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

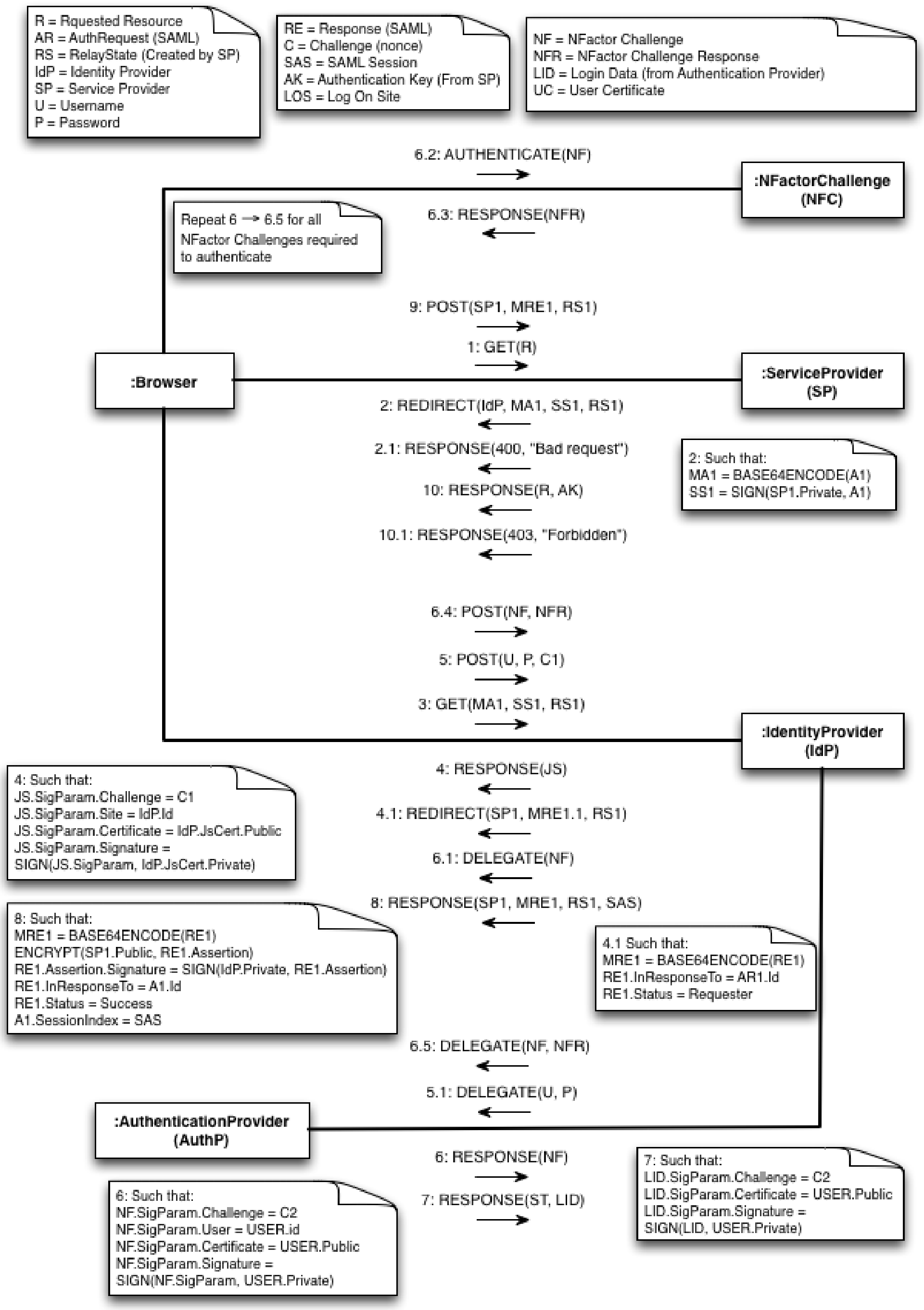
Smart stuff here

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()
    SS ← SIGN(M, SP.Private)
    MA ← UrlEnc(Base64Enc(DeflateCompress(AR)))
    RS ← UrlEnc(Base64Enc(R))
    return REDIRECT(IdP, MA, SS, 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 Id-
  PJsCert.Public and IdP has JavaScript from AuthP
  AR ← DeflateDecompress(Base64Dec(UrlDec(MA)))
  if VERIFY(AR, SS, 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.Id
    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.id
    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

```
SigParams ← NF.SigParams
if VERIFY(SigParams, SigParams.Signature, SigParams.Certificate) then
  if NFR is acceptable result of NF then
    USER ← GetUser(SigParam.USER, SigParam.Certificate)
    C2 ← GenChallenge()
    if USER.HasNextChallenge then
      NF ← GetNextNFactorChallenge(USER)
      NF.SigParam.User ← USER.id
      NF.SigParam.Challenge ← C2
      NF.SigParam.Certificate ← USER.Public
      NF.SigParam.Signature ← SIGN(NF.SigParam, USER.Private)
      return RESPONSE(NF)
    else
      LID ← CreateLogInData()
      LID.SigParam.Challenge ← C2
      LID.Certificate ← USER.Public
      LID.Signature ← SIGN(LID, USER.Private)
      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

```
if VERIFY(LID, LID.Signature, LID.Certificate) then
  if ST = ”OK” then
    MA $\leftarrow$ ARC.AR
    SS $\leftarrow$ ARC.SS
    RS $\leftarrow$ ARC.RS
    AR $\leftarrow$ DeflateDecompress(Base64Dec(UrlDec(MA)))
    if VERIFY(AR, SS, SP.Public) then
      A $\leftarrow$ BuildAssertion(LID.Certificate)
      SI $\leftarrow$ GenerateSessionIndex()
      A.InResponseTo $\leftarrow$ AR.Id
      A.Issuer $\leftarrow$ IdpP.Id
      A.Audience $\leftarrow$ SP.Id
      A.SessionIndex $\leftarrow$ SI
      A.Signature $\leftarrow$ SIGN(A, IdP.Private)
      EA $\leftarrow$ ENCRYPT(A, SP.Public)
      RE $\leftarrow$ CreateResponse()
      RE.Assertion $\leftarrow$ EA
      RE.InResponseTo $\leftarrow$ AR.Id
      Re.Status $\leftarrow$ ”Success”
      MRE $\leftarrow$ Base64Enc(RE)
      SAS $\leftarrow$ CreateSAMLSession(SI, SP.Id, LID.CertificateSubject)
      return REDIRECT(SP, MRE, RS, SAS)
    else
      RE $\leftarrow$ CreateResponse()
      RE.InResponseTo $\leftarrow$ AR.Id
      RE.Status $\leftarrow$ ”Requester”
      MRE $\leftarrow$ Base64Enc(RE)
      return REDIRECT(SP, MRE, RS)
    end if
  else
    return RESPONSE(ST)
  end if
else
  return RESPONSE(ERROR)
end if
```

TEXT DESCRIBING ALGORITHM 11

Algorithm 11 Process 9

```
Require: POST is well.formed and SP.Private and IdP.Public
  RE ← Base64Dec(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
```