IT University of Copenhagen

Bachelor Project

Verifiable Secure Open Source Alternative to NemID

Authors:

Andreas Hallberg Kjeldsen
ahal@itu.dk

Morten Chabert Eskesen
mche@itu.dk

Supervisor:

Joseph Roland Kiniry
josr@itu.dk

May 22, 2013

Abstract

Your abstract goes here...

Contents

Chapter 1

Introduction

. . .
    We're extending the work done by Jacob Hjgaard in his Masters Thesis 'Securing Single Sign-On Systems With Executable Models'. Jacobs research has focused on the current implementation of NemID and therefore describes, outlines and models the current system used in Denmark as of May 2013.

1.1   Objectives

Some explaining text here

1. Describe and outline the OpenNemID protocol, including but not limited to registration and login.

2. Formalize the specification of OpenNemID in F* to the extent possible.

1.2   Scope

This project has had it focus towards specifying a new protocol that could replace NemID. The intent of this project is therefore not to develop a complete system, but to make the specification for a system that could then later be developed based on the specification.

1.3   Background

. . .

Chapter 2

Technical Background

2.1 SAML Protocol

2.2 Static Analysis

2.3 Selection of verification tool

2.4 N Factor Authentication

Videreudvikling af two factor authentication
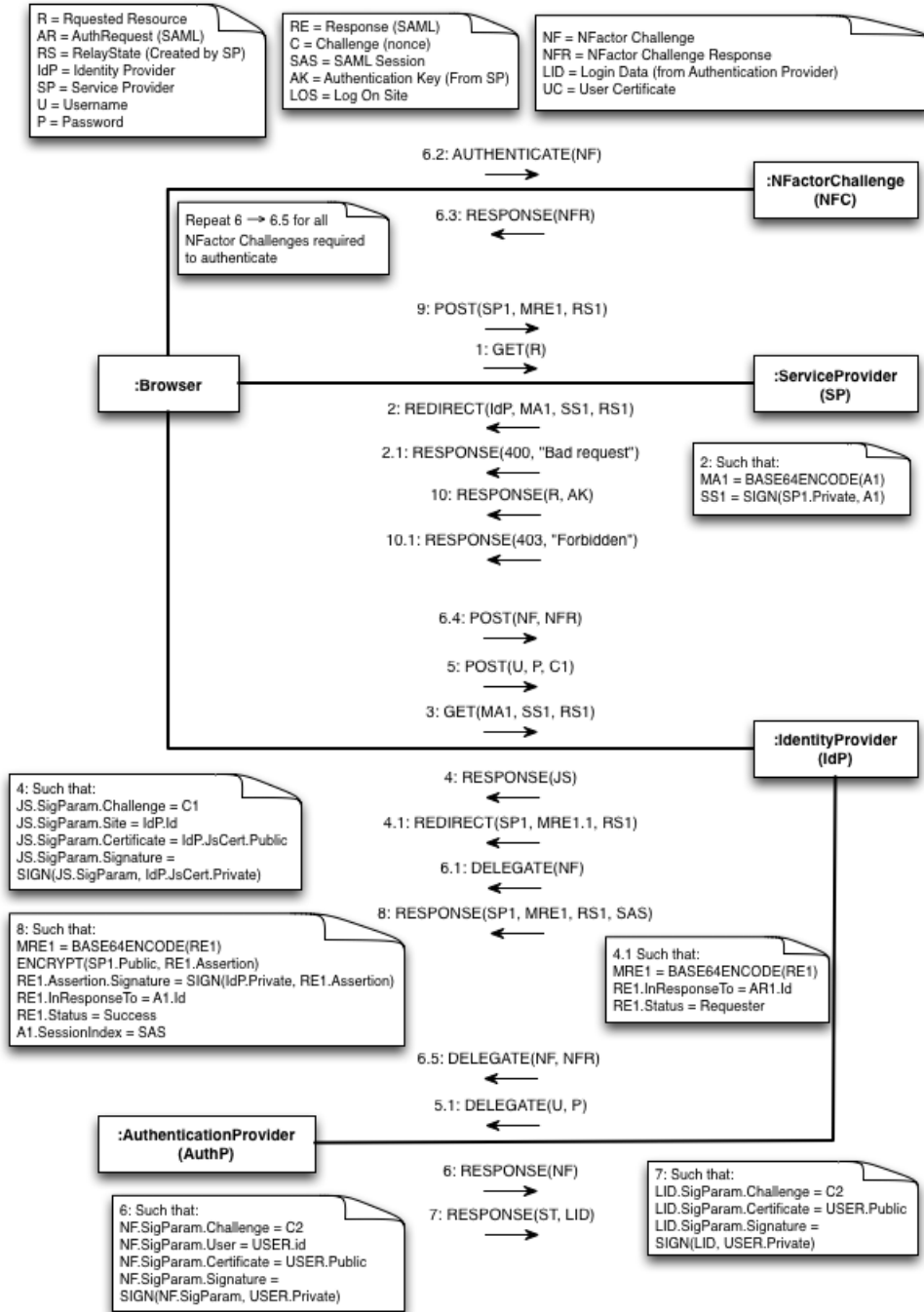
Chapter 3

Remodelling the protocol

3.1 How It Is Today

3.2 How It Could Be

3.3 Communication Model

The communication model displays a graphical overview of how data should be communicated between the involved parties.

R = Rquested Resource
AR = AuthRequest (SAML)
RS = RelayState (Created by SP)
IdP = Identity Provider
SP = Service Provider
U = Username
P = Password

RE = Response (SAML)
C = Challenge (nonce)
SAS = SAML Session
AK = Authentication Key (From SP)
LOS = Log On Site

NF = NFactor Challenge
NFR = NFactor Challenge Response
LID = Login Data (from Authentication Provider)
UC = User Certificate

6.2: AUTHENTICATE(NF)

:NFactorChallenge
(NFC)

Repeat 6 → 6.5 for all
NFactor Challenges required
to authenticate

6.3: RESPONSE(NFR)

9: POST(SP1, MRE1, RS1)

1: GET(R)

:Browser

:ServiceProvider
(SP)

2: REDIRECT(IdP, MA1, SS1, RS1)

2.1: RESPONSE(400, "Bad request")

2: Such that:
MA1 = BASE64ENCODE(A1)
SS1 = SIGN(SP1.Private, A1)

10: RESPONSE(R, AK)

10.1: RESPONSE(403, "Forbidden")

6.4: POST(NF, NFR)

5: POST(U, P, C1)

3: GET(MA1, SS1, RS1)

:IdentityProvider
(IdP)

4: RESPONSE(JS)

4: Such that:
JS.SigParam.Challenge = C1
JS.SigParam.Site = IdP.Id
JS.SigParam.Certificate = IdP.JsCert.Public
JS.SigParam.Signature =
SIGN(JS.SigParam, IdP.JsCert.Private)

4.1: REDIRECT(SP1, MRE1.1, RS1)

6.1: DELEGATE(NF)

8: RESPONSE(SP1, MRE1, RS1, SAS)

8: Such that:
MRE1 = BASE64ENCODE(RE1)
ENCRYPT(SP1.Public, RE1.Assertion)
RE1.Assertion.Signature = SIGN(IdP.Private, RE1.Assertion)
RE1.InResponseTo = A1.Id
RE1.Status = Success
A1.SessionIndex = SAS

4.1 Such that:
MRE1 = BASE64ENCODE(RE1)
RE1.InResponseTo = AR1.Id
RE1.Status = Requester

6.5: DELEGATE(NF, NFR)

5.1: DELEGATE(U, P)

:AuthenticationProvider
(AuthP)

6: RESPONSE(NF)

7: Such that:
LID.SigParam.Challenge = C2
LID.SigParam.Certificate = USER.Public
LID.SigParam.Signature =
SIGN(LID, USER.Private)

7: RESPONSE(ST, LID)

6: Such that:
NF.SigParam.Challenge = C2
NF.SigParam.User = USER.id
NF.SigParam.Certificate = USER.Public
NF.SigParam.Signature =
SIGN(NF.SigParam, USER.Private)

TEXT DESCRIBING ALGORITHM 1

Algorithm 1 Process 1

Require: GET is well-formed **and** IdP.Public **and** SP.Private
  if R exists **then**
    AR ← CreateAuthnRequest()
    SAR ← SIGN(AR, SP.Private)
    MA ← UrlEnc(Base64Enc(DeflateCompress(AR)))
    RS ← UrlEnc(Base64Enc(R))
    return REDIRECT(IdP, MA, SAR, RS)
  else
    return RESPONSE(400, BadRequest)
  end if

TEXT DESCRIBING ALGORITHM 2

Algorithm 2 Process 3

Require: GET is well-formed **and** IdP.Private **and** SP.Public **and** IdPJsCert.Public **and** IdP has JavaScript from AuthP
  AR ← DeflateDecompress(Base64Dec(UrlDec(MA)))
  if VERIFY(AR, SAR, SP.Public) **then**
    C1 ← GenChallenge()
    JS ← StoredJavaScript()
    JS.SigParams.Challenge ← C1
    JS.SigParams.Certificate ← IdPJsCert.Public
    JS.SigParams.Signature ← SIGN(JS.SigParams, IdPJsCert.Private)
    return RESPONSE(JS)
  else
    RE ← CreateResponse()
    RE.InResponseTo ← AR
    RE.Status ← Requester
    MRE ← Base64Enc(RE)
    return REDIRECT(SP, MRE, RS)
  end if

TEXT DESCRIBING ALGORITHM 3

Algorithm 3 Process 4

Require: U **and** P **and** Browser allows JavaScript
  SigParams ← Js.SigParams
  if VERIFY(SigParams, SigParams.Signature, SigParams.Certificate) **then**
    C1 ← SigParams.Challenge
    return POST(U, P, C1)
  else
    print ERROR
  end if

TEXT DESCRIBING ALGORITHM 4

Algorithm 4 Process 5

Require: POST is well formed
  if C1 matches challenge issued by IdP **then**
    Delegate U **and** P **to** AuthP
  else
    return RESPONSE(ERROR)
  end if
Require: C1 matches challenge issued by IdP

TEXT DESCRIBING ALGORITHM 5

Algorithm 5 Process 5.1

  USER $\leftarrow$ GetUser(U, P)
  if USER is valid **then**
    C2 $\leftarrow$ GenChallenge()
    NF $\leftarrow$ GetNextNFactorChallenge(USER)
    NF.SigParam.User $\leftarrow$ USER
    NF.SigParam.Challenge $\leftarrow$ C2
    NF.SigParam.Certificate $\leftarrow$ USER.Public
    NF.SigParam.Signature $\leftarrow$ SIGN(NF.SigParam, USER.Private)
    return RESPONSE(NF)
  else
    return RESPONSE(ERROR)
  end if

TEXT DESCRIBING ALGORITHM 6

Algorithm 6 Process 6

  SigParams $\leftarrow$ NF.SigParams
  if VERIFY(SigParams, SigParams.Signature, SigParams.Certificate) **then**
    RELATE(SigParams.User, SigParams.Challenge)
    Delegate NF **to** Browser
  else
    Delegate ERROR **to** Browser
  end if

TEXT DESCRIBING ALGORITHM 7

Algorithm 7 Process 6.1

SigParams ← NF.SigParams
if VERIFY(SigParams, SigParams.Signature, SigParams.Certificate) **then**
  AUTHENTICATE(NF)
else
  print ERROR
end if

TEXT DESCRIBING ALGORITHM 8

Algorithm 8 Process 6.2

NFR ← NFactorResult(NF)
return RESPONSE(NFR)

TEXT DESCRIBING ALGORITHM 9

Algorithm 9 Process 6.5

Require: Stored relation for (NF.SigParams.USER, NF.SigParams.Certificate)
SigParams ← NF.SigParams
if VERIFY(SigParams, SigParams.Signature, SigParams.Certificate) **then**
  if NFR is acceptable result of NF **then**
    USER ← GetUser(SigParams.USER, SigParams.Certificate)
    C2 ← GenChallenge()
    if USER.HasNextChallenge **then**
      NF ← GetNextNFactorChallenge(USER)
      NF.SigParams.User ← USER
      NF.SigParams.Challenge ← C2
      NF.SigParams.Certificate ← USER.Public
      NF.SigParams.Signature ← SIGN(NF.SigParams, USER.Private)
      return RESPONSE(NF)
    else
      LID ← CreateLogInData()
      ST ← OK
      return RESPONSE(ST, LID)
    end if
  else
    return RESPONSE(ERROR)
  end if
else
  return RESPONSE(ERROR)
end if

TEXT DESCRIBING ALGORITHM 10

Algorithm 10 Process 7

Require: SP.Public **and** LID is well-formed **and** stored AuthRequest for
(LID.User, LID.Challenge)

  if ST = "OK" **then**

    ARC ← GetAuthRequest(LID.User, LID.Challenge)

    MA ← ARC.AR

    SAR ← ARC.SAR

    RS ← ARC.RS

    AR ← DeflateDecompress(Base64Dec(UrlDec(MA)))

    if VERIFY(AR, SAR, SP.Public) **then**

      A ← BuildAssertion(LID.Certificate)

      SI ← GenerateSessionIndex()

      A.InResponseTo ← AR

      A.Issuer ← IdP

      A.Audience ← SP

      A.SessionIndex ← SI

      A.Signature ← SIGN(A, IdP.Private)

      EA ← ENCRYPT(A, SP.Public)

      RE ← CreateResponse()

      RE.Assertion ← EA

      RE.InResponseTo ← AR

      RE.Status ← "Success"

      MRE ← DeflateCompress(Base64Enc(UrlEnc(RE)))

      SAS ← CreateSAMLSession(SI, SP, LID.CertificateSubject)

      return REDIRECT(SP, MRE, RS, SAS)

    else

      RE ← CreateResponse()

      RE.InResponseTo ← AR

      RE.Status ← "Requester"

      MRE ← DeflateCompress(Base64Enc(UrlEnc(RE)))

      return REDIRECT(SP, MRE, RS)

    end if

  else

    return RESPONSE(ST)

  end if

TEXT DESCRIBING ALGORITHM 11

Algorithm 11 Process 9

Require: POST is well.formed **and** SP.Private **and** IdP.Public

  RE ← UrlDec(Base64Dec(DeflateDecompress(MRE)))

  A ← DECRYPT(RE.Assertion, SP.Private)

  if VERIFY(A, A.Signature, IdP.Public) **then**

    AK ← GenAuthKey()

    R ← Base64Dec(UrlDec(RS))

    RES ← GetResource(R)

    return RESPONSE(RES, AK)

  else

    return RESPONSE(403, Forbidden)

  end if

Chapter 4

Modelling with F*

4.1 Introducing F*

4.2 Syntax and semantics

4.3 Refinement types

4.4 Protocol specification in F*

hello world

4.4.1 Specification of the type functionality module

```
1  module TypeFunc
2
3  type Authentication =
4     | Facebook: id:int -> Authentication
5     | SMS: generated:int -> Authentication
6     | Google: id:int -> Authentication
7     | OpenId: id:int -> Authentication
```

Listing 4.1: TypeFunc module

4.4.2 Specification of the SAML Protocol

```
1  module SamlProtocol
2
3  open Crypto
4  open TypeFunc
5
6  type assertiontoken = string (*Add refinements*)
7  type signedtoken = string (*Add refinements*)
8  type id = string
```

```
9   type endpoint = string
10  type uri = string
11
12
13  type AuthnRequest =
14    | MkAuthnRequest: IssueInstant:string ->
15      Destination:endpoint -> Issuer:prin ->
16      message:string -> sig:dsig ->
17      AuthnRequest
18
19  type LoginData =
20    | MkLoginData:  user:prin -> signature:dsig ->
21      cert:pubkey user -> challenge:nonce ->
22      site:string -> data:string ->
23      LoginData
24
25  type LoginInfo =
26    | UserLogin:  userid:string -> password:string ->
27    LoginInfo
28
29  type AuthInfo =
30    | UserAuth:   userid:string -> authmethod:Authentication ->
31    authresponse:Authentication -> AuthInfo
32
33  type Assertion =
34    | SignedAssertion: assertiontoken -> dsig -> Assertion
35    | EncryptedAssertion: cypher -> Assertion
36
37  type SamlStatus =
38    | Success: SamlStatus
39    | Requester: SamlStatus
40    | Responder: SamlStatus
41    | User: SamlStatus
42
43  type SamlMessage =
44    | SPLogin: uri -> SamlMessage
45    | Login: loginInfo:LoginInfo -> challenge:nonce ->
46        SamlMessage
46    | LoginResponse: string -> SamlMessage
47    | AuthnRequestMessage: issuer:prin ->  destination:endpoint
          -> message:string -> dsig -> SamlMessage
48    | LoginRequestMessage: issuer:prin ->  destination:endpoint
          -> loginInfo:LoginInfo -> SamlMessage
49    | NfactAuthRequest: issuer:prin -> destination:endpoint ->
          authInfo:AuthInfo -> challenge:nonce -> dsig ->
          SamlMessage
50    | AuthResponseMessage: issuer:prin -> destination:endpoint ->
           Assertion -> SamlMessage
51    | LoginResponseMessage: issuer:prin -> destination:endpoint
          -> auth:Authentication -> challenge:nonce -> dsig ->
          SamlMessage
52    | UserAuthenticated: status:string -> logindata:LoginData ->
          authnReq:AuthnRequest -> SamlMessage
53    | UserCredRequest: javascript:string -> challenge:nonce ->
          dsig -> SamlMessage
54    | UserAuthRequest: authmethod:Authentication -> challenge:
          nonce -> dsig -> SamlMessage
```

12

```
55      | UserAuthResponse: authInfo:AuthInfo -> challenge:nonce ->
            dsig -> SamlMessage
56      | LoginSuccess: status:string -> issuer:prin -> destination:
            endpoint -> SamlMessage
57      | Failed: SamlStatus -> SamlMessage
58      | DisplayError: int -> SamlMessage
59
60
61   val SendSaml: prin -> SamlMessage -> unit
62   val ReceiveSaml: prin -> SamlMessage
63
64   val CreateAuthnRequestMessage: issuer:prin -> destination:prin
        -> string
65   val CreateLoginRequestMessage: issuer:prin -> destination:prin
        -> string
66   val CreateNfactAuthReqMessage: issuer:prin -> destination:prin
        -> string
67   val IssueAssertion: issuer:prin -> subject:prin -> audience:
        prin -> inresto:AuthnRequest -> assertiontoken
68   val MakeAssertion: issuer:prin -> subject:prin -> audience:prin
         -> assertiontoken
69   val AddSignatureToAssertion: assertiontoken -> dsig ->
        signedtoken
70   val EncryptAssertion: receiver:prin -> pubkey receiver ->
        signedtoken -> Assertion
71   val DecryptAssertion: receiver:prin -> privkey receiver ->
        Assertion -> (signedtoken * dsig)
```

Listing 4.2: SAML Protocol module

4.4.3 Specification of cryptographic elements

```
1    module Crypto
2
3    open Protocol
4    open TypeFunc
5
6    type prin = string
7    type pubkey :: prin => *
8    type privkey :: prin => *
9    type dsig
10   type nonce = string
11   type cypher
12
13   (*Verification*)
14   type Log :: prin => string => E
15
16   type LogAuth :: prin => Authentication => E
17
18   val Keygen: p:prin
19      -> (pubkey p * privkey p)
20
21   val Sign: p:prin
22    -> privkey p
23    -> msg:string{Log p msg}
```

```
24    -> dsig
25
26  val SignAuth: p:prin
27    -> privkey p
28    -> msg:Authentication{LogAuth p msg}
29    -> dsig
30
31  val VerifySignature: p:prin
32    -> pubkey p
33    -> msg:string
34    -> dsig
35    -> b:bool{b=true ==> Log p msg}
36
37  val VerifySignatureAuth: p:prin
38    -> pubkey p
39    -> msg:Authentication
40    -> dsig
41    -> b:bool{b=true ==> LogAuth p msg}
42
43  val Encrypt: p:prin
44    -> pubkey p
45    -> string
46    -> cypher
47
48  val Decrypt: p:prin
49    -> privkey p
50    -> cypher
51    -> string
52
53  val GenerateNonce: prin -> nonce (*Add refinement to ensure
        uniqueness*)
```

Listing 4.3: Crypto module

4.4.4 Specification of certificate store module

```
1  module CertStore
2
3  open Crypto
4
5  val GetPublicKey: p:prin -> pubkey p
6  val GetJSPublicKey: p:prin -> pubkey p
7  (*Prin needs to be updated to include credentials*)
8  val GetPrivateKey: p:prin -> privkey p
9  val GetJSPrivateKey: p:prin -> privkey p
```

Listing 4.4: CertStore module

4.4.5 Specification of the messaging protocol

```
1  module Messaging
2
3  open Crypto
```

```
4   open TypeFunc
5
6   type Status =
7     | Successful: Status
8     | Unsuccessful: Status
9
10  type Message =
11    | NewSiteRequest: idp:prin -> Message
12    | ChallengeResponse: challenge:nonce -> Message
13    | IdpChalResponse: challenge:nonce -> Message
14    | AcceptedIdp: idp:prin -> pubkey:pubkey idp -> authp:prin ->
          authpubkey:pubkey authp -> signedjavascript:string ->
          Message
15    | RequestForLogin: userid:string -> password:string -> Message
16    | ReqLoginResponse: challenge:nonce -> Message
17    | CreateLogin: generatedpassword:string -> challenge:nonce ->
          Message
18    | ChangeUserId: userid:string -> newUserId:string -> password:
          string -> Message
19    | ChangePassword: userid:string -> password:string ->
          newPassword:string -> Message
20    | UserRevokeIdp: userid:string -> password:string -> idp:
          string -> Message
21    | AddNfactor: userid:string -> password:string -> nfact:
          Authentication -> Message
22    | RemoveNfactor: userid:string -> password:string -> nfact:
          Authentication -> Message
23    | StatusMessage: Status -> Message
24
25
26  val SendMessage: prin -> Message -> unit
27  val ReceiveMessage: prin -> Message
```

Listing 4.5: Messaging module

4.4.6 Specification of the Service Provider

```
1   module Serviceprovider
2
3   open SamlProtocol
4   open Crypto
5
6   val serviceprovider:  me:prin -> client:prin -> idp:prin ->
        unit
7
8   let rec serviceprovider me client idp =
9    let req = ReceiveSaml client in
10   match req with
11     | SPLogin (url) ->
12       let authnReq = CreateAuthnRequestMessage me idp in
13       assume(Log me authnReq);
14       let myprivk = CertStore.GetPrivateKey me in
15       let sigSP = Sign me myprivk authnReq in
16       let resp = AuthnRequestMessage me idp authnReq sigSP in
17       SendSaml client resp;
```

```
18      serviceprovider me client idp
19   | AuthResponseMessage (issuer, destination, encassertion) ->
20     let myprivk = CertStore.GetPrivateKey me in
21     let assertion = DecryptAssertion me myprivk encassertion in
22     match assertion with
23     | SignedAssertion (token,sigIDP) ->
24       let pubkissuer = CertStore.GetPublicKey idp in
25       if VerifySignature idp pubkissuer token sigIDP
26       then
27         (assert(Log idp token);
28         let resp = LoginResponse "You are now logged in" in
29         SendSaml client resp)
30       else SendSaml client (DisplayError 403);
31       serviceprovider me client idp
32
33     | _ -> SendSaml client (DisplayError 400);
34         serviceprovider me client idp
```

Listing 4.6: ServiceProvider Module

4.4.7 Specification of the Identity Provider

```
1   module Identityprovider
2
3   open SamlProtocol
4   open Crypto
5   open TypeFunc
6   open Messaging
7
8   val userloggedin: user:prin -> bool
9   val getjavascript: string
10  val decodeMessage: message:string -> AuthnRequest
11  val getauthnrequest: user:prin -> challenge:nonce ->
        AuthnRequest
12  val getuserchallenge: user:prin -> nonce
13  val relatechallenge: user:prin -> challenge:nonce -> unit
14  val verifychallenge: user:prin -> challenge:nonce -> bool
15  val relate: user:prin -> challenge:nonce -> authnReq:
        AuthnRequest -> unit
16
17  val handleUserAuthenticated: me:prin -> user:prin -> authnReq:
        AuthnRequest -> unit
18
19  let handleUserAuthenticated me user authnReq =
20   match authnReq with
21   | MkAuthnRequest(issueinst,dest,sp,msg,sigSP) ->
22    let pubksp = CertStore.GetPublicKey sp in
23     if (VerifySignature sp pubksp msg sigSP) then
24    (assert (Log sp msg);
25    let assertion = IssueAssertion me user sp authnReq in
26    let myprivk = CertStore.GetPrivateKey me in
27    assume(Log me assertion);
28    let sigAs = Sign me myprivk assertion in
29    let signAssertion = AddSignatureToAssertion assertion sigAs
        in
```

```fstar
30    let encryptedAssertion = EncryptAssertion sp pubksp
          signAssertion in
31    let resp = AuthResponseMessage me sp encryptedAssertion in
32    SendSaml user resp)
33        else
34   SendSaml user (Failed Requester)

35
36  val handleauthresponse: me:prin -> user:prin -> authp:prin ->
        unit

37
38  let handleauthresponse me user authp =
39   let resp = ReceiveSaml authp in
40   match resp with
41   | LoginResponseMessage(issuer, destination, authmethod,
         challenge, sigUser) ->
42    let pubkeyuser = CertStore.GetPublicKey user in
43    if VerifySignatureAuth user pubkeyuser authmethod sigUser
          then
44     (assert (LogAuth user authmethod);
45     relatechallenge user challenge;
46     let resp = UserAuthRequest authmethod challenge sigUser in
47     SendSaml user resp)
48    else
49     SendSaml user (DisplayError 403)
50   | LoginSuccess(status, issuer, destination) ->
51    if (status = "OK") then
52     let challenge = getuserchallenge user in
53     let authnReq = getauthnrequest user challenge in
54     handleUserAuthenticated me user authnReq
55    else
56     SendSaml user (DisplayError 403)
57   | _ -> SendSaml user (DisplayError 400)

58
59  val identityprovider: me:prin -> user:prin -> authp:prin ->
        unit

60
61  let rec identityprovider me user authp =
62   let request = ReceiveSaml user in
63   match request with
64   | AuthnRequestMessage(issuer, destination, message, sigSP) ->
65    let pubkissuer = CertStore.GetPublicKey issuer in
66    if (VerifySignature issuer pubkissuer message sigSP) then
67     (assert (Log issuer message);
68     let authnReq = decodeMessage message in
69     let myprivk = CertStore.GetPrivateKey me in
70     if not (userloggedin user) then
71      let challenge = GenerateNonce me in
72      relate user challenge authnReq;
73      relatechallenge user challenge;
74      let js = getjavascript in
75      assume(Log me js);
76      let myprivk = CertStore.GetJSPrivateKey me in
77      let sigIdP = Sign me myprivk js in
78      let resp = UserCredRequest js challenge sigIdP in
79      SendSaml user resp;
80      identityprovider me user authp
81     else
```

17

```fsharp
82      let assertion = IssueAssertion me user issuer authnReq in
83      assume(Log me assertion);
84      let myprivk = CertStore.GetPrivateKey me in
85      let pubksp = CertStore.GetPublicKey issuer in
86      let sigAs = Sign me myprivk assertion in
87      let signAssertion = AddSignatureToAssertion assertion sigAs
           in
88      let encryptedAssertion = EncryptAssertion issuer pubksp
           signAssertion in
89      let resp = AuthResponseMessage me issuer encryptedAssertion
           in
90      SendSaml user resp)
91    else
92     SendSaml user (Failed Requester);
93     identityprovider me user authp
94    | Login (loginInfo, challenge) ->
95     if (verifychallenge user challenge) then
96      let req = LoginRequestMessage me authp loginInfo in
97      SendSaml authp req;
98      handleauthresponse me user authp;
99      identityprovider me user authp
100    else
101     SendSaml user (DisplayError 400);
102     identityprovider me user authp
103    | UserAuthResponse(authInfo, challenge, sigAuth) ->
104     let req = NfactAuthRequest me authp authInfo challenge
           sigAuth in
105     SendSaml authp req;
106     handleauthresponse me user authp;
107     identityprovider me user authp
108    | _ -> SendSaml user (DisplayError 400);
109     identityprovider me user authp
110
111  val savejavascript: javascript:string -> unit
112  val savepublickey: owner:prin -> publickey:pubkey owner -> unit
113
114  val connectwithauthp: me:prin -> authp:prin -> unit
115
116  let connectwithauthp me authp =
117   let req = NewSiteRequest me in
118   let _ = SendMessage authp req in
119   let resp = ReceiveMessage authp in
120   match resp with
121   | ChallengeResponse(challenge) ->
122    let _ = SendMessage authp (IdpChalResponse challenge) in
123    let res = ReceiveMessage authp in
124    match res with
125    | AcceptedIdp(idp, idppubkey, authp, authppubkey, signedjs)
          ->
126     (*Established secure connection*)
127     savejavascript signedjs;
128     savepublickey authp authppubkey;
129     savepublickey idp idppubkey
130    | _ -> res; ()
131   | _ -> resp; ()
```

Listing 4.7: Identity Provider module

4.4.8 Specification of the Database Handler

```
1   module Database
2
3   open Crypto
4   open CertStore
5   open TypeFunc
6
7   (*Identity provider functionality*)
8   val whitelist: idp:prin -> unit
9   val blacklist: idp:prin -> unit
10  val addidp: idp:prin -> bool
11  val whitelisted: idp:prin -> bool
12
13  (*User functionality*)
14  val createuser: user:prin -> userid:string -> password:string
        -> bool
15  val usercreation: user:prin -> generatedPassword:string -> bool
16  val changeuserid: user:string -> newuser:string -> password:
        string -> bool
17  val changeuserpassword: user:string -> password:string ->
        newpassword:string -> bool
18
19  val addnfactor: user:string -> password:string -> nfactor:
        Authentication -> bool
20  val removenfactor: user:string -> password:string -> nfactor:
        Authentication -> bool
21
22  val getnfactor: user:string -> Authentication
23  val checknfactor: user:string -> Authentication -> bool
24  val allnfactauthed: user:string -> bool
25  val resetnfact: user:string -> unit
26
27  val checklogin: user:string -> password:string -> bool
28
29  val revokeidp: user:string -> password:string -> idp:string ->
        bool
30
31  val revokedidp: user:string -> idp:prin -> bool
```

Listing 4.8: Database module

4.4.9 Specification of the Authentication Provider

```
1   module Authenticationprovider
2
3   open SamlProtocol
4   open Crypto
5   open Database
6   open TypeFunc
7   open Messaging
8
9   val relatechallenge: user:prin -> challenge:nonce -> unit
10
11  val verifychallenge: user:prin -> challenge:nonce -> bool
```

```
12
13  val nfactauth: me:prin -> idp:prin -> user:prin -> userid:
        string -> unit
14
15  let nfactauth me idp user userid =
16   if (allnfactauthed userid) then
17    resetnfact userid;
18    let status = "OK" in
19    let resp = LoginSuccess status me idp in
20    SendSaml idp resp
21   else
22    let challenge = GenerateNonce me in
23    let authmethod = getnfactor userid in
24    assume(LogAuth user authmethod);
25    let userprivkey = CertStore.GetPrivateKey user in
26    let sigUser = SignAuth user userprivkey authmethod in
27    let resp = LoginResponseMessage me idp authmethod challenge
          sigUser in
28    SendSaml idp resp
29
30  val authenticationprovider: me:prin -> idp:prin -> user:prin ->
        unit
31
32  let rec authenticationprovider me idp user =
33   let req = ReceiveSaml idp in
34   match req with
35   | LoginRequestMessage (issuer, destination, loginInfo) ->
36    if (whitelisted idp) then
37     match loginInfo with
38     | UserLogin(userid, password) ->
39      if not (revokedidp userid idp) && (checklogin userid
            password) then
40       let challenge = GenerateNonce me in
41       let authmethod = getnfactor userid in
42       assume(LogAuth user authmethod);
43       let userprivkey = CertStore.GetPrivateKey user in
44       let sigUser = SignAuth user userprivkey authmethod in
45       relatechallenge user challenge;
46       let resp = LoginResponseMessage me idp authmethod
            challenge sigUser in
47       SendSaml idp resp;
48       authenticationprovider me idp user
49      else
50       SendSaml idp (Failed User);
51       authenticationprovider me idp user
52     | _ -> SendSaml idp (Failed Requester);
53      authenticationprovider me idp user
54    else
55     SendSaml idp (Failed Requester);
56     authenticationprovider me idp user
57   | NfactAuthRequest(issuer, destination, authInfo, challenge,
        sigAuth) ->
58    if (whitelisted idp) then
59     match authInfo with
60     | UserAuth(userid, authmethod, authresponse) ->
61      let userpubkey = CertStore.GetPublicKey user in
```

```
62      if VerifySignatureAuth user userpubkey authmethod sigAuth
           && verifychallenge user challenge then
63       if not (revokedidp userid idp) && (checknfactor userid
            authresponse) then
64        nfactauth me idp user userid;
65        authenticationprovider me idp user
66       else
67        SendSaml idp (Failed User);
68        authenticationprovider me idp user
69      else
70       SendSaml idp (Failed User);
71       authenticationprovider me idp user
72      | _ -> SendSaml idp (Failed Requester);
73       authenticationprovider me idp user
74     else
75      SendSaml idp (Failed Requester);
76      authenticationprovider me idp user
77     | _ -> SendSaml idp (Failed Requester);
78      authenticationprovider me idp user
79
80
81  val usercommunication: me:prin -> user:prin -> unit
82
83  let rec usercommunication me user =
84    let req = ReceiveMessage user in
85    match req with
86    | RequestForLogin(userid, password) ->
87     if createuser user userid password then
88      let challenge = GenerateNonce me in
89      relatechallenge user challenge;
90      SendMessage user (ReqLoginResponse challenge);
91      usercommunication me user
92     else
93      SendMessage user (StatusMessage Unsuccessful);
94      usercommunication me user
95    | CreateLogin(generatedpassword, challenge) ->
96     if (verifychallenge user challenge) && (usercreation user
          generatedpassword) then
97      let challenge = GenerateNonce me in
98      relatechallenge user challenge;
99      SendMessage user (StatusMessage Successful);
100     usercommunication me user
101    else
102     SendMessage user (StatusMessage Unsuccessful);
103     usercommunication me user
104   | ChangePassword(userid, password, newPassword) ->
105    if changeuserpassword userid password newPassword then
106     SendMessage user (StatusMessage Successful);
107     usercommunication me user
108    else
109     SendMessage user (StatusMessage Unsuccessful);
110     usercommunication me user
111   | ChangeUserId(userid, newUserId, password) ->
112    if changeuserid userid newUserId password then
113     SendMessage user (StatusMessage Successful);
114     usercommunication me user
115    else
```

21

```
116      SendMessage user (StatusMessage Unsuccessful);
117      usercommunication me user
118    | UserRevokeIdp(userid, password, idp) ->
119     if revokeidp userid password idp then
120      SendMessage user (StatusMessage Successful);
121      usercommunication me user
122     else
123      SendMessage user (StatusMessage Unsuccessful);
124      usercommunication me user
125    | AddNfactor(userid, password, nfact) ->
126     if addnfactor userid password nfact then
127      SendMessage user (StatusMessage Successful);
128      usercommunication me user
129     else
130      SendMessage user (StatusMessage Unsuccessful);
131      usercommunication me user
132    | RemoveNfactor(userid, password, nfact) ->
133     if removenfactor userid password nfact then
134      SendMessage user (StatusMessage Successful);
135      usercommunication me user
136     else
137      SendMessage user (StatusMessage Unsuccessful);
138      usercommunication me user
139    | _ -> SendMessage user (StatusMessage Unsuccessful);
140     usercommunication me user
141
142  val getsignedjavascript: string
143
144  val establishidp: me:prin -> idp:prin -> unit
145
146  let rec establishidp me idp =
147   let req = ReceiveMessage idp in
148   match req with
149   | NewSiteRequest(idp) ->
150    let challenge = GenerateNonce me in
151    relatechallenge idp challenge;
152    SendMessage idp (ChallengeResponse challenge);
153    establishidp me idp
154   | IdpChalResponse(challenge) ->
155    if (verifychallenge idp challenge) && (addidp idp) then
156     let idppubkey = CertStore.GetPublicKey idp in
157     let mypubk = CertStore.GetPublicKey me in
158     let signedjs = getsignedjavascript in
159     let resp = AcceptedIdp idp idppubkey me mypubk signedjs in
160     SendMessage idp resp;
161     establishidp me idp
162    else
163     SendMessage idp (StatusMessage Unsuccessful);
164     establishidp me idp
165   | _ -> SendMessage idp (StatusMessage Unsuccessful);
166    establishidp me idp
```

Listing 4.9: Authentication Provider module

4.4.10   Specification of the Browser

```
1   module Browser
2
3   open SamlProtocol
4   open Crypto
5   open CertStore
6   open TypeFunc
7   open Messaging
8
9   val loginWithFb: Authentication
10  val loginWithGoogle: Authentication
11  val loginWithSMS: Authentication
12  val loginWithOpenId: Authentication
13  val userid: string
14  val password: string
15  val fakeprint: str:string -> unit
16  val newUserId: string
17  val newPassword: string
18  val idpToRevoke:string
19  val nfactToRemove: Authentication
20  val nfactToAdd: Authentication
21  (*Handle the two-factor authentication*)
22
23  val handleAuthMethod: auth:Authentication -> Authentication
24
25  let handleAuthMethod auth =
26   match auth with
27   | Facebook(fbid) -> loginWithFb
28   | Google(gid) -> loginWithGoogle
29   | SMS(gen) -> loginWithSMS
30   | OpenId(oid) -> loginWithOpenId
31
32  val loop: user:string -> idp:prin -> sp:prin -> unit
33
34  let rec loop userid idp sp =
35   let loginresp = ReceiveSaml idp in
36    match loginresp with
37    | UserAuthRequest(authmethod, challenge, sigAuth) ->
38     let authresponse = handleAuthMethod authmethod in
39     let authInfo = UserAuth userid authmethod authresponse in
40     let authresp = UserAuthResponse authInfo challenge sigAuth
          in
41     SendSaml idp authresp;
42     loop userid idp sp
43     | AuthResponseMessage(idenp, dest, assertion) ->
44     SendSaml sp loginresp
45     | _ -> loginresp; ()
46
47  val browser: sp:prin -> res:uri -> unit
48
49  let browser sp resource =
50   let req = SPLogin resource in
51   let _ = SendSaml sp req in
52    let res = ReceiveSaml sp in
53    match res with
54    | AuthnRequestMessage(sp, idp, message, sigSP) ->
55     let _ = SendSaml idp res in
56     let idpResp = ReceiveSaml idp in
```

```
57      match idpResp with
58      | UserCredRequest(javascript, challenge, sigIdP) ->
59       let pubkissuer = CertStore.GetJSPublicKey idp in
60       if VerifySignature idp pubkissuer javascript sigIdP then
61        (assert (Log idp javascript);
62        let loginInfo = UserLogin userid password in
63        let loginreq = Login loginInfo challenge in
64        SendSaml idp loginreq;
65        loop userid idp sp;
66        let spResp = ReceiveSaml sp in
67        match spResp with
68        | LoginResponse(str) ->
69          fakeprint str
70        | _ -> spResp; ())
71       else
72        fakeprint "Validation Error"
73      | _ -> idpResp; ()
74     | _ -> res; ()
75
76   val retrieveGeneratedPassword: string
77
78   val createUser: authp:prin -> unit
79
80   let createUser authp =
81    let name = userid in
82    let pw = password in
83    let req = RequestForLogin name pw in
84    let _ = SendMessage authp req in
85     let resp = ReceiveMessage authp in
86     match resp with
87     | ReqLoginResponse(challenge) ->
88      let reqlresp = CreateLogin retrieveGeneratedPassword
          challenge in
89      let _ = SendMessage authp reqlresp in
90      let createloginresp = ReceiveMessage authp in
91      match createloginresp with
92      | StatusMessage(status) ->
93       match status with
94       | Successful -> fakeprint "You have created an account"
95       | Unsuccessful -> fakeprint "Something went wrong. No
           account has been created"
96      | _ -> createloginresp; ()
97     | _ -> resp; ()
98
99   val changeUserPassword: authp:prin -> unit
100
101  let changeUserPassword authp =
102   let name = userid in
103   let pw = password in
104   let newpw = newPassword in
105   let req = ChangePassword name pw newpw in
106   let _ = SendMessage authp req in
107    let resp = ReceiveMessage authp in
108    match resp with
109    | StatusMessage(status) ->
110     match status with
111     | Successful -> fakeprint "You have change your password"
```

```
112        | Unsuccessful -> fakeprint "Something went wrong. You have
                  not changed your password"
113      | _ -> resp; ()
114
115  val changeUserUserId: authp:prin -> unit
116
117  let changeUserUserId authp =
118   let name = userid in
119   let pw = password in
120   let newid = newUserId in
121   let req = ChangeUserId name newid pw in
122   let _ = SendMessage authp req in
123   let resp = ReceiveMessage authp in
124    match resp with
125    | StatusMessage(status) ->
126      match status with
127      | Successful -> fakeprint "You have change your userid"
128      | Unsuccessful -> fakeprint "Something went wrong. You have
                  not changed your userid"
129      | _ -> resp; ()
130
131  val identityrevoke: authp:prin -> unit
132
133  let identityrevoke authp =
134   let name = userid in
135   let pw = password in
136   let idp = idpToRevoke in
137   let req = UserRevokeIdp name pw idp in
138   let _ = SendMessage authp req in
139   let resp = ReceiveMessage authp in
140    match resp with
141    | StatusMessage(status) ->
142      match status with
143      | Successful -> fakeprint "You have revoked the
                  identityprovider"
144      | Unsuccessful -> fakeprint "Something went wrong. You have
                  not revoked the identityprovider"
145      | _ -> resp; ()
146
147  val addNfact: authp:prin -> unit
148
149  let addNfact authp =
150   let name = userid in
151   let pw = password in
152   let nfact = nfactToAdd in
153   let req = AddNfactor name pw nfact in
154   let _ = SendMessage authp req in
155   let resp = ReceiveMessage authp in
156    match resp with
157    | StatusMessage(status) ->
158      match status with
159      | Successful -> fakeprint "You have added this
                  authentication method"
160      | Unsuccessful -> fakeprint "Something went wrong. You have
                  not added this authentication method"
161      | _ -> resp; ()
162
```

```
163  val removeNfact: authp:prin -> unit
164
165  let removeNfact authp =
166   let name = userid in
167   let pw = password in
168   let nfact = nfactToRemove in
169   let req = RemoveNfactor name pw nfact in
170   let _ = SendMessage authp req in
171   let resp = ReceiveMessage authp in
172    match resp with
173    | StatusMessage(status) ->
174      match status with
175      | Successful -> fakeprint "You have removed this
             authentication method"
176      | Unsuccessful -> fakeprint "Something went wrong. You have
             not removed this authentication method"
177    | _ -> resp; ()
```

Listing 4.10: Browser module

4.5   Introducing adversaries

```
1   module Main
2
3   open SamlProtocol
4   open Crypto
5   open Serviceprovider
6   open Identityprovider
7   open Authenticationprovider
8
9   val Fork: list (unit -> unit) -> unit
10
11  let main attacker =
12   Fork [ attacker;
13    (fun () -> serviceprovider "serviceprovider.org" "browser" "
         identityprovider.org");
14    (fun () -> identityprovider "identityprovider.org" "browser"
         "authenticationprovider.org");
15    (fun () -> authenticationprovider "authenticationprovider.org
         " "identityprovider.org" "browser")]
```

Listing 4.11: Main module for introducing adversaries

Chapter 5

Evaluation