Discussions note on Contract eXchange Requirement

Nat Sakimura (=nat), Nomura Research Institute, Ltd.
Date : 2009-07-30

Abstract

Contract Exchange (CX) is supposed to allow parties to exchange mutually digitally signed legal contract. It is a file format for such a contract, and the protocol to actually exchanging it over the network. This article first discusses abstract requirement and then discusses profiling it onto OpenID protocols.

1. Contract

1.1 Items in a Contract

Contract is a mutually signed data that fulfill certain properties. In general, a Contract will have the following structure:

- Contract
  - Contract ID
  - Party A ID
  - Party B ID
  - Signatory of Party A (text)
  - Signatory of Party B (text)
  - Contact Address of Party A (text or uri?)
  - Contact Address of Party B (text or uri?)
  - Main Content of this Contract
    - What is to be provided (text)
    - What is received in return (text)
  - Term and Termination
    - Term / Validity Period of the Contract (Datetime-Datetime)
    - Termination (text)
    - Survival of Certain Terms (text)
  - Damages
    - Explanation (text)
    - max amount from A to B (number?)
    - max amount from B to A (number?)
  - Non Disclosure
    - How to specify (text)
    - How Long (datetime-datetime)
  - Relationship to other Contracts (text)
  - Signature of the Signatory of Party A (text)
  - Date of the Signature A (datetime)
  - Signature of the Signatory of Party B (text)
  - Date of the Signature B (datetime)

It is the aim of this extension to create such a document over a simple protocol.

1.1 Document Format

OpenID choice of the document format seems to be name=value pair and XML seems to have been generally avoided. For sometime, some of the proposers also have been thinking in this line to avoid XML Signature.

However, now that XRD requires XML Signature with exclusive c14n, most OpenID platform would have XML processor and Signature facility. If we opt to use XML, then, we would not be required to define Signature method. Also, as a long term document, being able to leverage on various standards on timestamping on XML would benefit us.

Thus, unless there is a strong opposition, it would be quite logical to use XML as the preferred format.

Other possibility includes such things like JSON, XDI, etc.

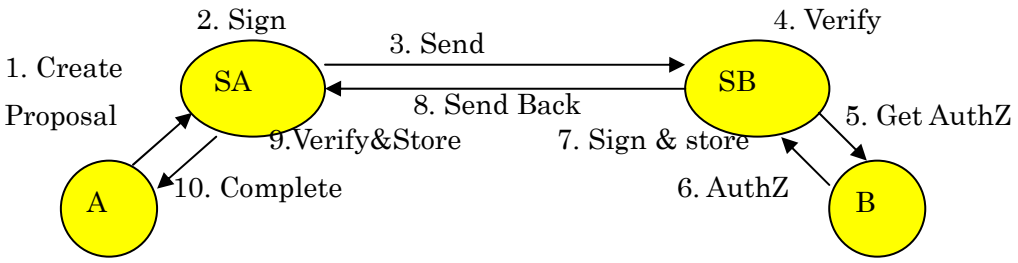
See Issues/Discussion at the end of this document.

2. Use Case Patterns and Protocol

There are typically 4 actors, A, SA, B, SB, where SA and SB are the designated signatory of A and B respectively.

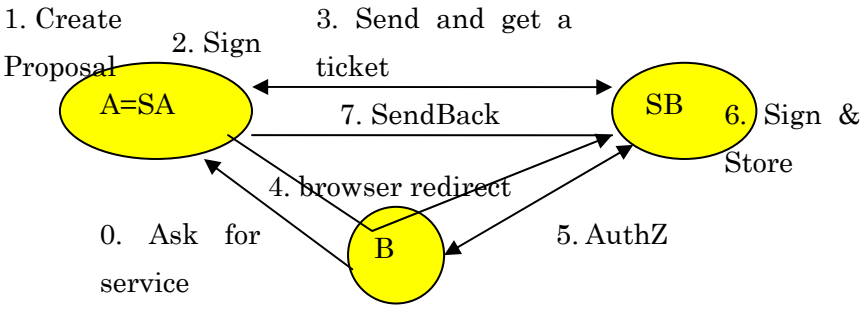
Pattern 1: Generic 4 actors model.

This is the most generic pattern that all of A, B, SA, SB present. In this pattern, A creates (or picks) proposal, SA signs and sends it to SB, SB asks if B agrees to it, and if B agrees, SB signs it and send it back to SA. Note that in this case, 5. Get AuthZ is has to have some kind of push or notification mechanism since B may as well be offline.



Pattern 2: 3 Legs OpenID/OAuth like use case.

This is a more typical case that is talked about usually. This is a special case of the abover where A=SA and B kicks the process at A.



3. Subject-Signer Relationship

When Signatory of A is actually A, it is evident from looking at the signature that A is making the offer. However, if the Signatory of A is not A, then the Signatory might be forging the document without A's instruction. Thus, if Signatory of A != A, then we must verify A's Authorization by asking A.

This is where the trust profile comes into the picture.

One way of doing it is to make it the task of the B's Service Provider. To do so, it needs to first authenticate A and ask if A agreed to let Signatory of A to sign it on his behalf.

Another way of doing it is to utilize XRD. Party A can publish Designated Signatory for

CX service in his XRD. E.g.

```
<xrd>
    <subject>http://partya.example.com/</subject>
    <link>
          <rel>http://specs.openid.net/rel/mycxrp</rel>
          <url>http://cxrp.example.com/</url>
          <Subject>http://cxrp.example.com/</Subject>
          <ds :KeyInfo> </ds :KeyInfo>
     </link>
</xrd>
```

Note that here can be multiple <link> that specifies CX services.

Here, xrd/link/ds:Keyinfo signifies one of the Designated Signatory of Party A for CX service.

This kind of xrd/link may be regarded as the pre-authorization for the Signatory so that the user intervention will not be required. From the user interface perspective, this should be the default, and the active AuthN/AuthZ should be done only if this fails.

4. Protocol

Since the Proposal and Agreement can be fairly large as it may include various text that it sometimes is not desirable to go through the client browser because it may choke.

Thus, sending and receiving proposal and agreement in direct communication channel is desired. In this way, only small reference to whatever have been moved across the wire in the direct communication is needed to sent through the client browser.

In the following, Capitalized words has the following interpretation.

A: Party A
SA: Party A's signatory and service provider.
B: Party B
SB: Party B's signatory and service provider
Sign: XML DSig with exclusive Canonicalization

XRD: XRD 1.0 defined in OASIS Open XRI TC.

Aurl: Contract Authorization End Point URL of B

Pattern 1

1 A creates a Proposal. (Usually, picks a preset "plan" available.)
2 SA Signs the Proposal
3 SA sends the Proposal to SB through direct communication.
4 SB fetches Party A's XRD and verifies the Subject-Signatory relationship.
5 SB asks B if he agrees.
6 B agrees.
7 SB counter signs the Proposal.
8 SB sends it back to SA
9 SA fetch's B's XRD to verify B-SB relationship and Stores.
10 The transaction Complete.

Pattern 2

1 B asks A=SA for a service.
2 A=SA creates a Proposal. (Usually, picks a preset "plan" available.)
3 A=SA Signs the Proposal
4 A=SA sends the Proposal to SB through direct communication.
5 SB fetches Party A's XRD and verifies the Subject-Signatory relationship.
6 SB returns Ticket that contains unique identifier for this transaction and Aurl to A=SA
7 A=SA creates 302 Redirect to Aurl.
8 SB asks B if he agrees.
9 B agrees.
10 SB counter signs and store the Proposal.
11 SB creates 302 Redirect to ReturnToUrl. Parameter includes Contract location.
12 A=SA fetches the signed contract from SB.
13 SA fetch's B's XRD to verify B-SB relationship and Stores.
14 The transaction Complete.

5. Privacy and Security Considerations

In Pattern 2, B might want to remain anonymous from the privacy reasons. In this case, Proposal cannot include B's identifier. Instead, it should include identifier_select as the

value. Under such circumstances, SB must authenticate B and create PPID (Pair wise Pseudonymous Identifier.) This has implication on both the Contract format and Proposal verification rule that on top of the B's identifier field in Proposal, there has to be an independent field for B in the Agreement. When real identifier is set in the proposal, the both must match. If the identifier of B in the Proposal is identifier_select, then Agreement's corresponding field takes precedence.

6. Contract format

Thus, Contract consists of Proposal and Agreement. Reorganizing the fields in 1. :

```
Contract
  Proposal
    Contract ID (uri)
    PartyA
        Party Auri
        Signatory of Party A (ds:KeyInfo)
        Contact Address of Party A (text or uri?)
    Party B
        Party Buri
        Signatory of Party B (text)
        Contact Address of Party B (text or uri?)
    Main Content of this Contract
        Obligations of Party A (text)
        Obligations of Party B (text)
    Term and Termination
        Term / Validity Period of the Contract (Datetime-Datetime)
        Termination (text)
        Survival of Certain Terms (text)
    Damages
        Explanation (text)
        max amount from A to B (number?)
        max amount from B to A (number?)
    Non Disclosure
        How to specify (text)
        How Long (datetime-datetime)
    Relationship to other Contracts (text)
```

Signature of the Signatory of Party A (text)
Date of the Signature A (datetime)
Agreement
    Party B
        Party Buri
        Signatory of Party B (text)
        Contact Address of Party B (text or uri?)
    Signature of the Signatory of Party B (text)
    Date of the Signature B (datetime)

7. OpenID Binding of Pattern 2

7.1 Mapping the sequence onto OpenID Authenticaiont 2.0 and AX

Now, let us map the sequence to OpenID Authentication 2.0.

1 B asks A=SA for a service.
2 A=SA creates a Proposal. (Usually, picks a preset "plan" available.)
3 A=SA Signs the Proposal
4 A=SA sends the Proposal to SB through direct communication: Association or Future AX Direct Message.
5 SB fetches Party A's XRD and verifies the Subject-Signatory relationship.
6 SB returns Ticket that contains unique identifier for this transaction and Aurl to A=SA : Association or Future AX Direct Message.
7 A=SA creates 302 Redirect to Aurl with Ticket : AX schema
8 SB asks B if he agrees.
9 B agrees.
10 SB counter-signs and store the Proposal.
11 SB creates 302 Redirect to ReturnToUrl. Parameter includes ContractID.
12 A=SA request the signed contract from SB. : Piggyback on AuthN 2.0. check_authentication message.
13 On the response to 11, Contract Specified in openid.cx.contractid is returned in addition to normal AuthN response.
14 SA fetch's B's XRD to verify B-SB relationship, and verify the signature, and Stores.
15 The transaction Complete.

7.2 Issues/Discussion Points

There apparently are some issues though.

a) Performance Issues associated with piggybacking on association.

In step 3, if we were to piggy back on association, it probably means that association needs to be re-done each time. This may create a scalability problem especially when DH association is used.

While allowing it to be done with the association, it may be good to have separate message mode for the case we do not want to re-do the association.

e.g., openid.ns=__cx_type_url__&openid.mode=cxproposal

When AX 2.0 comes along, this pain will be eased so that openid.ns=__ax2.0_type_url__&openid.ax.cxp=__cx_proposal_schema_url_ etc.

b) Performance Issues associated with piggybacking on check_authentication

In most cases, AuthN response is valid. It is redundant to do the check_authentication just for the sake of the piggybacking. It might be more desirable to create a custom message type. Again, when AX 2.0 comes along, this problem is solved.

c) Contract Format

In the above, I have only stated the contract abstractly. It could be expressed in any of name=value, XML, XDI, JSON, etc. as long as there is sufficiently deployed public key crypto based signature scheme.

Since signed XRDs are used here, XML parser and XML Signature is a prerequisite anyways. So, XML seems to be a logical choice, but other formats has other merits as well. It needs some discussion. Perhaps we can just state the supported format in XRD so that we can use any of them when it is appropriately profiled.