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

Static analysis

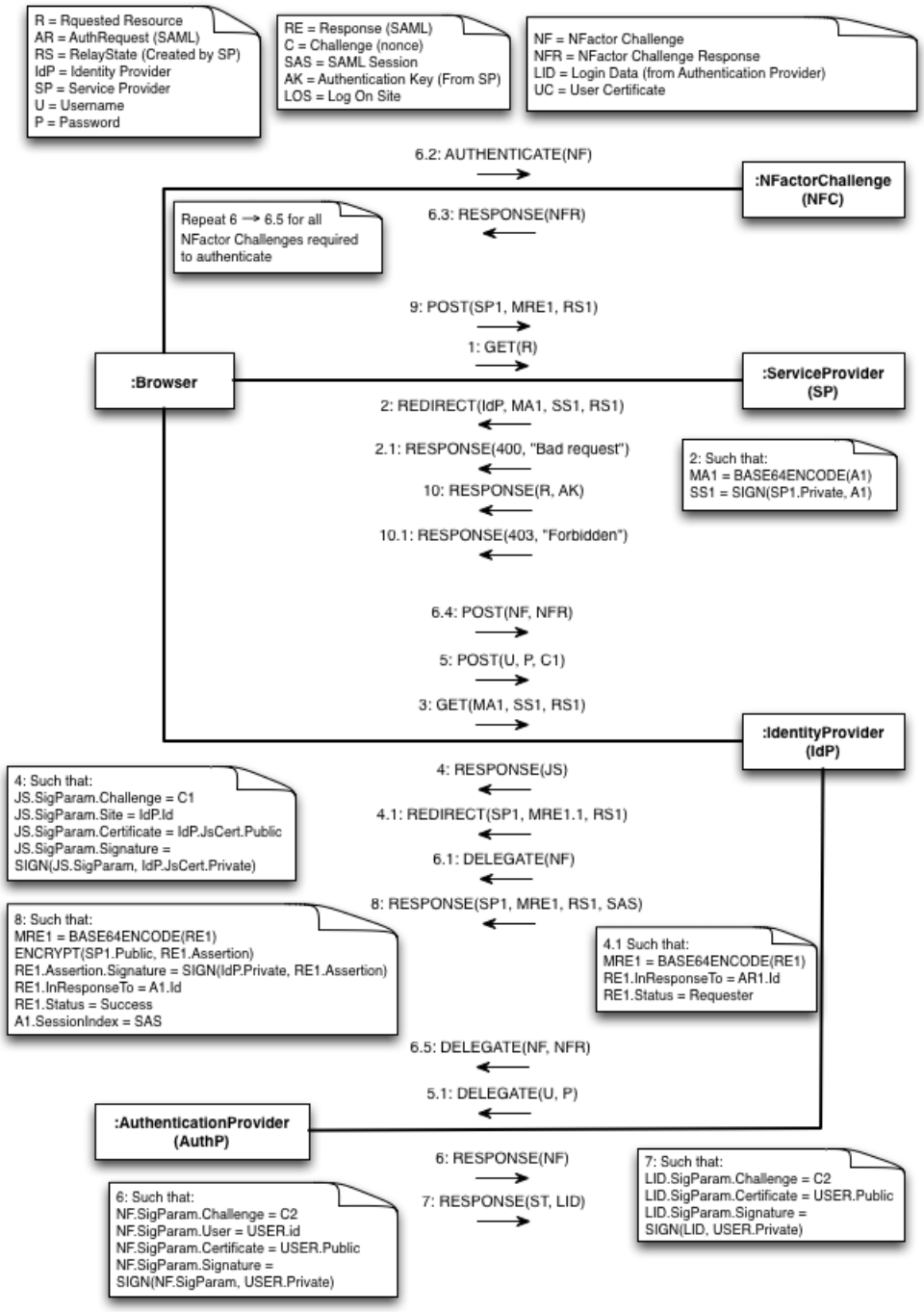
Smart stuff here

Chapter 3

Remodelling the protocol

3.1 Communication Model

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



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

F*

This is F* code, doesn't it look pretty?

```
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
 (* This is awesome comment *)
  | 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 4.1: ServiceProvider Module