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

List of Tables

List of Figures

Chapter 1

Introduction

...

We're extending the work done by Jacob Højgaard 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.

Jacobs report sums up some of the problems with the current implementation of NemID, these problems include but are not limited to the system being very opaque. We're in the age of information, we want to be able to get information about everything regardless if it's in our best interest or not. NemID is not sharing any information about the internals of the system. If a person wanted to test some part of the NemID system, they would first have to analyze the public parts of the system to figure out how to communicate with the NemID system, afterwards all they would be able to do, would be black box testing. We need a system that is transparent, testable by everyone and doesn't cost a whole lot. For the sake of being able to reference this system through out the report, we will refer to it as *OpenNemID*.

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

2.1.1 OIOSAML

2.2 Static Analysis

2.3 Selection of verification tool

F* - formal specification language that is also executable

2.4 N Factor Authentication

Viderudvikling af two factor authentication

Chapter 3

Modelling the protocol

Before formalizing the protocol, it's required to specify and explain some of the words, concepts and meanings used within the protocol. This will be done by using graphical representation of the message flow.

3.1 Protocol Prerequisites

It's important to have some requirements as to how the systems should function. The requirements helps define certain properties the involved participants must have or obey to.

3.1.1 Shared

The NemLog-in specification mandates the use of OIOSAML, this will most likely not be excluded, therefore we assume that OpenNemID also has to use it. Further OIOSAML mandates the use of one-way SSL/TLS for all bindings, the mandate does not specify a specific version, though it can be assumed that a minimum version of SSL 3.0 due to the fact that SSL 2.0 is in general considered deprecated. We assume the use of SSL 3.0 or TLS 1.0 in this report.

As mentioned before, SAML uses the browser to transfer messages from one principal to the other. The way to do this is through HTTP REDIRECTs, which could be either a HTTP-GET REDIRECT or a HTTP-POST REDIRECT. The HTTP protocol accepts a Location head in the HTTP RESPONSE which indicates where the browser should redirect to. The location header redirect will act as a HTTP-GET REQUEST which excludes the usage of POST data, thereby limiting the amount of data that can be transferred. To overcome this problem HTTP-POST REDIRECTs are used, these are not a part of the HTTP protocol, but is synthesized by using JavaScript to emulate a regular HTTP-POST REQUEST. Therefore it is required for the users browser to follow redirects and to have JavaScript enabled.

For there to be any actual messages to flow between the service provider

and identity provider, it's assumed that they reside in different domains and are different entities.

The identity provider is only to issue assertions to known service providers, this requires that SAML metadata has been changed beforehand. The certificates used for signing and encrypting has to be checked for revocation and validity whenever used.

To summarize:

1. OIOSAML mandates the use of SSL(3.0)/TLS(1.0).
2. The browser must follow redirects.
3. The browser must have JavaScript enabled.
4. Service provider and identity provider are different entities residing in different domains.
5. SAML metadata must have been exchanged between the service provider and identity provider.
6. Signature check and encryption requires validity/revocation check of the certificate.

3.1.2 NemID Specifics

It's required for the OCES certificates used for signing and encrypting to have been issued by DanID.

3.1.3 OpenNemID Specifics

For the communication between the authentication provider and the identity provider, a secure tunnel must have been set up. Further the user must have registered at the authentication provider.

3.2 Formalizing Protocol Messages

The UML communication diagrams depicting the protocols are made up of two or more participants, henceforth principals, and the messages flowing through the system. The line between two principals indicates a channel where communication can flow, this channel is assumed to be a secure channel, meaning for HTTP messages, the HTTPS protocol would be used. An arrow indicates the direction of the message and the text on top of the arrow is the message being sent. The messages does not conform to any specific formalism, but follows a simple syntax. Messages are, very applicable, named in accordance with their HTTP protocol verb. The messages are to be interpreted the following way:

> **GET** means a HTTP-GET request, the parameter is the resource being requested.

POST means a HTTP-POST request, the parameters are the destination for the request followed by the data being submitted.

REDIRECT means either a HTTP-REDIRECT or a JavaScript redirect, whichever is used is not important for the purpose of the description. The parameters are the destination for the redirect followed by the parameters to include in the redirect.

RESPONSE means a HTTP-RESPONSE messages. The parameters are either the data included in the response or a HTTP status code indicating the type of the response along with a message, this is used for indicating when error happen.

DELEGATE means forwarding the data from the previous request, the parameters are the parameters from the previous request that were to be delegated.

AUTHENTICATE is to be interpreted as the sequence of actions required to be authenticated at the specified NFactorChallenge. The AUTHENTICATE message is defined this generically on purpose, as the way a user would authenticate for a NFactorChallenge can vary a lot depending on which technology is used (SMS, Facebook, phone call etc.).

3.3 Communication Model

To make the changes from NemID to OpenNemID clear, we will first show the communication diagram for NemID, afterwards we will show the communication diagram for OpenNemID. Both diagrams make use of abbreviations, these abbreviations are listed at the top of the diagram along the word or phrase they abbreviate.

3.3.1 Message Processing

Information regarding the processing of messages have not been included in the diagrams, this is to prevent cluttering of the diagrams. To circumvent this, the processing rules will be described afterwards. The descriptions will be linked to a specific process number, meaning Process 3 would be the handling and response of message 3 in the diagram. Messages that are self-explanatory will not be further described.

3.3.2 Communication Diagram For NemID

Description of the message processing for this diagram has been left out of our report, they can however be found in Jacob Højgaards report [1].

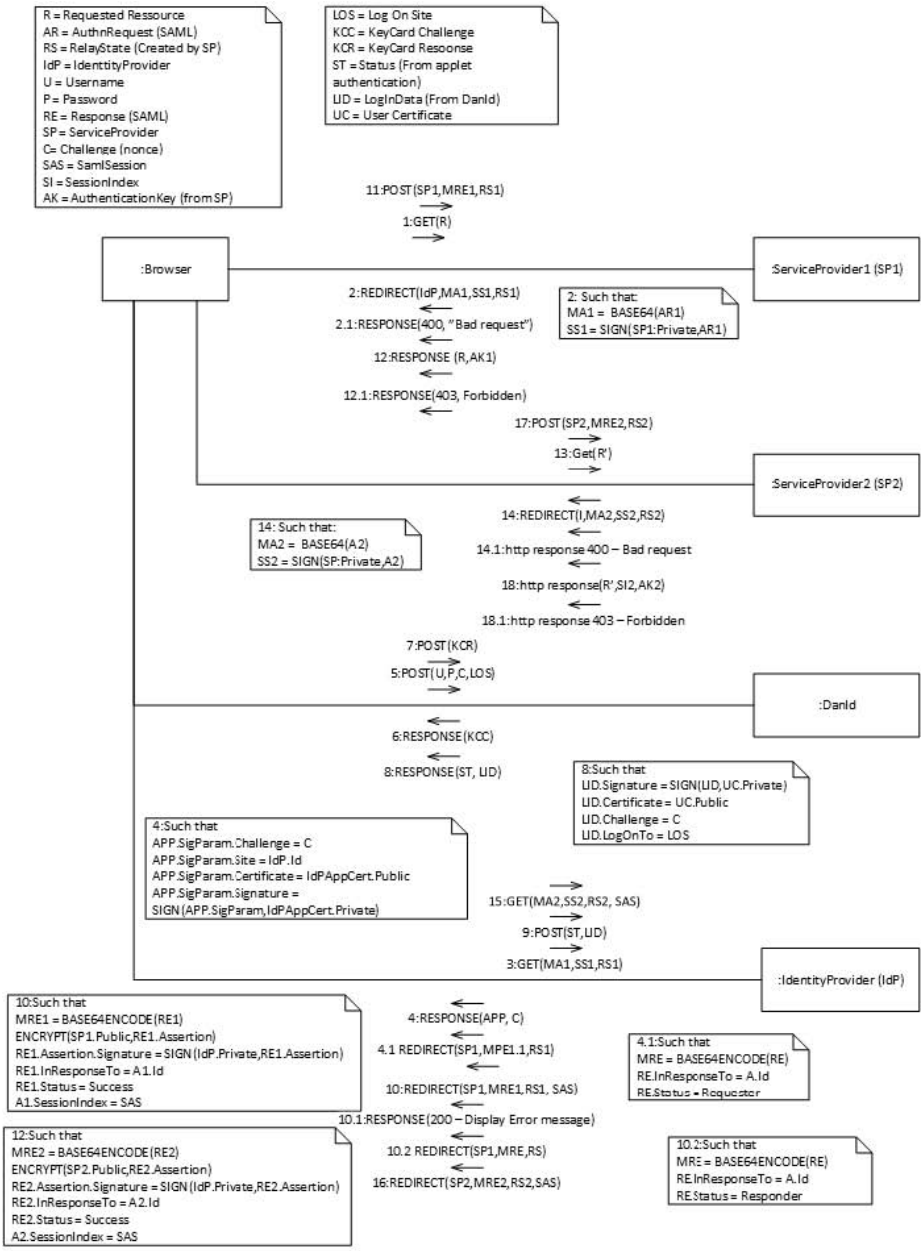

Figure 3.1: Communication diagram for the complete NemID protocol [1]

3.3.3 Communication Diagram For OpenNemID

In this diagram, we have chosen to leave out the additional request to another service provider than the one initially used, this is due to the communication flow being exactly the same as in Jacobs diagram, see Figure 3.1, message 13 to 18.

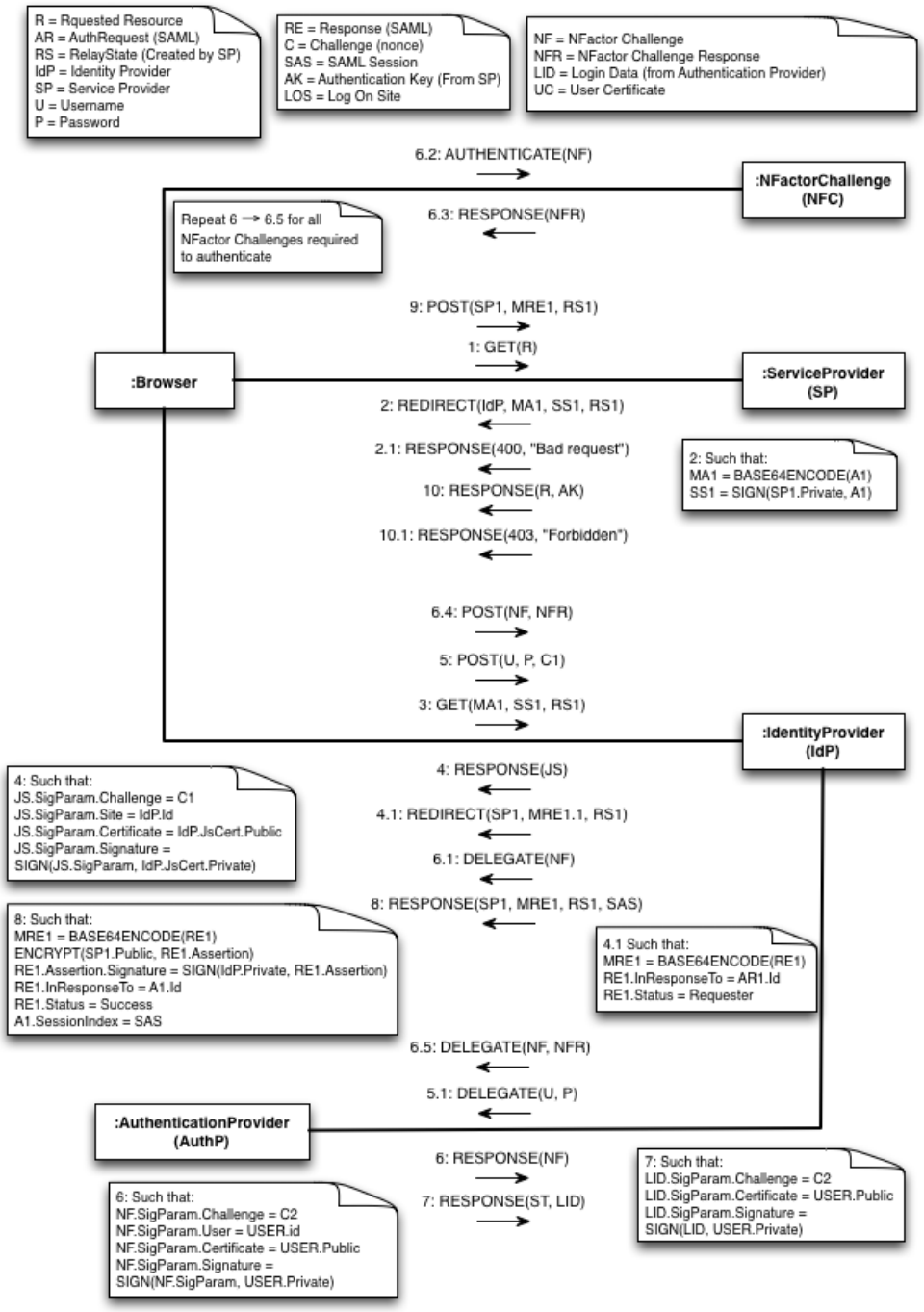

Figure 3.2: Communication diagram for the OpenNemID protocol

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

TEXT DESCRIBING ALGORITHM 6

Algorithm 6 Process 6

```
SigParams ← 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*

This chapter will introduce the language F* that can be used to model a security protocol. Despite being a formal specification language F* is also executable. F* is described as a *A Verifying Compiler for Distributed Programming*. This chapter will describe how we have used F* to build a formal specification of our protocol shown in the previous chapter.

4.1 Introducing F*

F* is a research language from Microsoft Research. F* primarily subsumes two research languages from Microsoft Research, F7[1] and Fine[2]. F* is at this time considered to be an α-release.The purpose of designing F* is to enable the construction and communication of proofs of program properties and of properties of a program's environment in a verifiable secure way. F* is a dialect of ML and compiles to .NET bytecode in type-preserving style. This means that it can interop with other .NET languages and the types defined in F* can be used by other .NET lanuages without loosing type information. Furthermore there also exists a fully abstract compiler from F* to JavaScript. This makes it possible to deploy F* programs on web pages as JavaScript meanwhile there is a formal guarantee that the program still behaves just as they would according to F* semantics. The compiling and type-checking of F* code utilizes the Z3[3] SMT solver for proving assumptions made with refinement types. F* has been formalized and verified using Coq[4].

[1] http://research.microsoft.com/en-us/projects/f7/
[2] http://research.microsoft.com/en-us/projects/fine/
[3] http://z3.codeplex.com/
[4] Coq is an interactive theorem prover written in OCaml

4.2 Syntax and semantics

F* inherits syntax and semantics from ML. F* is a functional language which means that it has features like immutability by default, polymorphic types and type inference. In Listing 4.1 we have shown the classic Hello World example in F*. This is the simplest way this example could have been written. This example shows how to specify a main method

```
module HelloWorld

let _ = print "Hello world!"
```

Listing 4.1: Hello World example in F*

4.3 Refinement types

4.4 Protocol specification in F*

The code in this section represents the state of the project now. This is in no way a complete implementation of the protocol. Implementation was carried out in an incremental manner. First the focus was on understanding Jacob's work and expanding that with the authentication part (Authentication Provider) of the protocol, which before was done by NemID, and then adding the functionality of creating login, establishing connection between Identity Provider and the Authentication Provider and so on. All of the code

4.4.1 Specification of the type functionality module

```
module TypeFunc

type Authentication =
  | Facebook: id:int -> Authentication
  | SMS: generated:int -> Authentication
 | Google: id:int -> Authentication
 | OpenId: id:int -> Authentication
```

Listing 4.2: TypeFunc module

4.4.2 Specification of the SAML Protocol

```
module SamlProtocol

open Crypto
open TypeFunc

type assertiontoken = string (*Add refinements*)
type signedtoken = string (*Add refinements*)
type id = string
```

```
type endpoint = string
type uri = string


type AuthnRequest =
  | MkAuthnRequest: IssueInstant:string ->
      Destination:endpoint -> Issuer:prin ->
      message:string -> sig:dsig ->
      AuthnRequest

type LoginData =
  | MkLoginData:  user:prin -> signature:dsig ->
    cert:pubkey user -> challenge:nonce ->
    site:string -> data:string ->
    LoginData

type LoginInfo =
  | UserLogin:  userid:string -> password:string ->
  LoginInfo

type AuthInfo =
  | UserAuth:   userid:string -> authmethod:Authentication ->
  authresponse:Authentication -> AuthInfo

type Assertion =
  | SignedAssertion: assertiontoken -> dsig -> Assertion
  | EncryptedAssertion: cypher -> Assertion

type SamlStatus =
  | Success: SamlStatus
  | Requester: SamlStatus
  | Responder: SamlStatus
  | User: SamlStatus

type SamlMessage =
  | SPLogin: uri -> SamlMessage
  | Login: loginInfo:LoginInfo -> challenge:nonce ->
      SamlMessage
  | LoginResponse: string -> SamlMessage
  | AuthnRequestMessage: issuer:prin ->  destination:endpoint
      -> message:string -> dsig -> SamlMessage
  | LoginRequestMessage: issuer:prin ->  destination:endpoint
      -> loginInfo:LoginInfo -> SamlMessage
  | NfactAuthRequest: issuer:prin -> destination:endpoint ->
      authInfo:AuthInfo -> challenge:nonce -> dsig ->
      SamlMessage
  | AuthResponseMessage: issuer:prin -> destination:endpoint ->
       Assertion -> SamlMessage
  | LoginResponseMessage: issuer:prin -> destination:endpoint
      -> auth:Authentication -> challenge:nonce -> dsig ->
      SamlMessage
  | UserAuthenticated: status:string -> logindata:LoginData ->
      authnReq:AuthnRequest -> SamlMessage
  | UserCredRequest: javascript:string -> challenge:nonce ->
      dsig -> SamlMessage
  | UserAuthRequest: authmethod:Authentication -> challenge:
      nonce -> dsig -> SamlMessage
```

```
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 4.3: 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}
```

```
  -> dsig

val SignAuth: p:prin
 -> privkey p
 -> msg:Authentication{LogAuth p msg}
 -> dsig

val VerifySignature: p:prin
 -> pubkey p
 -> msg:string
 -> dsig
 -> b:bool{b=true ==> Log p msg}

val VerifySignatureAuth: p:prin
 -> pubkey p
 -> msg:Authentication
 -> dsig
 -> b:bool{b=true ==> LogAuth p msg}

val Encrypt: p:prin
 -> pubkey p
 -> string
 -> cypher

val Decrypt: p:prin
 -> privkey p
 -> cypher
 -> string

val GenerateNonce: prin -> nonce (*Add refinement to ensure
    unqueness*)
```

Listing 4.4: Crypto module

4.4.4 Specification of certificate store module

```
module CertStore

open Crypto

val GetPublicKey: p:prin -> pubkey p
val GetJSPublicKey: p:prin -> pubkey p
(*Prin needs to be updated to include credentials*)
val GetPrivateKey: p:prin -> privkey p
val GetJSPrivateKey: p:prin -> privkey p
```

Listing 4.5: CertStore module

4.4.5 Specification of the messaging protocol

```
module Messaging

open Crypto
```

```
open TypeFunc

type Status =
  | Successful: Status
  | Unsuccessful: Status

type Message =
  | NewSiteRequest: idp:prin -> Message
  | ChallengeResponse: challenge:nonce -> Message
  | IdpChalResponse: challenge:nonce -> Message
  | AcceptedIdp: idp:prin -> pubkey:pubkey idp -> authp:prin ->
      authpubkey:pubkey authp -> signedjavascript:string ->
      Message
  | RequestForLogin: userid:string -> password:string -> Message
  | ReqLoginResponse: challenge:nonce -> Message
  | CreateLogin: generatedpassword:string -> challenge:nonce ->
      Message
  | ChangeUserId: userid:string -> newUserId:string -> password:
      string -> Message
  | ChangePassword: userid:string -> password:string ->
      newPassword:string -> Message
  | UserRevokeIdp: userid:string -> password:string -> idp:
      string -> Message
  | AddNfactor: userid:string -> password:string -> nfact:
      Authentication -> Message
  | RemoveNfactor: userid:string -> password:string -> nfact:
      Authentication -> Message
  | StatusMessage: Status -> Message


val SendMessage: prin -> Message -> unit
val ReceiveMessage: prin -> Message
```

Listing 4.6: Messaging module

4.4.6 Specification of the Service Provider

```
module Serviceprovider

open SamlProtocol
open Crypto

val serviceprovider:  me:prin -> client:prin -> idp:prin ->
    unit

let rec serviceprovider me client idp =
 let req = ReceiveSaml client in
 match req with
  | SPLogin (url) ->
    let authnReq = CreateAuthnRequestMessage me idp in
    assume(Log me authnReq);
    let myprivk = CertStore.GetPrivateKey me in
    let sigSP = Sign me myprivk authnReq in
    let resp = AuthnRequestMessage me idp authnReq sigSP in
    SendSaml client resp;
```

```
      serviceprovider me client idp
  | AuthResponseMessage (issuer, destination, encassertion) ->
    let myprivk = CertStore.GetPrivateKey me in
    let assertion = DecryptAssertion me myprivk encassertion in
    match assertion with
    | SignedAssertion (token,sigIDP) ->
      let pubkissuer = CertStore.GetPublicKey idp in
      if VerifySignature idp pubkissuer token sigIDP
      then
        (assert(Log idp token);
        let resp = LoginResponse "You are now logged in" in
        SendSaml client resp)
      else SendSaml client (DisplayError 403);
      serviceprovider me client idp

  | _ -> SendSaml client (DisplayError 400);
        serviceprovider me client idp
```

Listing 4.7: ServiceProvider Module

4.4.7 Specification of the Identity Provider

```
module Identityprovider

open SamlProtocol
open Crypto
open TypeFunc
open Messaging

val userloggedin: user:prin -> bool
val getjavascript: string
val decodeMessage: message:string -> AuthnRequest
val getauthnrequest: user:prin -> challenge:nonce ->
    AuthnRequest
val getuserchallenge: user:prin -> nonce
val relatechallenge: user:prin -> challenge:nonce -> unit
val verifychallenge: user:prin -> challenge:nonce -> bool
val relate: user:prin -> challenge:nonce -> authnReq:
    AuthnRequest -> unit

val handleUserAuthenticated: me:prin -> user:prin -> authnReq:
    AuthnRequest -> unit

let handleUserAuthenticated me user authnReq =
 match authnReq with
 | MkAuthnRequest(issueinst,dest,sp,msg,sigSP) ->
  let pubksp = CertStore.GetPublicKey sp in
   if (VerifySignature sp pubksp msg sigSP) then
  (assert (Log sp msg);
  let assertion = IssueAssertion me user sp authnReq in
  let myprivk = CertStore.GetPrivateKey me in
  assume(Log me assertion);
  let sigAs = Sign me myprivk assertion in
  let signAssertion = AddSignatureToAssertion assertion sigAs
      in
```

```
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
```

```
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 4.8: Identity Provider module

4.4.8 Specification of the Database Handler

```
module Database

open Crypto
open CertStore
open TypeFunc

(*Identity provider functionality*)
val whitelist: idp:prin -> unit
val blacklist: idp:prin -> unit
val addidp: idp:prin -> bool
val whitelisted: idp:prin -> bool

(*User functionality*)
val createuser: user:prin -> userid:string -> password:string
    -> bool
val usercreation: user:prin -> generatedPassword:string -> bool
val changeuserid: user:string -> newuser:string -> password:
    string -> bool
val changeuserpassword: user:string -> password:string ->
    newpassword:string -> bool

val addnfactor: user:string -> password:string -> nfactor:
    Authentication -> bool
val removenfactor: user:string -> password:string -> nfactor:
    Authentication -> bool

val getnfactor: user:string -> Authentication
val checknfactor: user:string -> Authentication -> bool
val allnfactauthed: user:string -> bool
val resetnfact: user:string -> unit

val checklogin: user:string -> password:string -> bool

val revokeidp: user:string -> password:string -> idp:string ->
    bool

val revokedidp: user:string -> idp:prin -> bool
```

Listing 4.9: Database module

4.4.9 Specification of the Authentication Provider

```
module Authenticationprovider

open SamlProtocol
open Crypto
open Database
open TypeFunc
open Messaging

val relatechallenge: user:prin -> challenge:nonce -> unit

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
```

```
    SendMessage user (StatusMessage Unsuccessful);
    usercommunication me user
  | UserRevokeIdp(userid, password, idp) ->
   if revokeidp userid password idp then
    SendMessage user (StatusMessage Successful);
    usercommunication me user
   else
    SendMessage user (StatusMessage Unsuccessful);
    usercommunication me user
  | AddNfactor(userid, password, nfact) ->
   if addnfactor userid password nfact then
    SendMessage user (StatusMessage Successful);
    usercommunication me user
   else
    SendMessage user (StatusMessage Unsuccessful);
    usercommunication me user
  | RemoveNfactor(userid, password, nfact) ->
   if removenfactor userid password nfact then
    SendMessage user (StatusMessage Successful);
    usercommunication me user
   else
    SendMessage user (StatusMessage Unsuccessful);
    usercommunication me user
  | _ -> SendMessage user (StatusMessage Unsuccessful);
   usercommunication me user

val getsignedjavascript: string

val establishidp: me:prin -> idp:prin -> unit

let rec establishidp me idp =
 let req = ReceiveMessage idp in
 match req with
 | NewSiteRequest(idp) ->
  let challenge = GenerateNonce me in
  relatechallenge idp challenge;
  SendMessage idp (ChallengeResponse challenge);
  establishidp me idp
 | IdpChalResponse(challenge) ->
  if (verifychallenge idp challenge) && (addidp idp) then
   let idppubkey = CertStore.GetPublicKey idp in
   let mypubk = CertStore.GetPublicKey me in
   let signedjs = getsignedjavascript in
   let resp = AcceptedIdp idp idppubkey me mypubk signedjs in
   SendMessage idp resp;
   establishidp me idp
  else
   SendMessage idp (StatusMessage Unsuccessful);
   establishidp me idp
 | _ -> SendMessage idp (StatusMessage Unsuccessful);
  establishidp me idp
```

Listing 4.10: Authentication Provider module

4.4.10 Specification of the Browser

```
module Browser

open SamlProtocol
open Crypto
open CertStore
open TypeFunc
open Messaging

val loginWithFb: Authentication
val loginWithGoogle: Authentication
val loginWithSMS: Authentication
val loginWithOpenId: Authentication
val userid: string
val password: string
val fakeprint: str:string -> unit
val newUserId: string
val newPassword: string
val idpToRevoke:string
val nfactToRemove: Authentication
val nfactToAdd: Authentication
(*Handle the two-factor authentication*)

val handleAuthMethod: auth:Authentication -> Authentication

let handleAuthMethod auth =
 match auth with
  | Facebook(fbid) -> loginWithFb
  | Google(gid) -> loginWithGoogle
  | SMS(gen) -> loginWithSMS
  | OpenId(oid) -> loginWithOpenId

val loop: user:string -> idp:prin -> sp:prin -> unit

let rec loop userid idp sp =
 let loginresp = ReceiveSaml idp in
  match loginresp with
   | UserAuthRequest(authmethod, challenge, sigAuth) ->
    let authresponse = handleAuthMethod authmethod in
    let authInfo = UserAuth userid authmethod authresponse in
    let authresp = UserAuthResponse authInfo challenge sigAuth
        in
    SendSaml idp authresp;
    loop userid idp sp
   | AuthResponseMessage(idenp, dest, assertion) ->
    SendSaml sp loginresp
   | _ -> loginresp; ()

val browser: sp:prin -> res:uri -> unit

let browser sp resource =
 let req = SPLogin resource in
 let _ = SendSaml sp req in
  let res = ReceiveSaml sp in
  match res with
   | AuthnRequestMessage(sp, idp, message, sigSP) ->
    let _ = SendSaml idp res in
    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 4.11: 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 4.12: Main module for introducing adversaries

Chapter 5

Evaluation

Bibliography

[1] Jakob Højgaard: *Securing Single Sign-On System With Executable Models.* Master Project, IT University of Copenhagen, 2013.

[2] David Basin, Patrick Schaller, Michael Schlpfer: *Applied Information Security - A Hands-on Approach.* Springer, Berlin Heidelberg, 2011.