Development, Testing, and End-User Evaluation of Pervasive Community Signatures and Micro-Agreements Infrastructure

Architecture, Android Implementation, Performance Tests, Usage Examples, and User Evaluation

Mitja Vardjan, Helena Halas, Simon Jureša, and Jan Porekar
Research Department
SETCCE
Ljubljana, Slovenia
{mitja.vardjan, helena.halas, simon.juresa, 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. In the paper we present the performance and robustness tests of the current prototypal community signatures infrastructure. We also present the results of end user trials and measure the quality of experience perceived by end-users that are using a pervasive application that is interacting with the community signatures infrastructure.**

Keywords-community; agreement; digital signature; mobile environment, pervasive, e-business, infrastructure

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.

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] and [2].

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 [3]. 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 [4] solutions, especially the improved and multi-party versions [5][6][7] 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 [8][9], or allow only community members to sign [10], which is not suitable for communities that are formed in an ad-hoc fashion. 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 be willing 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 (see [2] for examples). On the other hand, 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 section describes the initial document creation and its distribution to other users. The section is followed by descriptions of document signing and finalization procedure. Afterwards, various privacy and security aspects of the whole process are explained. Next, the implementation of community signatures infrastructure is presented and afterwards thorough tests of both performance and robustness to give an overview of its current capabilities and features. This section is followed with usage examples to illustrate a few implemented and suggested services that are using the presented community signature infrastructure. Special focus is dedicated to the "Pervasive Meeting Minutes" application and service that allows for unobtrusive capturing of meeting minutes. The application uses and interacts with the community signature infrastructure. At the end of the paper we present the results of the end-user trials focusing on perceived added value of end-users and on how the pervasive application, that is using the community signatures infrastructure, adds to quality of experience.

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 [3]. 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) [11] 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 [12] 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 1). 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 [13] of agreement document is digitally signed with a private key that is stored in participant's smart phone's secure storage (see Figure 2 for more) 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 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.

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

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.

Unlike a group signature [8] 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.

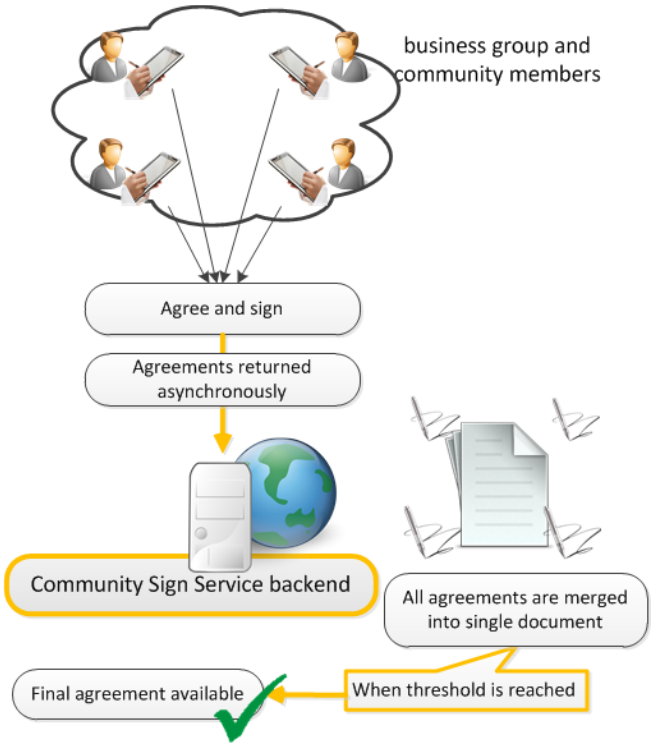

Figure 3. Community signature and document finalization.

The downside of not using the concept of group signature [8] 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 [14][15], although care must be taken because some such solutions have issues [16].

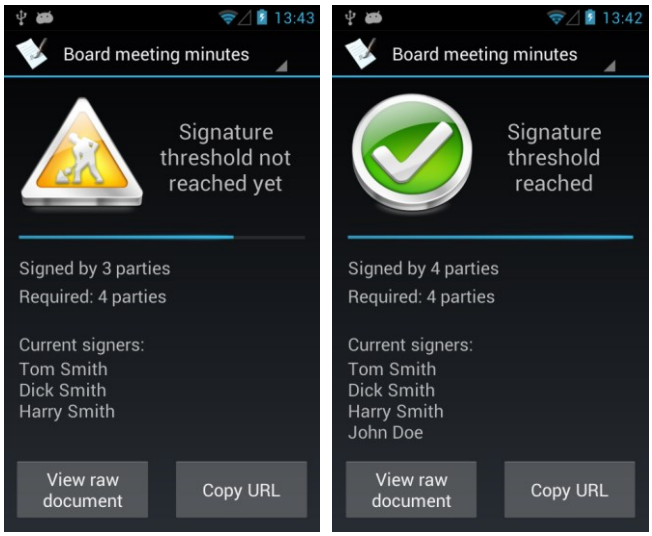

Figure 4. Viewing current status of the document signing process.

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 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

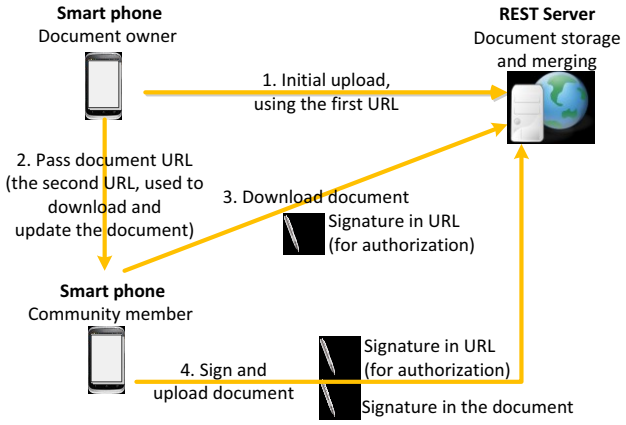

Figure 5. The two roles of signatures.

There are established protocols to encrypt the network traffic from eavesdropping. However, a custom solution described in Section 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).

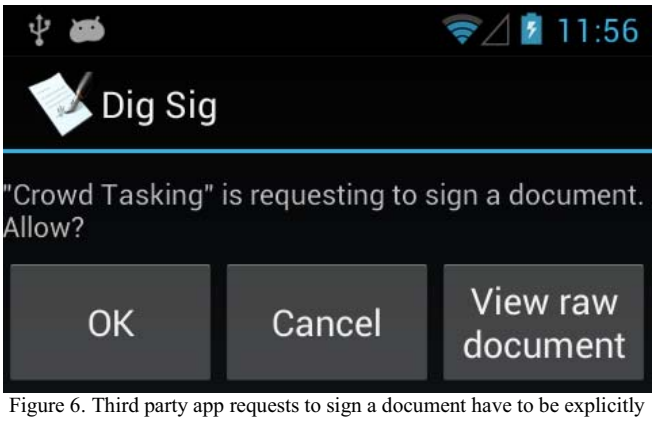

Figure 6. Third party app requests to sign a document have to be explicitly confirmed by the user.

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 [2]. 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.

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. PERFORMANCE TESTS

Commonly, the primary reported problems after field release of software are not crashes or incorrect responses, but poor performance or inability to achieve required system throughput [17]. It is not uncommon that although the software has gone through extensive functionality testing, it has never been really tested to assess its expected performance. Such neglect of planning for performance issues often leads to performance problems once the software is released to the field, which in turn often significantly impacts the project's ultimate success or failure. The presented implementation has been continuously tested for functionality issues as its parts were developed. Once it reached sufficient level of completion, but still at early prototype phase, it has been tested also for performance issues that could possibly arise from the following two reasons:

- Multiple users are signing the same document and due to large number of users, thus, large number of signatures being appended and verified, the overall signing process takes more time to complete.
- Multiple users are signing the same document simultaneously. In addition to the reason above, the server load is increased because multiple HTTP requests and multiple database read/write requests are served in same time interval.

The tests were divided into two groups accordingly. Aspects of performance include latency, throughput, scalability, and reliability. In terms of reliability, the prototype performed successfully every time in each test. No errors were observed, except the out-of-memory error described in subsection C "Verifying signatures on the Android client". In some of real world tests and demonstrations, unreliable network and outbound port filtering used on some public networks caused additional delays and failures. While network connectivity related delays are unavoidable and the solution is designed and implemented to be used behind firewalls that filter out all incoming connections, outbound port filtering can usually be amended by configuring the REST server to listen on a more common port such as 80 or 443, which are usually not outbound-filtered.

With a powerful cloud computing infrastructure, the performance can be greatly improved, of course. However, to assess possibility of using a cheap server solution, the tests were carried out with a few years old everyday Core 2 Duo desktop PC running all necessary server software, including the database. For Android devices, Android software emulator and Google Nexus 10 were used.

The aim of performance tests is to realistically model expected common use cases of the implemented infrastructure under test. On the other hand, the tests were performed in a controlled environment so they could be repeated with same parameters. In the real world, the performance could be greatly affected also by various unavoidable difficulties like intermittent or extremely slow network, overloaded system resources on Android device, caused by other apps or malware, etc. The effect of such parameters is out of scope of these tests.

The following subsections focus on latency, throughput, and scalability aspects of performance.

A. Subsequently Signing a Document

For the first group of tests, an initial document version was uploaded to the REST server. Then, an Android client downloaded it from the REST server, locally signed it, and sent it back to the server. The server merged the signature and notified the client about successful finish of operation. For both the client and server, the real Android implementation was used. The Android client was run in test mode, which meant the user's clicks on appropriate buttons to approve the signing process and select the identity were performed automatically. After the operation finished, the whole operation (except the initial upload of the first document version) was repeated until the document contained 20 signatures. With every additional signature, the document got larger and the XML manipulation took longer. Besides, the signatures are verified at each step and as the number of signatures increases, the overall verification time at each step is increased. This effect is expected to be most profound in case of documents with large content to sign. In the tests, the whole document was being signed, except the existing signatures from previous steps.

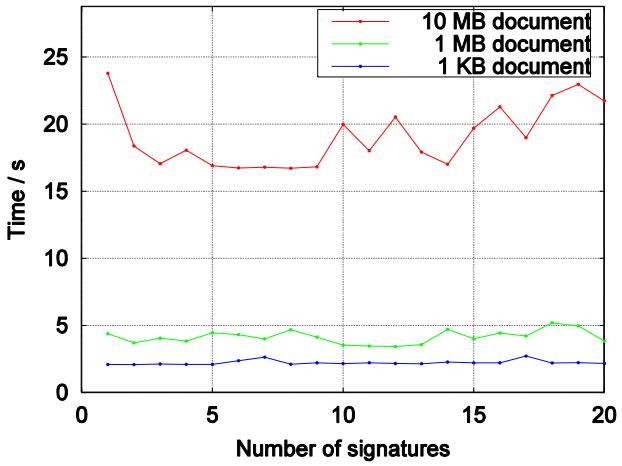

Figure 7. Time of subsequent signing.

Three different document sizes were used (Figure 7). The smallest document (1 KB size) contained minutes of a very short meeting. The larger 1 MB and 10 MB documents had additional text inserted. In both cases, the text was generated by a pseudo random generator, so the document could not be efficiently compressed. This modelled small and XML-embedded binary data that is already efficiently encoded or compressed, e.g., voice recording or other multimedia contents.

As expected and explained above, the time to verify the signatures increases with their number, and this effect is most noticeable with the largest document size.

Another increase in time is for the first signing process. This increase too, is most evident for the largest document size, but also noticeable for the medium size. It is expected that the later signing processes are faster because of data caching on the server and various connections and libraries already initialized during the first signing. Thus, the increased time of first signing procedure (or the decreased time of subsequent signing procedures) is probably unavoidable.

Median times of subsequently signing 1 KB, 1 MB, and 10 MB documents were 2.19, 4.09, and 18.21 seconds, respectively. Despite using a low-end hardware, shorter times were expected. Based on inspection of server logs, some interactions with the database consistently took at least half second and up to one second even for the smallest 1 KB files. Inserting a new XML node with signature and storing the new document also consistently took about one second for small XML documents. This indicates bottlenecks in our prototype that should be optimized first.

B. *Simultaneously Signing a Document*

The other test mimicked simultaneous signing of a document by multiple users. For the most realistic test, multiple Android devices should sign the document within a short time interval. However, a more feasible approach was used where the multiple Android devices or users were modelled by multiple threads running on a single Android emulator. This test is even more focused on the REST server than the previous tests, so the local signing on Android was skipped to avoid unnecessary load of already burdened client. The threads on Android only downloaded a signed document from the server and then uploaded it back. The server would still perform the usual request authorization, signature merging, and document update routines for each request.

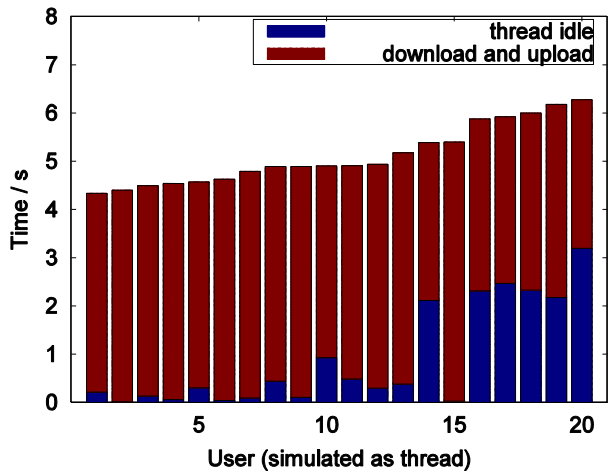

Figure 8. Time of simultaneous signing.

Figure 8 shows the time (from test start) until a thread actually started (blue bars) and the time a thread took to download the document, upload it to the server, wait for the server to perform the usual verification and signature merge, and receive notification about successful completion from the REST server (red bars). To assess primarily the concurrency issue, the 1 KB document was used here to eliminate influence of network bottlenecks and other consequences of large files. Time for simultaneously signing a 1 KB document is represented by the red bars in Figure 8. The median value was 4.37 seconds. The blue bars in Figure 8 mostly increase with thread index and some later threads reach 2 or even 3 seconds. This indicates client overload which possibly lead to over-estimated delays. However, based on server logs, most delays are caused at the server side, so the performance of a real setup would be only slightly better. It should be noted that the variable idle time of threads (blue bars in Figure 8) before they start the download and upload routines more realistically models an actual signing process than a strictly simultaneous start would, because it is unlikely that all users would sign a document at exactly the same time. So some randomness in thread start is necessary for realistic results, provided that at some point, all threads run simultaneously (between 3.2 s and 4.3 s in Figure 8), because it is the simultaneous signing that is being tested.

It should be stressed that the bar chart in Figure 8 is monotonically increasing only because the threads in Figure 8 are ordered by the time they received notification about successful completion from the server, not by the time they started.

Obviously, the results could be greatly improved not only by software optimization, but also by using more capable server-side hardware.

C. *Verifying signatures on the Android client*

After the document signing process is finalized and a user downloads the final document version, his Android client verifies all signatures in the document. The benefit is that the user does not have to ultimately trust the server, but this additional verification takes additional processing time.

Time of document signatures verification has been measured on Samsung Nexus 10 device with two different documents: a document where 1 KB of contents has been signed and a document where 10 MB of contents has been signed. One signature has been added to both documents and then same tests repeated with 10 signatures per document, which resulted in four distinct tests. Each of those four tests has been performed 20 times to account for variance in results. The measured time includes the time spent by invoking the digital signature app by sending it an intent and getting the results back to the original app, as it is this complete time that is relevant for a third party app that uses the infrastructure.

It has been observed that on old devices like Google Nexus S that have less memory than most modern Android devices, a fatal out-of-memory error occurs during verification of signatures for the 10 MB test document. The error occurs also on modern devices when the document is significantly larger, e.g., a 100 MB document on Nexus 10 tablet. The cause of this error is parsing XML with the

simple and easy to use Document Object Model (DOM) [17] parser interface which was required by previous versions of Apache XML Security for Java [19]. Since Apache XML Security for Java version 2.0.0, released in May 2014, StAX parser [20] can be used instead. Unlike the DOM parser, StAX is a streaming parser and does not load the whole document into memory. In case of large XML documents, it is usually more efficient, faster, and consumes less memory. It is expected that switching to StAX parser would not only solve the out-of-memory errors, but also optimize the signing and verification procedures on the Android client in terms of speed and resource usage, especially in case of large XML documents where only a small portion of the document is signed or being signed. According to severity classification of problems [17], the out-of-memory error is potentially a critical problem and should be among the first issues resolved.

TABLE I. VERIFICATION TIME ON ANDROID IN SECONDS

Document size	**1 KB**		**10 MB**	
Signatures	***1***	***10***	***1***	***10***
Time min	0.294	0.365	2.012	7.882
Time Q1	0.318	0.402	2.086	7.960
Time median	0.328	0.425	2.113	8.053
Time Q3	0.352	0.436	2.188	8.114
Time max	0.397	0.466	2.225	8.177

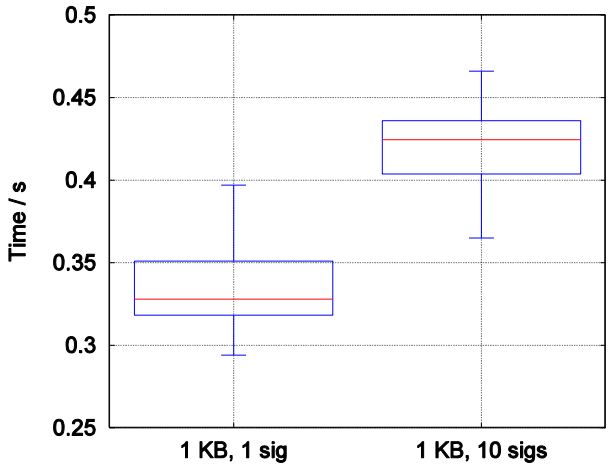

Figure 9. Verification time on Android client for 1 KB documents.

The statistical parameters of elapsed time for all four tests are listed in Table I and presented by box plots in Figure 9 and Figure 10. Minimum, the first quartile, median, the third quartile, and maximum are shown for verification time of each document. As expected, the time to verify signatures significantly increases with the size of signed data in the document (which approximately matches the document size). It also increases with number of signatures, but this increase is, as expected, more evident with large document sizes (Figure 9 and Figure 10). Unlike the server processing times, the Android client processing times are much more consistent and less variant. There is also no noticeable difference between the time of first verification after Android system restart and subsequent verifications. These results confirm that the user would experience small enough times of local verification in any case where the number of signatures and the document size are not too large. For cases where this assumption is not satisfied, an alternative solution with group signatures may be more appropriate.

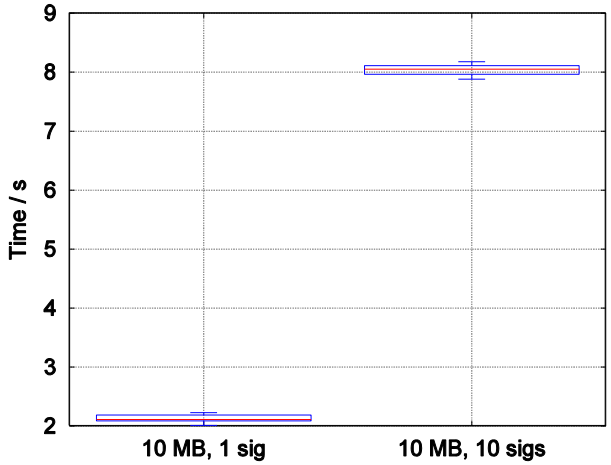

Figure 10. Verification time on Android client for 10 MB documents.

VI. 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. *Pervasive Meeting Minutes*

"Pervasive Meeting Minutes" application allows for unobtrusive capturing of meeting minutes. The application uses and interacts with the community signature infrastructure. Community micro-agreements infrastructure allows 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 12) and its URL is distributed to meeting participants. The REST servers which handle distribution of document URLs and receive notifications about document finalization (Figure 12) 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 (Figure 11), or – as the last resort – copy the URL that is displayed below the QR code (Figure 11). 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.

Figure 11. Meeting attendants without the app can scan the QR code to download the signed document.

Arrows in Figure 12 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.

On the client side, directly seen by the users, the central Android device is running the main meeting app, seen by all users and shown in Figure 13. On the right, meeting details and minutes are shown. On the left, list of currently registered and manually added users is shown. Any number of users can use this central tablet as an alternative for their own Android devices to participate in the meeting. To provide input, a user only has to click his name in the list and speak or type his input. For the registered users, it is of course more convenient to use their own smart phones and tablets to provide their input, which is then automatically and instantly distributed to the other users and the central Android device in Figure 13 through dedicated REST server and GCM, in similar manner as the document URL is distributed in Figure 12.

Regardless of the implementation, the signatures are always in standard XMLDSig form, as in Figure 14. 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.

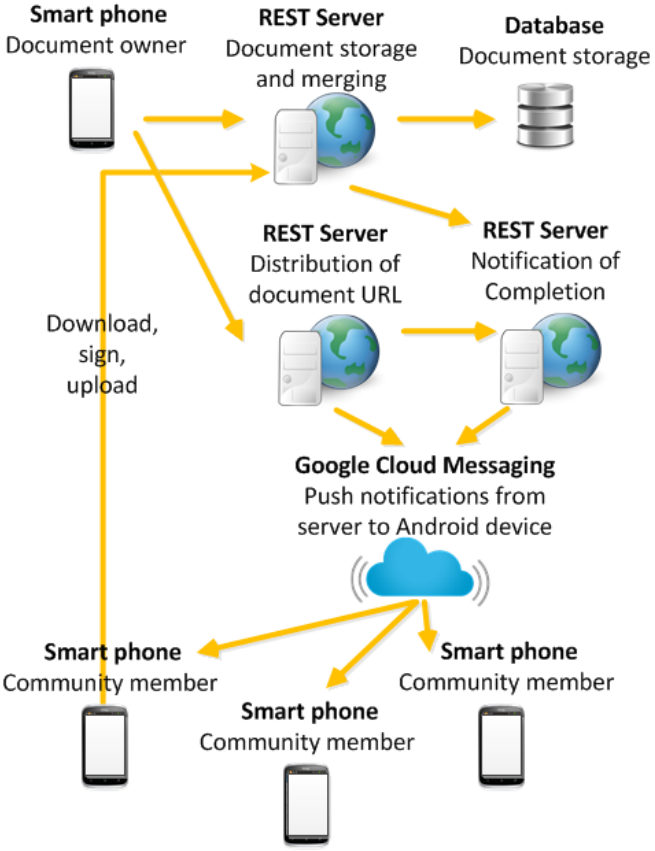

Figure 12. Process and information flow between devices in a chosen implementation for capturing meeting minutes.

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 14 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 14.
- 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.

Figure 13. Pervasive Meeting minutes app with a meeting in progress.

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 15), 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.

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.

```
<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 14. An example of meeting minutes document structure which is signed by two parties.

Third party apps can also take a common approach to integrate with the community signature solution and provide a single complete service. E.g., within Crowd Tasking, the user can initiate new meetings within a task or join existing meetings to discuss task management and work distribution with other users. The pervasive meeting minutes software described in previous section is then automatically initialized and shown on the central Android display (Figure 13). When the meeting is finished, the users receive notification to sign them within the Crowd Tasking app, which then uses community signature solution to sign the document.

C. *Service Sharing Within a Community*

The policy negotiation described in [2] 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.

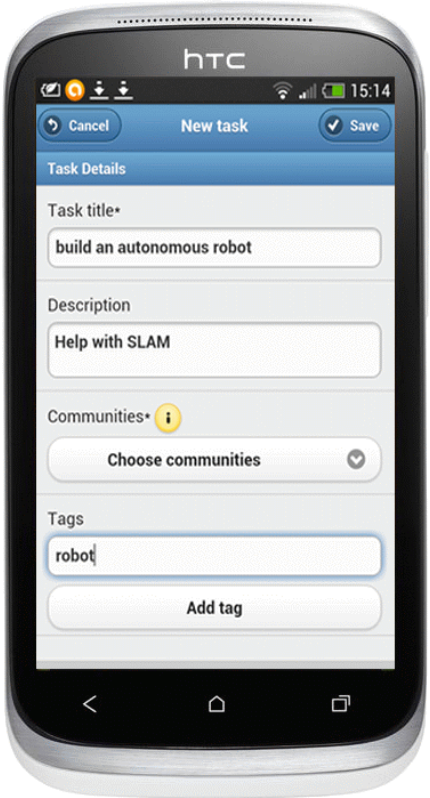

Figure 15. Crowd Tasking Service.

VII. User Evaluation

Separate user trials were organized in order to gain feedback about the implemented functionality and features, and how they could be applied to business communities. The concept of focus groups has been used. A focus group is a moderated discussion among a small group of people who discuss a topic under the direction of a moderator, whose role is to promote interaction and keep the discussion on the topic of interest [21][22]. A focus group technique stems from social research, but is now widely used to investigate new ideas in many research fields. In design science focus group technique can be useful as an exploratory method to achieve incremental improvements in artefact design or as confirmatory method to establish the utility of the design in a field use [22]. In our case purpose of focus groups was to collect ideas to improve user experience, which is very important in pervasive environment.

Two focus groups with external users took place in January and February 2014 at SETCCE research laboratory in Tehnološki Park 21, Ljubljana, Slovenia.

Prior to conducting the focus groups, the software was integrated with facilities in SETCCE research laboratory. Then an internal evaluation with SETCCE employees took place. The participants used software prototypes and gave their opinion and propositions, which have been taken into consideration when further improvements were made. The improved version was used for both first focus group with two participants and second focus group with three participants.

A. The Trial Participants

The external participants selected for evaluation have been chosen to represent the business communities. They are SETCCE's key customers coming mainly form Telecommunication and Insurance industries. The participants were all men, aged 31-60, and have from up to 5 years to over 10 years of experience in negotiation and participation in business meetings and assemblies. They are involved in decision making. They very frequently use mobile apps (most use Android), but had not participated in the design of the proposed solutions and were not familiar with the proposed infrastructure and concepts. Because of this, the evaluation was conducted as a focus group rather than a "trial" where participants are given tasks and interact with the software themselves. There are many possibilities how Crowd Tasking can be used so we had to focus usage to get relevant feedback.

B. Trial Objectives and Deployment

The meetings started with a demonstration video and presentations to explain Crowd Tasking service and the proposed infrastructure for community signatures. These were followed first by prototype demonstration and then by a moderated discussion and filling a questionnaire. During the trials, the participants were given the smart phones and tablets on which they were able to use the Crowd Tasking service, the pervasive meeting minutes service, and community signatures under moderated conditions. Four Android devices were used: a tablet to capture meeting minutes and serve as the initial document owner; three Android smartphones / tablets were used as normal user devices.

Demonstration consisted of the following steps:

- Creation of a new task and creation of a new meeting within the task.
- A user starts the meeting and the other users check in to the meeting.
- Holding a meeting.
- Meeting finalization.

Community signature solution is used in the last step. When the participants agreed on details about further task actions and solving, the meeting was finalized. Each participant received a notification to sign the meeting minutes. While one user signed the minutes, the others watched their phones to see how the document status changed when the first user signed it and how it changed when the process finished when the required number of users signed it.

The main objective is to get answers to the following questions:

- Are the implementation concepts of collaborative signature and micro-agreements suitable for use by other applications?
- Are the functionality and main concepts of collaborative signature easy to understand and be

accepted by typical business users? What are the obstacles for adoption by business communities?

C. Trial Staff and Rollout

The moderator was responsible to provide an introduction to the focus group, explain participants the code of behaviour and confidentiality (the names of the participants will not be made available publicly, no voice recording takes place, except during voice to text conversion during service demonstration, etc.), ask the questions as specified by the questioning route (see below), complement/clarify these questions if necessary, maintain the flow of conversation, make sure everyone has a chance to share their meanings, etc. He should create a comfortable, open atmosphere and avoid head nodding and verbal comments that signal approval of a meaning, avoid giving personal opinions.

The two presenters are responsible for providing an objective and balanced presentation of the concepts, demonstration of the services, aiding the participants in experimenting with the services themselves, etc. A presenter should not try to address criticism that comes up during the discussion. However, this was hard to avoid since both presenters had the best knowledge and understanding of the services and technologies used.

The assistant moderator is responsible to collect the list of participants, write down relevant ideas, comments and other parts of discussion, take notes throughout the session, and record any non-verbal activity that might help to correctly interpret the users' comments, or that might signal approval or disapproval.

D. Trial Questionnaire

Questionnaire consists of both open-ended questions (respondent formulates his own answer) and closed-ended questions (respondent picks an answer from a given number of options). The latter type includes polytomous (respondent has more than two options) and continuous (respondent presented with a continuous scale) questions. When preparing the questionnaire we strove that questions follow each other logically, from the least sensitive to the most sensitive, from the factual and behavioral to the attitudinal, and from the more general to the more specific.

E. User Feedback

The users provided separate feedback for the Crowd Tasking service and the underlying Micro-Agreements functionality. The complete questionnaire and aggregated answers are available in [23]. The items below summarize their answers and draw some conclusions.

1) Feedback Related to Crowd Tasking and Pervasive meeting minutes

Options to easily create new communities and invite members to meetings have been unanimously marked as useful.

Most of the questions about the usefulness and appropriateness of the "Crowd Tasking" was answered unanimously. Participants agree that the app has value in the business world/business processes, that by using the app some aspects of the business could be optimized and their work could be simplified. The app is easy to use. The app works as you would like, user experience is a positive one and the app is not too complex. Different functionalities in the app are well integrated, the app seems useful to them and they would use it. They would recommend the app to a colleague, customer, and partner. The app seems to be appropriate to confirm the agreements reached in the negotiations and to confirm decisions made at the Assembly meetings.

The participants did not agree if the app supports all the expected functionality and that the app is suitable for all levels of users.

Participants would like to see support for different workflows and better document visualization.

As positive aspects of the app they see mobility and virtual business environment, interoperability of different devices, simplicity and ease of use. A negative aspect is that the app works only on Android platform. In particular community there will be also someone with iPhone, Win device. Participants said that their experience of this app was wonderful, easy, satisfying and stimulating. Although most of this feedback is not directly related to the solution described in this paper, the feedback shows it is possible to create a simple to use and satisfying app based on the micro-agreements solution described here.

2) Feedback Related to Micro-Agreements platform

The existing and presented functionality of micro-agreements has been estimated as useful and important by the participants. Equally important was the ability to be easily integrated and used by third party services. However, suggested integration with other document repositories in the cloud (e.g., DropBox and Google Docs) has been marked as less important.

When signing contracts or agreements, trust in the other party or parties is particularity important. The participants feel that an assessment of trust level for other parties would be very beneficial. The participants think that they would benefit from clear visual presentation of trust levels at the time of signing. Nevertheless, only some would be willing to sign an agreement with users they are not familiar with, even if they had their trust level displayed. This is yet another lead to conclusion that the business users are still somewhat conservative and cautious when adopting new solutions and technologies that do not strictly mimic the established paper based business practices.

In general, the participants did not bring up any big privacy concerns or issues. They were more concerned about the trust in other signers, as already described above.

An issue that does not seem to bother business users too much, or not at all, is fair exchange [5][6][7] of signatures. The signing process of micro-agreements uses the common digital signatures with identity information embedded in XML document (XMLDSig). When a user signs an agreement, his identity is added to the signature and subsequent signers of that document can see the list of all previous signers. This is in contrast to the newer concept of "fair exchange" of digital signatures where the identities are not disclosed until a so called keystone is released. At that

moment, all identities become known to all users. While the fair exchange concept is indeed fairer to members of communities where all members are equal, the participants valued the micro-agreements feature of showing the list of identities that have signed the document so far. The business users, especially the more traditional ones, may be reluctant to sign a document where they cannot see the existing signatures from previous signers. Premature exposure of early signers' identities seems to be necessary, even though it does not result in a fair exchange of signatures.

Surprisingly, not all users considered asynchronous signing (without a pre-determined order) very useful, which again shows some users prefer traditional ways of signing documents where the signers sign a document in a sequential manner.

VIII. 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.

The tests of implemented prototype showed it is usable for small documents even if only a single low cost server is used. With the exception of 10 MB documents, the signing times are low enough for most realistic use cases, but should still be significantly improved for a production version.

The focus group and observation meetings with the help of business community users from telecom and insurance industries were an enriching experience. We learnt about their work process and extensively explored their needs. The participants were open for new technologies and further cooperation, and provided several suggestions and ideas how their own business processes could be improved using such technologies.

The trial participants expressed clearly that they would prefer to use a system tailored to their business process and business needs. They informed us that solutions built on SOCIETIES results would need to reuse and reintegrate the elements and approaches from SOCIETIES project [24] and adapt them to their business context, rather than reuse the whole integrated SOCIETIES platform. By conducting the user evaluation we were able to gain information on how to upgrade and improve the pervasive meeting minutes application as well as the community micro-agreements infrastructure in order the end-users' experience.

Next steps will be to separate the community micro-agreements infrastructure from the integrated SOCIEITES open source platform. The focus will be to make a stand alone prototype that will only feature the functionalities that support the e-business community requirements and scenarios. During this process we will improve the implemented functionalities according to the trial results. We will integrate it with different mobile e-business applications that require digital signing capabilities to gain valuable feedback from 3rd party application developers.

We plan to extend micro-agreements infrastructure to support Stork2 eID based cross-border authentication [25] and due to its open-source nature to promote is as one of main candidates for smart phone authentication building blocks for eSENS project [26] .

Additionally we plan commercially reuse parts of community micro-agreements infrastructure. To achieve this we will need i.) to port the technology to others mobile platforms such as Windows Phone and iOS, ii.) to improve the verification of correct binding and mapping between community member identities and X509 certificate based identities used for digital signing, iii.) to extend the infrastructure with e-signing and e-archiving capabilities and finally to iv.) integrate results into SETCCE's commercial services and products, such as ePero®Start [27], eNvoices® [28] and ProXSign® [29].

Acknowledgment

Authors would like to thank the SOCIETIES project [24] consortium, the STORK2 [25] project consortium, the eSENS [26] project consortium and the EC for sponsoring the projects.

References

[1] M. Vardjan and J. Porekar, "An infrastructure for community signatures and micro-agreements," ICDS 2014: The Eighth International Conference on Digital Society, ISBN: 978-1-61208-324-7, March 2014, pp. 13-19.

[2] 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.

[3] XML-DSig, XML Signature Syntax and Processing, 2nd Edition http://www.w3.org/TR/xmldsig-core/, [retrieved November, 2014].

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

[5] X. Tan, Q. Huang, and D. S. Wong, "Concurrent signature without random oracles," IACR Cryptology ePrint Archive, 2012.

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

[7] 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.

[8] 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 p. 307-313, DOI: 10.1049/ip-cdt:19941293.

[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.

[10] 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.

[11] ITU Recommendation X.667 (09/04), http://www.itu.int/rec/T-REC-X.667-200409-S/en [retrieved November, 2014].

[12] X.509 standard recommendation, http://www.itu.int/rec/T-REC-X.509/en [retrieved November, 2014].

[13] Canonical XML 1.1, W3C recommendation, http://www.w3.org/TR/xml-c14n11/, [retrieved November, 2014].

[14] 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.

[15] 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.

[16] 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.

[17] E. J. Weyuker and Filippos I. Vokolos, "Experience with performance testing of software systems: issues, an approach, and case study," IEEE Transactions on Software Engineering, Vol. 26, No. 12, pp. 1147-1156, Dec. 2000, doi:10.1109/32.888628.

[18] Document Object Model (DOM), http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113, [retrieved November, 2014].

[19] Apache XML Security for Java, http://santuario.apache.org/javaindex.html, [retrieved November, 2014].

[20] StAX Java XML parser, https://sjsxp.java.net/, [retrieved November, 2014].

[21] D. W. Stewart, P. N. Shamdasani, and D. W. Rook, "Focus groups: theory and practice," 2nd ed., vol. 20, Sage Publications, Newbury Park, CA, 2007.

[22] M. C. Tremblay, A. R. Hevner, and D. J. Berndt, "The use of focus groups in design science research," Design Research in Information Systems, vol. 22, Integrated Series in Information Systems, DOI 10.1007/978-1-4419-5653-8_10, Springer US, 2010.

[23] SOCIETIES FP7 EU project deliverable D8.9, "Final evaluation report," 2014.

[24] Self Orchestrating CommunIty ambiEnT IntelligEnce Spaces (SOCIETIES), EU FP7 project, Information and Communication Technologies, Grant Agreement Number 257493. http://www.ict-societies.eu/ [retrieved November, 2014].

[25] Secure idenTity acrOss boRders linKed 2.0 (STORK2), EU cofunded project INSFO-ICT-PSP-297263, https://www.eid-stork2.eu/ [retirieved November, 2014].

[26] Electronic Simple European Networked Services (e-SENS) is a large-scale project that embodies the idea of European Digital Market development through innovative ICT solutions, http://www.esens.eu [retrieved November, 2014].

[27] SETCCE ePeroStart @ http://www.epero.si/ [retrieved November, 2014].

[28] SETCCE e-invoicing platform (eNvoices) @ http://www.setcce.si/item.php?catId=22&itemId=70&section=4 [retrieved November, 2014].

[29] SETCCE ProXSign @ http://www.proxsign.com/ [retrieved November, 2014].