Negotiation Based on Individualization: Incorporating Personalization into Federation

Javed I Khan, Kailas B Bobade, and Manas S Hardas

Abstract— **In virtual business place, organizations store information of its members. Federated Access Control Systems such as Shibboleth, Active Directory Federation Service allow virtual organizations to share their member's information. Based on this information, members enjoy seamless access to federated resources. However in this federated world, a member's information is divulged by her home organization. The member has little say in it. We have presented an extension to this work where members can personalize their own attribute release policy. As opposed to simple request reply based communication, such personalization inherently necessitates a mechanism of negotiation. To facilitate such personalization, we have presented negotiation enabled framework in federation. In this paper, we provide an extension to this framework to facilitate selection of negotiation flavor on per-need-basis. This is supported by negotiation protocol which defines the ordering of the messages and unique message structure that carries negotiation information.**

Index Terms— **Privacy, Security, Negotiation, Federation, Authentication**

I. Introduction

A Federation is an association of organizations that uses common set of attributes, practices and policies to exchange information about their members. Based on this exchanged information, members enjoy seamless access to federated services. Such services are provided by service provider organizations subject to Access Control Policy. When a member requests a service, service provider organization enables backend exchange protocol to retrieve necessary attributes of that member by querying member's home organization. Such system is called *Federated Access Control System* (FACS) and examples are Active Directory Federation Services [6], and Shibboleth [5]. FACS facilitates service provider organizations to receive federated member's attributes issued by her home organization. So it has claimed to increase privacy of federated members. In a way it is true, but private person is actually absent from such system so does individualized privacy. In FACS, access control policy is set by home organizations- not by individuals. Individuals have very little knowledge- least say in how their information is released by home organization. However there are additional aspects of privacy. Alan Westin [7] has defined privacy as: "The right of individuals to determine for themselves when, how and to what extent information about them is communicated to others." There could be various scenarios where a home organization has to disclose member's private information by her acquiescence. Let's consider federation of Universities and Companies.

In this federation, students will apply for jobs in federated companies to schedule an interview. Companies request information like Transcript, SSN, and Email from students. University could release student's information as per her acquiescence. But students often prefer to release same information to different companies under different conditions. For example, provide a student's Transcripts only when company is offering job in Operating System or Software Engineering. Same could be true for the companies. To provide such facility, it is imperative to have Individualized Policies for federated members. Individualized policy is a course of action created according to the specifications of an individual to determine information release decisions in context of service provider's offer.

Impact of such personalization is however non-trivial. As opposed to simple request reply based communication, such personalization inherently triggers exchange of information between parties which is called as negotiation. In [4], we presented a negotiation enabled framework. This framework has Negotiation Agent which negotiates with its peer, both acting on behalf of the individual members, to produce customized negotiation results. In this paper, we extend that framework to facilitate selection of negotiation flavor on per-need-basis For example; users might be interested in less number of communication steps or disclosing only necessary credentials or releasing credentials only when it becomes clear that successful negotiation is possible. To accommodate most of these likely requirements, we are presenting a negotiation protocol in section 2 and algorithm in section 3.

Previously, [1] proposed a model to authorize action on personal data i.e. Individualization. But it doesn't provide interaction-mechanism between policy holders - an inherent need of Individualization. Also, [3] has integrated trust negotiation with federation. But this work does not consider negotiation of attributes, negotiation protocol, and multiple negotiation flavors.

Manuscript received July 30, 2007.

J. I. Khan is Professor with Kent State University, Kent, OH 44241 USA (e-mail: javed@kent.edu).

K. B. Bobade is Graduate Student with Kent State University, Kent, OH 44241 USA (e-mail: kbobade@kent.edu).

M. S. Hardas is Doctoral Student with Kent State University, Kent, OH 44241 USA (e-mail: mhardas@kent.edu).

II. NEGOTIATION PROTOCOL

This section provides a prototype of protocol for bi-partite negotiation. Protocol determines the sequence of messages and a message structure carrying negotiation information.

A. State Transition Diagram

A negotiation proceeds through six states namely Advertisement, greetings, strategy, active, negotiation, and adieu shown in figure 1. Important messages used in negotiation are Greeting, Advertisement, Solicitation, Strategy, Negotiation, Deal, No_Deal and Reporting.

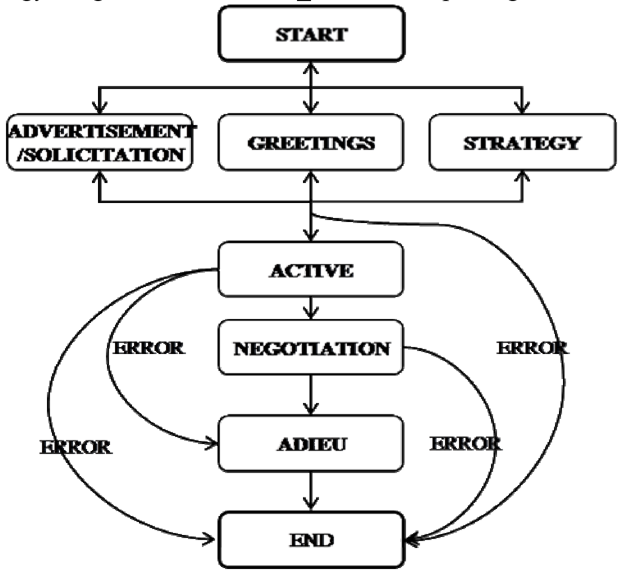


Fig 1. State Transition Diagram for negotiation Protocol

Negotiation moves from start to greeting state, when one party invites other for a negotiation session by offering initial identity. After sending greetings message, each party waits for greetings message from the opposite party. Opposite party can accept or reject the invitation (and negotiation proceed to adieu state). If both parties agree then negotiation enters into active state. Here, negotiation can also moves to advertisement or strategy state. In advertisement state, a party sends advertisement messages naming Target Resources and waits for the other party to reciprocate by sending solicitation (advertisement) message. If one party is interested to offer the Target Resource and other party is interested to obtain it then negotiation moves to active state otherwise moves to adieu state if any one of them is not interested. In strategy state, a party accepts list of negotiation strategies from opposite party, and if both parties agree on a strategy then proceed to active state (or jump to adieu state). Negotiation moves from active to negotiation state when both parties fulfilled prerequisite like receiving each other's identity or deciding strategy or deciding resource to negotiate. In negotiation state, actual negotiation starts by sending negotiation message and protocol moves through series of exchanges until negotiation becomes successful or unsuccessful. After this, negotiation moves to adieu state. In adieu state both sides perform reporting about current negotiation session and then moves to the end state. Negotiation protocol moves to end state if any error condition occurs at any given state.

B. Negotiation Message

Negotiation Message has two main compartments - header and body. Header contains Action, Session Id, Strategy, and Security fields. Strategy field carries negotiation strategy chosen by Negotiation Initiator and Negotiation Responder. This field is useful when two parties negotiate for negotiation strategy. Security field carries encrypted identities of participant organizations, and members. Action field contains one of the messages described in section 2.1.

Body has Initiator's Resource List (IRL) and Responder's Resource List (RRL). Each item in these lists has the following fields. Resource Descriptor (RID) is the name of resource to uniquely identify it. The Value field contains the value or an URL to the value of the resource. The type field of a listed resource can have one of the following values namely Personal (P), Credential (C), Attribute (A) or Information (I). The License field contains the granted disclosure policy for the resource. State field contains current state of negotiation of a resource during negotiation process. Extra field carries state information of every resource involved in negotiation, if necessary. Previous two fields achieve stateless resolution process i.e. Negotiation parties wouldn't require maintaining complex state information.

III. STATELESS PROXY ATTRIBUTE NEGOTIATION

We have presented one flavor of negotiation resolution algorithm i.e. Stateless Eager Attribute Negotiation (SEAN) in [4]. Here, we will discuss Stateless Proxy Attribute Negotiation (SPAN) which is based on PRUNES [2]. Above mentioned negotiation message can accommodate SEAN as well as SPAN.

In the first phase of proxy negotiation, two sides exchange the resources' name to let each other know their requirement constraints without disclosing resources values. During this phase both parties track the requirement dependencies. Here, goal is to determine if the deal is possible and If it is, only then they perform the actual resource exchange in second phase. Based on tracked requirement dependencies in the first phase, both parties exchange resources using optimum sequence in second phase. We describe this algorithm next.

Symmetrical SPAN runs at both negotiating parties, but here we assume that SPAN is running at Responder's end.

```
SPAN_state_machine(rlist, $P^R$) {
1. OLD_REQSET={$A^R \in$ rlist: $A^R$. state = REQUESTED}
2. OLD_RELSET ={A $\in$ rlist: A. state = AVAILABLE}
3. OLD_DENSET ={A $\in$ rlist: A. state = DENIED}
4. OLD_PENSET ={$A^I \in$ rlist: $A^I$. state = PENDING}
5. RELSET = REQSET = {Ø}
6. nego_state = CONTINUE
7. If ({$A^R \in$ rlist: {$A^R$ is service request} and {$A^R \in$
        OLD_DENSET} })
8.      nego_state = NO-DEAL
9.      return ({Ø},{Ø},{Ø},{Ø},nego_state)
10.Else
11.   If ({$A^R \in$ rlist: {$A^R$ is service request} and
         {$A^R \in$ OLD_RELSET}})
12.      nego_state= DEAL
13.      return ({Ø},{Ø},{Ø},{Ø},nego_state)
14.   Else
      {
15.     For (each $A^R$ in OLD_REQSET)
          {
16.        {{REQSET}, {RELSET}} =
           SPAN_rule_resolver($A^R$, rlist, RULE[ $A^R$])
17.        NEW_PENSET = {NEW_PENSET} U
           {$A^R \in$ OLD_REQSET: {REQSET $\neq$ Ø}
           and {RELSET=Ø}}
18.        NEW_REQSET = {NEW_REQSET} U
           {REQSET}
19.        NEW_RELSET = {NEW_RELSET} U
           {RELSET}
20         NEW_DENSET = {NEW_DENSET} U
           {$A^R \in$ OLD_REQSET: {REQSET=Ø} and
           {RELSET=Ø}
        } // End of for from line 15
21.     NEW_RELSET = {NEW_RELSET} U
        OLD_RELSET}
22.     return ({NEW_REQSET},{NEW_RELSET},
        {NEW_PENSET},{NEW_DENSET}, nego_state)
   } // End of else from line 14
}
```

```
SPAN_rule_resolver($A^R$, rlist, Rule)
 {
1. GARC = Count of Released attributes in M
2. If (Rule = = True)
3.    LURA [$A^R$] = ~
4.    $A^R \in$ RELSET
5. Else
6.  { While (Rule)
7.     { Clause = Next Clause in Rule
8.       If ($1^{st}$ time in Rule & CQ [$A^R$] $\neq$ null )
9.         { $A^I$ = CQ [$A^R$]
10.          Clause=Clause in Rule which contains first $A^I$
11.          switch ( $A^I$.state)
12.            Case Pending: Exit Rule
13.            Case Denied: Clause = Next clause
14.            Case Offered: Clause = Clause
         } // End of If from line 9
15.      Next_clause = False
16.      While (Clause || Next_clause = = False)
17.        { $A^I$ = Next credential in Clause
18.          If ($A^I$ ~ rlist)
19.               $A^I$ REQSET
20.               CQ [$A^R$] = $A^I$
21.               Exit Rule        // End of If
22.          If ($A^I$.state = =Denied)
23.               If (ARC [$A^I$] > GARC )
24.                   $A^I \in$ REQSET
25.                   CQ [$A^R$] = $A^I$
26.                   Exit Rule
27.          Else   // Else for if from line 22
28.               Next_clause = True
         } // End of While from line 16
29.     If (Next_clause = = False)
30.        $A^R \in$ RELSET
31.        LURA [$A^R$] = Current Clause
32.        GARC = GARC + 1
33         Exit Rule
    }   // End of While from line 6
34. If (Next_clause = = True)
35.    ARC [$A^R$] = GARC
  } // End of Else from line 5.
36.Return ({RELSET}, {REQSET}) }
```

Fig 3. Driver Process and Resolve Process in SPAN

A. Driver Process

The input to SPAN_State_Machine, in figure 3, is the responder's release policy and the newly received message from Initiator. It's output (line 22) is four sets- (1) those ready to be released (NEW_RELSET), (2) pending (NEW_PENSET), (3) those he would like to deny (NEW_DENSET), and (4) a list of new resources he would like to counter request (NEW_REQSET).

This routine calls the SPAN_Rule_Resolver routine (line 16) for each requested resource in the message list. Resolver routine determines if the resource could be released and if not, what other resources need to be counter requested, or if the resource has to be denied. Resolver routine's inputs are (i) a resource in the set of old requests (OLD_REQSET), (ii) resource lists in the message, and (iii) the rules particularly linked to selected resource from old requests set. The routine changes the states of the resources and it generates new release set (RELSET) which contains resources that need to be released (by owner) and the counter request set (REQSET) which contains resources that need to be requested from opposite party.

Corresponding to the four new sets, SPAN_State_Machine maintains four copies of old sets (OLD_REQSET, OLD_RELSET, OLD_DENSET, and OLD_PENSET) extracted directly from an incoming message (lines 1-4). The routine checks (lines 11-13) old sets to see if the negotiation has resulted in a deal. Negotiation becomes successful if Target Resource is now available in OLD_RELSET. Negotiation becomes unsuccessful if Target Resource is in OLD_DENSET (lines 7-9). If above conclusions cannot be made, then the Resolver routine is called (line 16) to process every resource in OLD_REQSET. Resolver routine generates two sets - REQSET and RELSET. These new REQSET, RELSET provided by the Resolver routine and already existed old sets generate the new sets (NEW_REQSET, NEW_RELSET, NEW_PENSET, and NEW_DENSET) for the outgoing message (lines 17-21).

B. Resolver Process

SPAN_Rule_Resolver uses following important variables. 1) ***GARC (Global Attribute Release Count):*** Total number of resources released by both parties. 2) ***ARC (Attribute Release Count):*** When a resource is denied (by owner), its ARC is used to save the current GARC value. 3) ***CQ (Counter Request):*** Resource asked as a counter request for each pending resource.4) ***LURA (Clause that causes release of attribute):*** A clause in a rule that has resulted in release of requested attribute and it is used to trace back the release sequence in second phase of proxy negotiation.

A rule in a policy is represented in the disjunctive normal form (DNF) as $R_1 \leftarrow C_1 \vee C_2 \ldots C_j$, where, clause $C_1 = I_1 \wedge I_2 \ldots. I_K$, which means resource R_1 will be released when either of clauses C_1, C_2, or C_j is satisfied. Here, clause C_1 requires all resources I_1, I_2 and I_K from the other side. Each rule and clause is processed from left to right.

Resolver routine considers requested resources one by one and the rules associated with that resource and processes as explained below: **STEP-1)** If a requested resource doesn't have CQ, means it is being requested for the first time, routine executes (lines 6-7, lines 15-21, lines 29-33) as explained next **:** If the first resource in requested resource's first clause is not released and not pending, then this first resource will be counter requested (REQSET) and stored in requested resource's CQ (lines 17-21). But if this first resource is already released (by opposite party), then the next resource in the same clause is counter requested and so on until all resources in one clause are exhausted. If all resources in one clause are released, which can be found out from Resource List available in the body of the message, then requested resource can be released. When released, GARC is incremented by 1 and current clause is saved in the LURA of the released resource (lines 29-33) **STEP-2)** If a requested resource already has CQ, then routine executes (lines 6-12 and lines 6-21, 29-33) as explained next **: 2.A**) If CQ of requested resource is still pending, then the routine will come out of the current rule and keep CQ of requested resource as it is (lines 6-12) **2.B**) If CQ of requested resource, however, has already been released by opposite party, then as explained in STEP-1 next resource from the same clause is considered for CQ and so on. **STEP-3)** If after STEP-1 or 2, there is a new counter request and if this new counter request (which is different from old counter request, if any) is already pending, then it will skip that clause and move on to next clause. This is because it indicates a *cyclic dependency* which cannot be resolved with current set of released resources. If rest of the clauses too has at least one resource whose request is already pending, then the requested resource is denied and the GARC is saved in its ARC (lines 34-35).**STEP-4)** Before finally placing the counter requests, the routine will check if that resource has been denied by the other side. If this new counter request resource had been denied then (line 22-28, 34-35): **4.A**) that resource become CQ only if it's ARC > GARC (lines 22-26). This is because, there are more releases now since the last denial - so previous cycle may not be a problem anymore. **4. B**) that resource doesn't become CQ if it's ARC = GARC. This is because nothing has changed since last denial. In this case, the clause containing that resource is skipped for next clause (line 28) to repeat STEP-1 or 2.

C. Release Process

At the end of successful proxy negotiation, message carries release clause for every negotiated attribute. Using this information each party generates logical dependency graph to find out sequence for exchanging resource's value. Based on dependency graph, both parties release resources as explained by Offered_Set routine.

Offered_Set routine executing at each end of negotiation finds out which resources can be offered to opposite party using i) logical dependency graph and ii) resources offered by opposite party (i.e. . old_offeredset) till that time of resource release phase, if any.

```
LDgraph_Generator (rlist) {
1.  release [] = {R Є rlist: {R. state = AVAILABLE or R. state = OFFERED}}
2.  lura_count = sizeof (LURA)
3.  For ( I =0; I < lura_count; I ++)
4.   { ldgraph [I] [0] = relble [I]
5.      lura [] = {resources that caused release of release[I]}
6.      luracount = sizeof (lura[])
7.      temp =0
8.     for (J=1; J<= luracount; J++)
9.        ldgraph [I] [J] = lura[temp++]
     }
11.  return ldgraph   }
Offered_Set (old_offeredset, ldgraph) {
1. count = sizeof (ldgraph [])
2. offer_count = sizeof (old_offeredset)
3. If (offer_count = = 0)
4.  { For (I=0; I <count; I++)
5.     { count1= sizeof (ldgraph[I])
6.       If((ldgraph[I][1]= = ~) AND (count1= =2))
7.           new_offeredset ={new_offeredset}U{ldgraph [I][0]}
     } // End of for from line 5
   }
8. Else
9.  {For (I=0; I<offer_count; I++)
10.   For (J=0;J<count; J++)
12.     {rel_count= sizeof(ldgraph [J])
12.     Temp[] = {all elements from ldgraph [J][1] to ldgraph [J][rel_count]}
13.     If (all elements in Temp[] are present in  old_offeredset)
14.         new_offeredset = {new_offeredset} U ldgraph [J] [0]
       }
   }
15.  return new_offeredset;  }
```

Fig 4. Dependency Graph Generator and Offered _Set routine in SPAN

LDGraph_generator routine generates dependency graph using *rlist* as input. A node in dependency graph is a resource whose state of negotiation is *offered* or *available* and other nodes are solved clause of its disclosure policy (with one edge from each resource in that clause connecting to *offered* node). This way resources in dependency graph are represented (lines 2-9). At the start of second phase, dependency graph is generated from the resources whose state of negotiation is *available* because none of the parties have offered any resources yet. But once parties start offering resources to each other, they will move state of negotiation of the resources from *available* to *offered*, and then Logical Dependency Graph is generated from resources whose state of negotiation is either *available* or *offered* (line 1).

In offered_set routine, one of the parties will offer resources from dependency graph which don't have disclosure policy (lines 3-7). After this, opposite party will

Attribute	Alias	Value	Policy
Company Name	(I_1)	ABC Inc	~
Job Location	(I_2)	Xyz, OH	$(R_3 \wedge R_{10} \wedge R_2)$
Job Title	(I_3)	Soft Engg.	$(R_2 \wedge R_{10})$
Job Profile	(I_4)	Resp.html	$(R_9 \wedge R_2 \wedge R_{10})$
Salary	(I_5)	50k	$(R_2 \wedge R_7)$
Benefits	(I_6)	Benef.htm	~
401	(I_7)	Yes	$(R_2 \wedge R_{10})$
Bonus	(I_8)	Bns.html	$(R_2 \wedge R_{10} \wedge R_3)$
Visa sponorship	(I_9)	Yes	~
Start Date	(I_{10})	M/D/Y	R_3

Attribute	Alias	Value	Policy
Company Name	(I_1)	CDE Inc	(R_2)
Job Location	(I_2)	Xyz, OH	(R_8)
Job Title	(I_3)	Soft Engg.	$(R_2 \wedge R_7)$
Job Profile	(I_4)	Resp.html	$(R_2 \wedge R_5 \wedge R_8)$
Salary	(I_5)	50k	$(R_7 \wedge R_2 \wedge R_8)$
Benefits	(I_6)	Benef.htm	~
401	(I_7)	Yes	$(R_2 \wedge R_8 \wedge R_3)$
Bonus	(I_8)	Bns.html	$(R_2 \wedge R_3 \wedge R_8)$
Visa sponorship	(I_9)	Yes	~
Start Date	(I_{10})	M/D/Y	~

Attribute	Alias	Value	Policy
Company Name	(I_1)	KLM Inc	$(R_2 \wedge R_7)$
Job Location	(I_2)	Xyz, OH	$(R_2 \wedge R_7 \wedge R_8)$
Job Title	(I_3)	Soft Engg.	$(R_7) \vee (R_2)$
Job Profile	(I_4)	Resp.html	$(R_2 \wedge R_7 \wedge R_9)$
Salary	(I_5)	50k	$(R_2 \wedge R_7 \wedge R_{10})$
Benefits	(I_6)	Benef.htm	~
401	(I_7)	10k	$(R_2 \wedge R_5)$
Bonus	(I_8)	Bns.html	$(R_2 \wedge R_9 \wedge R_3)$
Visa sponorship	(I_9)	No	~
Start Date	(I_{10})	M/D/Y	R_{11}

Attribute	Alias	Value	Policy
Interview	(R_1)	Yes	$(I_1 \wedge I_6 \wedge I_5)$
Name	(R_2)	Alice	~
Email	(R_3)	alice@k.edu	$(I_1 \wedge I_3 \wedge I_{10})$
Cell Phone	(R_4)	123123123	(I_4)
SSN	(R_5)	000000000	$(I_1 \wedge I_3 \wedge I_5)$
School	(R_6)	KSU	(I_3)
Major	(R_7)	Comp-Sci	~
Experience	(R_8)	Expr.html	$(I_1 \wedge I_{10})$
Transcripts	(R_9)	Tran.html	$(I_1 \wedge I_3 \wedge I_4)$
Publications	(R_{10})	Publ.html	~

Attribute	Alias	Value	Policy
Interview	(R_1)	Yes	$(I_2 \wedge I_1 \wedge I_{10})$
Name	(R_2)	Sajid	~
Email	(R_3)	sajid@k.edu	$(I_1 \wedge I_3)$
Cell Phone	(R_4)	123123123	(I_4)
SSN	(R_5)	000000000	$(I_1 \wedge I_2 \wedge I_5)$
School	(R_6)	KSU	(I_1)
Major	(R_7)	Comp-Sci	$(I_1 \wedge I_{11})$
Experience	(R_8)	Expr.html	$(I_5 \wedge I_6) \vee (I_6)$
Transcripts	(R_9)	Tran.html	$(I_1 \wedge I_3 \wedge I_4)$
Publications	(R_{10})	Publ.html	~

Attribute	Alias	Value	Policy
Interview	(R_1)	Yes	$(I_3 \wedge I_1)$
Name	(R_2)	Pooja	~
Email	(R_3)	pooja@k.edu	$(I_1 \wedge I_2)$
Cell Phone	(R_4)	123123123	(I_4)
SSN	(R_5)	000000000	$(I_1 \wedge I_2 \wedge I_5)$
School	(R_6)	KSU	~
Major	(R_7)	Comp-Sci	(I_3)
Experience	(R_8)	Expr.html	$(I_5 \wedge I_4)$
Transcripts	(R_9)	Tran.html	$(I_1 \wedge I_3 \wedge I_4)$
Publications	(R_{10})	Publ.html	(I_1)

Fig 5. Access Control Policies of ABC Inc/John, CDE Inc/Yang, KLM Inc/Bob, Alice, Sajid, and Pooja respectively from left-right and top-bottom.

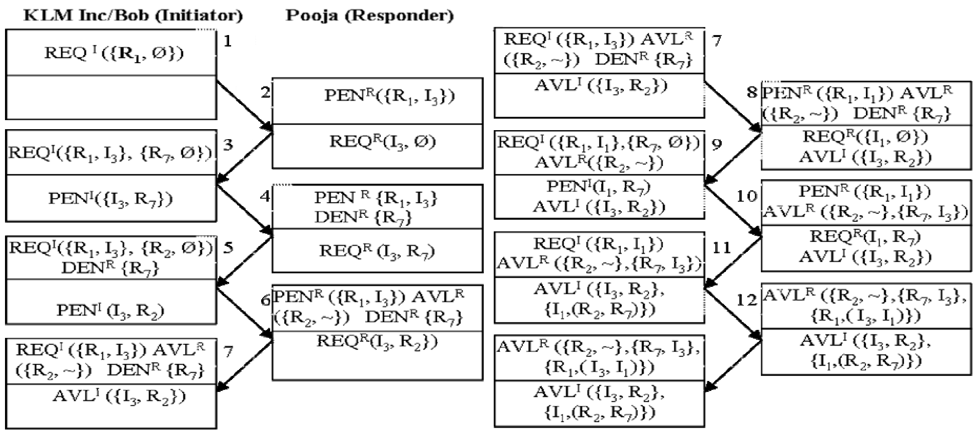
offer his resource from dependency graph according to offered resources it has got from first party (line 8-14). Thus two parties keep on offering resources to each other till Negotiation Requester gets his Target Resource.

IV. NEGOTIATION EXAMPLE

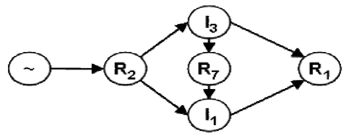
We consider a job seeker-hunter scenario in a complex federation setup - group of Universities and Companies. Students store *Individualized Policies* with their universities and various managers in the companies set up their requirements to search potential employees. In this setup, we consider three students Alice, Pooja and Sajid with home institution KSU and three hiring managers Bob, John, and Yang from KLM Inc, ABC Inc, and DEF Inc respectively with release policies in figure 5. Here, we will discuss one negotiation in detail where Negotiation Agent of KLM Inc initiates a search for potential candidate (Pooja) to fill-up the vacancy under Bob. Before that, let's understand state information carried by each negotiation message.

In our notation we group resources against their respective current states in a negotiation as STATE^{P} ($\{R_1{:}V_1\}$, $\{R_2{:}V_2\}$…. $\{Rn{:}\ Vn\}$). STATE can be the states of a resources specified in [4] such as REQ, AVL, PEN, DEN, OFF etc. Each argument is a pair where Rn is the ID of the resource and Vn is the special information about the resource. The superscript p represents the party (negotiation initiator (I) or negotiation responder (R)) involved in the latest update of a state. Top portion and bottom portion of each message corresponds to IRL and RRL as described in section 2.2.

KLM's agent starts negotiation by requesting *Interview* i.e. (REQ^{I} ($\{R_1, \varnothing\}$) from Pooja's agent in message 1 in fig 6 A).Here, Ø indicates that R_1 has no counter request (CQ). Bottom portion of this message i.e. Pooja's dataspace is empty. In response to message 1, Pooja's agent will counter request *Job_Title* (REQ^{R} $\{I_3, \varnothing\}$) in message 2 and also changes state of negotiation of *Interview* from *requesting* to *pending* i.e.


KLM Inc/Bob (Initiator)
Pooja (Responder)
1
$\text{REQ}^{I}(\{R_1, \varnothing\})$
2
$\text{PEN}^{R}(\{R_1, I_3\})$
$\text{REQ}^{R}(I_3, \varnothing)$
3
$\text{REQ}^{I}(\{R_1, I_3\}, \{R_7, \varnothing\})$
$\text{PEN}^{I}(\{I_3, R_7\})$
4
$\text{PEN}^{R}\{R_1, I_3\}$ $\text{DEN}^{R}\{R_7\}$
$\text{REQ}^{R}(I_3, R_7)$
5
$\text{REQ}^{I}(\{R_1, I_3\}, \{R_2, \varnothing\})$ $\text{DEN}^{R}\{R_7\}$
$\text{PEN}^{I}(I_3, R_2)$
6
$\text{PEN}^{R}(\{R_1, I_3\})$ $\text{AVL}^{R}(\{R_2, \sim\})$ $\text{DEN}^{R}\{R_7\}$
$\text{REQ}^{R}(I_3, R_2\})$
7
$\text{REQ}^{I}(\{R_1, I_3\})$ $\text{AVL}^{R}(\{R_2, \sim\})$ $\text{DEN}^{R}\{R_7\}$
$\text{AVL}^{I}(\{I_3, R_2\})$
7
$\text{REQ}^{I}(\{R_1, I_3\})$ $\text{AVL}^{R}(\{R_2, \sim\})$ $\text{DEN}^{R}\{R_7\}$
$\text{AVL}^{I}(\{I_3, R_2\})$
8
$\text{PEN}^{R}(\{R_1, I_1\})$ $\text{AVL}^{R}(\{R_2, \sim\})$ $\text{DEN}^{R}\{R_7\}$
$\text{REQ}^{R}(\{I_1, \varnothing\})$ $\text{AVL}^{I}(\{I_3, R_2\})$
9
$\text{REQ}^{I}(\{R_1, I_1\}, \{R_7, \varnothing\})$ $\text{AVL}^{R}(\{R_2, \sim\})$
$\text{PEN}^{I}(I_1, R_7)$ $\text{AVL}^{I}(\{I_3, R_2\})$
10
$\text{PEN}^{R}(\{R_1, I_1\})$ $\text{AVL}^{R}(\{R_2, \sim\}, \{R_7, I_3\})$
$\text{REQ}^{R}(I_1, R_7)$ $\text{AVL}^{I}(\{I_3, R_2\})$
11
$\text{REQ}^{I}(\{R_1, I_1\})$ $\text{AVL}^{R}(\{R_2, \sim\}, \{R_7, I_3\})$
$\text{AVL}^{I}(\{I_3, R_2\}, \{I_1, (R_2, R_7)\})$
12
$\text{AVL}^{R}(\{R_2, \sim\}, \{R_7, I_3\}, \{R_1, (I_3, I_1)\})$
$\text{AVL}^{I}(\{I_3, R_2\}, \{I_1, (R_2, R_7)\})$
$\text{AVL}^{R}(\{R_2, \sim\}, \{R_7, I_3\}, \{R_1, (I_3, I_1)\})$
$\text{AVL}^{I}(\{I_3, R_2\}, \{I_1, (R_2, R_7)\})$

(A)



(B)

Attributes/Resources Received By			
KLM Inc (Initiator)		**Pooja (Responder)**	
Attribute	*Value*	*Attribute*	*Value*
Name	Pooja	Company Name	KLM, Inc
Major	Comp-Sci	Job Title	Soft Engg
Interview	Yes	---------	

(C)

Fig 6. A) Messages exchanged during a negotiation between KLM Inc and Pooja to schedule an Interview using Proxy strategy. Messages are shown from 1-12. B) Dependency Graph resulted from negotiation C) At the end of Negotiation; each SA will store resources information for its user indicating attributes released and received.

Negotiating Parties	Negotiation Result	Number of Rules fired in a Negotiation		Number of Messages exchanged in a Negotiation		Number of attributes released in a Negotiation	
		Stateless Eager	Stateless Proxy	Stateless Eager	Stateless Proxy	Stateless Eager	Stateless Proxy
ABC-Alice	Success	10	6	4	12	10	6
ABC-Pooja	Success	9	5	4	10	9	5
ABC-Sajid	Success	14	8	6	16	14	8
CDE-Alice	Success	15	8	6	16	15	8
CDE-Pooja	Fail	7	4	4	8	7	0
CDE-Sajid	Success	9	6	4	16	9	6
KLM-Alice	Success	10	7	6	14	10	7
KLM-Pooja	Success	8	5	6	12	8	5
KLM-Sajid	Fail	4	5	4	10	6	0

Fig. 7. Figure shows statistics of nine negotiations using SEAN, and SPAN algorithm.

PEN^R ({R_1, I_3}). Message 3 shows request of R_7 i.e. REQ^I ({R_1, I_3}, {$\mathbf{R_7}$, Ø} and as per Pooja's policy, she needs I_3 to unlock R_7. But state of negotiation of I_3 is already pending and it requires R_7 to unlock. This creates a *cyclic dependency*. To resolve this, Pooja's agent will deny R_7 i.e. DEN^R (R_7, 0) in message 4. Here, 0 is ***GARC*** when R_7 is denied. In response to message 4, KLM's agent will find out that R_7 is counter request of I_3. So even though R_7 is denied, I_3 can still be unlocked using R_2. So KLM's agent will request R_2 in message 5. This process of searching an alternative path to unlock an attribute (I_3 in this case) and thus breaking cyclic dependency is called as *backtracking*. In backtracking, a denied attribute is requested again only if number attributes released by both participants (***GARC***) are more since last denial of the same attribute. As per this rule R_7 is again requested by KLM's agent in message 9. Here, GARC=1 in message 8 is more than ***GARC*** = 0 when R_7 was denied in message 4.

Finally, in message 12 Pooja's agent offers Interview i.e. AVL^R ({R2, ~}, {R_7, I_3}, {$\mathbf{R_1}$, {I_3, I_1}} to KLM's agent to finish first phase of negotiation. In second phase, agents will exchange attribute values using dependency graph in figure 6 B). At the end of second phase, agents will generate negotiation result for their participants in 6 C).

V. Analysis of SEAN and SPAN

Figure 7 shows results of negotiation among three organizations, and three students (policies shown in fig 5) for SEAN and SPAN. In successful negotiation, SEAN releases more attributes and fires more number of rules than SPAN but with fewer messages. In unsuccessful negotiation (CDE-Pooja) SEAN releases few attributes, but SPAN doesn't release any attribute. Here, users which are eager to reach a deal can opt for SEAN at the cost of disclosing more attributes while cautious users, with sensitive attributes, could opt for SPAN to avoid unnecessary disclosure of the attributes.

Algorithm	Worst case Communication Complexity	
	Number of messages	Size of a message
SPAN	$O(n^2)$	$O(n)$
SEAN	$O(n)$	$O(n)$

Fig. 8. Worst case Communication complexity, where n is number of attributes offered in a negotiation.

Communication Complexity of SPAN and SEAN [4] is shown in figure 8.

VI. Conclusion

We have contributed a novel negotiation enabled framework and protocol to enhance privacy of federated members. This framework facilitates federated members to choose negotiation flavor on per-need-basis. Two such flavors are Stateless Eager Attribute Negotiation (SEAN) and Stateless Proxy Attribute Negotiation (SPAN). SEAN results in successful negotiation with minimum communication steps while SPAN discloses only necessary attributes for negotiation.

The privacy implications of this work are quite interesting. With Federated Access Control System a member is no longer at the mercy of Service Provider about the disclosure policy, but still is at the mercy of the on-size-fits-all release policy set by the home organization. Through this work the individual's privacy is further enhanced - a member is not at the mercy of home organization.

References

[1] Kathy Bohrer, Stephen Levy, "Individualized Privacy Policy Based Access Control ," In Proceedings 6th International Conference on Electronic Commerce Research (ICECR-6) October 2003, Dallas, Texas, USA.

[2] T. Yu, X. Ma, and M. Winslett., "PRUNES: An Efficient and Complete Strategy for Trust Negotiation over the Internet," ACM Conference on Computer and Communications Security, Athens, November 2000.

[3] Abhilasha Bhargav, Anna C. Squicciarini, "Trust Negotiation in Identity Management", IEEE Security & Privacy, March-April 2007.

[4] J. Khan, K. Bobade, M. Hardas, "Intra-Federation credential Negotiation Based on Individualized Release Strategy," International Association of Science and Technology Development, Canada, July 2007.

[5] Shibboleth. [Online] http://shibboleth.internet2.edu/shib-tech-intro.html

[6] Active Directory Federation Services. [Online] http://www.microsoft.com/WindowsServer2003/R2/Identity_Management/ADFSwhitepaper.mspx

[7] Alan F. Westin (1970) *Privacy and Freedom*