, Issue 1, Jan. 2013, pp. 144-151.

[12] M. S. Hwang and C. C. Lee, "Research Issues and Challenges for Multiple Digital Signatures", International Journal of Network Security, Vol.1, No.1, Jul. 2005, pp.1-7.

[13] L. Chen, C. Kudla, and K. G. Paterson, "Concurrent Signatures", Advances in cryptology - EUROCRYPT 2004, Vol. 3027, May 2004, pp. 287-305.

[14] X. Tan, Q. Huang, and D. S. Wong, "Concurrent Signature without Random Oracles", IACR Cryptology ePrint Archive, 2012.

[15] C. Shieh, H. Lin, and S. Yen, "Fair multi-party concurrent signatures", Proc. of 18th Cryptology and Information Security Conference, 2008, pp. 108-118.

[16] J. Xushuai, Z. Zhou, W. Qin, Q. Jiang, and N. Zhou, "Multi-Party Concurrent Signature Scheme Based on Designated Verifiers", Journal of Computers, Vol. 8, No. 11, Nov. 2013, pp. 2823-2830.

 ISBN: 978-1-61208-324-7