Ísland.is

Identification and Authentication Services

Installation Instructions

SAML 2.0

These instructions are intended for technicians and service providers who wish to implement the *Ísland.is* identification and authentication services.

Table of Contents

1 Introduction

The *Ísland.is* identification and authentication services (IAS) provide an authentication system operated by Registers Iceland. The purpose of these services is to offer a choice of identification and authentication options suited to the differing needs of individual service providers and customers. The three options offered are digital certificates, on Smartcards, debetcards or SIM cards in mobile phones, IceKeys and multi-factor IceKeys, with both types of IceKey being issued by Registers Iceland.

Whereas IAS at *Ísland.is* was previously based on SAML 1.2, the system described here is based on SAML 2.0.

In most situations, the implementation of SAML 2.0 should not take very long, as it is generally quite similar to SAML 1.2. Moreover, SAML 2.0 is convenient in that there is no need to contact web services, so that responses can be processed immediately.

The examples given below are intended to be of help, and you are encouraged to use the codes presented whenever convenient.

1.1 IceKey

IceKeys are issued by Registers Iceland, which also issues paper and plastic IDs, including passports. An IceKey is composed of an Icelandic identification number (Social Security Number (SSN)) and a password with a minimum of 10 characters that includes letters, numbers and symbols. The multi-factor IceKey consists of both an IceKey and a six-digit verification code that is sent to the user's mobile telephone.

1.2 SAML 2.0

Security Assertion Markup Language 2.0 (SAML 2.0) is a standard that allows different parties to exchange authentication and documentation information. Based on XML, SAML 2.0 uses security tokens containing assertions in order to transmit information on the person to be authenticated from an authentication service to an Internet service provider.

1.3 Previous Ísland.is identification and authentication services

To utilise the older form of IAS at *Ísland.is*, which was based on SAML 1.2, the service provider had to make a web service call to the *Ísland.is* e-services layer to be able to retrieve a SAML message generated by the identification and authentication service. Token IDs were used for retrieving this message.

2 Installing and connecting - an overview

Inquiries and applications for connections may be sent to island@island.is. You may also contact Registers Iceland at Borgartún 21, Reykjavík, or telephone +354 515 5300. Registers Iceland is the party operating *Ísland.is*.

2.1 Service provider agreements

Registers Iceland makes written contracts with Internet service providers, and to do so requires the following information:

a. Name, address and Icelandic identification number of the service provider

b. Name and e-mail address of an administrative contact person

c. Name and e-mail address of a technical contact person

2.2 Technical matters

Each application must be accompanied by the following details:

a. A return URL to a safe site (https) which will be opened upon authentication Example: https://www.stofnun.is/eydublad

b. Since the service provider's logo will be presented in the authentication window along with the *Ísland.is* logo, and thus the provider's logo must be submitted, presented on a white background with a width of 200 pixels and a height of 60 pixels.

Registers Iceland generates a unique service provider ID. Although this is usually identical to the service provider's main website URL, there may be instances whereby providers have more than one ID.

Examples: skra.is, vmst.is

A service provider which wants to switch from SAML 1.2 to SAML 2.0, using the same service provider ID as before, must be in touch with island@island.is to have the communication method for the ID changed.

2.3 Connecting to the Ísland.is identification and authentication services

a. The service provider begins by forwarding a user to the *Ísland.is* authentication page, with the inclusion of the provider's ID, as in:

   https://innskraning.island.is/?id=skra.is
   https://innskraning.island.is/?id=vmst.is

b. If the service provider wants to decide what authentication choices are offered to the customer, then a parameter is added to the end of the URL the user is sent to (example above).
   &qaa=3 for multi-factor IceKeys or digital certificates, &qaa=4 for digital certificates only.

c. If the service provider wants to make sure the origin of the authentication request was from him, then he can add &authid=<id>, where <id> is in GUID format. Example &authid=5110C405-E94A-4B75-9770-6A4CAB5C7AD4

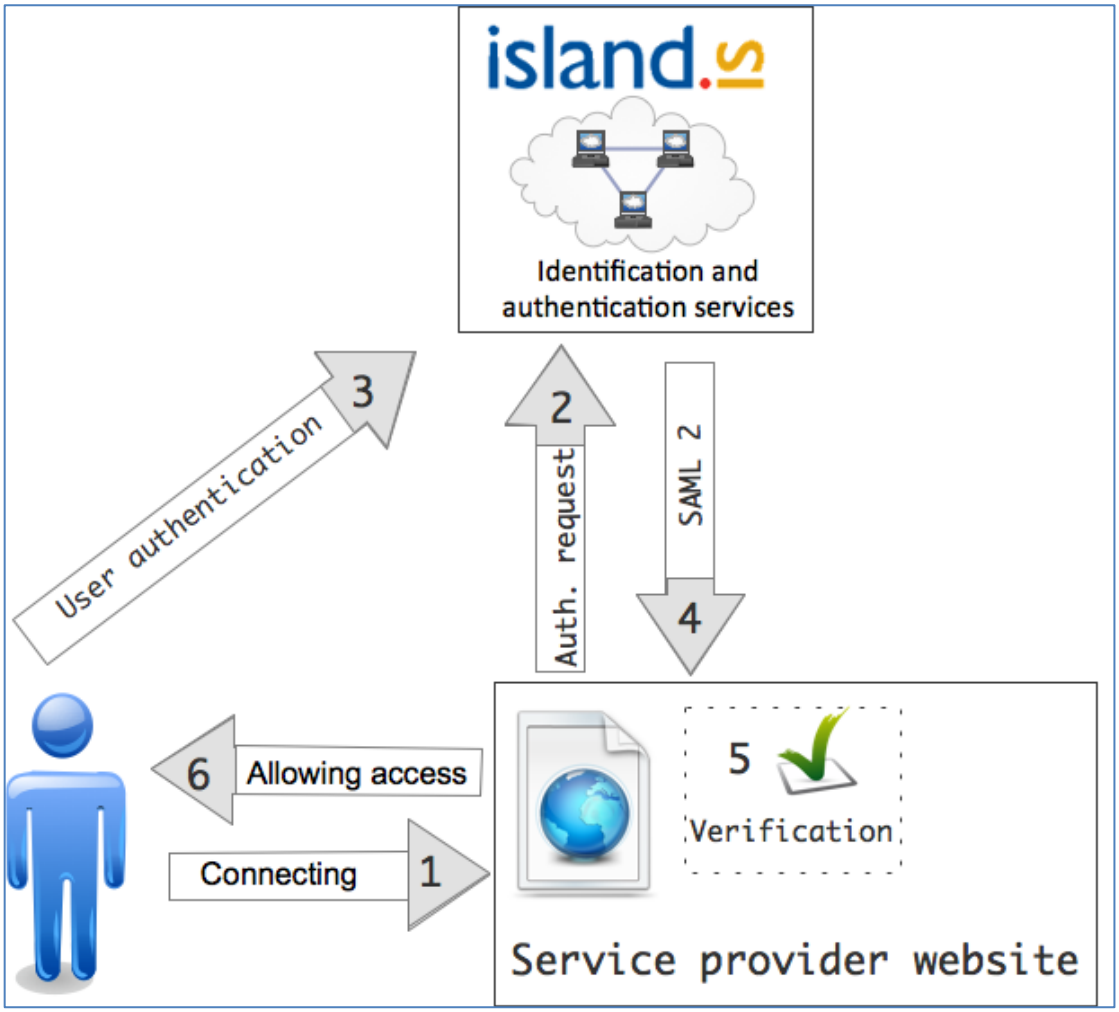
d. Upon completing user authentication, the *Ísland.is* identification and authentication services return a digitally signed SAML 2 token to the service provider site (as a POST parameter, encoded in Base 64 and UTF-8), for decoding by the service provider.

e. Note: It is imperative that the service provider carry out the e-signature validation correctly.

f. Further information on these communications can be found in the sections below.

3 Communications

This section describes how authentication occurs during *Ísland.is* identification and authentication services.



1. The user connects to the service provider's website and requests access to the part of the site that uses IAS at *Ísland.is*.

2. The service provider assesses the authentication strength it will require for allowing access to the relevant material or service at its website, then forwards the user to IAS at *Ísland.is*, together with an authentication request that specifies the minimum authentication strength required. Three levels of strength are offered: digital certificates, IceKey and multi-factor IceKey.

3. The user is then asked for authentication, utilising the means of authentication that meets the minimum requirements expressed in the authentication request. For instance, if a digital certificate is the minimum requirement, it is enough for the user

to authenticate her/himself with a digital certificate. Should the minimum requirement be for IceKey, the user may authenticate her/himself with either IceKey or a digital certificate.

4. The *Ísland.is* IAS services identify and authenticate the user, finding the name in the population register or in the enterprise register. Once the user has been identified and authenticated, the services generate a message consisting of a SAML 2.0 token with a digital signature from the authentication service. Next, this authentication token is transmitted to the return URL specified in IAS.

5. The service provider's website receives the SAML 2.0 authentication token and verifies whether it is reliable and has been digitally signed by *Ísland.is*. If the SAML 2.0 token passes verification, variables are extracted from the token subject. The SAML token is digitally signed by digital certificate for Þjóðskrá Íslands issued by Traustur búnaður. The trust chain can be reached on http://www.audkenni.is/adstod/skilrikjakedjur.cfm, but it must be present in the certificate store of the server which verifies the token. Please note the chain has 3 certificates: Auðkennisrót (root certificate), Traust auðkenni (intermediate certificate), Traustur búnaður (intermediate certificate), which all need to be present on the server processing the SAML token.

6. Based on the SAML 2.0 token subject, the service provider's website determines whether this particular user should have access to the site, and if so what access, before opening the appropriate access. **It is extremely important to verify message signatures and subjects when implementing SAML 2, because the security of identification and authentication depends on correct processing.**

4 The contents of SAML 2.0 messages

Under AttributeStatement, a SAML message always includes the following details about the user:

- The user's Icelandic identification number (UserSSN – Kennitala)
- User (Name – Nafn)
- User authentication (Authentication – Auðkenning) in comprehensible wording, for example "IceKey" or "Digital Certificate"
- User IP (IP Address – IP tala)
- User browser info (UserAgent – NotandaStrengur)
- The recipient's Icelandic identification number (DestinationSSN – Kennitala móttakanda)

If the user identified her/himself by means of an IceKey, information is also transmitted about her/his authentication (KeyAuthentication), indicating how the user was identified when the current IceKey was originally generated. If on the other hand the user identified her/himself by means of an employee certificate, the SAML message includes the Icelandic identification number of the enterprise and its name as a legal entity (CompanySSN and CompanyName).

Since the Attribute Statement may assign certain pre-determined values to the standard variables, this section will now explain these values.

4.1 User's Icelandic identification number

The UserSSN variable (Friendly Name "Kennitala") consists of the Icelandic identification number of the user who is authenticating her/himself.

4.2 User name

The Name variable (Friendly Name "Nafn") consists of the user's name as recorded by Registers Iceland for this individual person or enterprise.

4.3 User authentication

The Authentication variable (Friendly Name "Auðkenning") represents the user's means of identifying her/himself, and can express the following values:

- Digital certificate (Rafræn skilríki) – when the user has authenticated her/himself with such a certificate
- Employee certificate (Rafræn starfsmannaskilríki) – when the user has authenticated her/himself with a digital certificate used for employees
- IceKey (Íslykill) – when the user has authenticated her/himself with an IceKey
- Multi-factor IceKey (Styrktur Íslykill) – when the user has authenticated her/himself with an IceKey as well as with the enforcement code sent to her/his phone or e-mail address
- Multi-factor digital certificate (Styrkt rafræn skilríki) – when the user has authenticated her/himself with a digital certificate as well as the enforcement code sent to her/his phone or e-mail address

- Multi-factor employee certificate (Styrkt rafræn starfsmannaskilríki) – when the user has authenticated her/himself with a digital certificate for employees as well as the enforcement code sent to her/his phone or e-mail address
- Unknown (Óþekkt) – when the authentication could not be identified (this is an error and is not displayed)

4.4 User IP

The IPAddress variable (Friendly Name "IPTala") consists of the user's IP address during authentication. Please note the user may be connecting to the service provider using a different IP address (due to load balancing etc.), so it is not necessarily the same IP address as in the SAML token.

4.5 Browser info

The variable UserAgent (friendly name “NotandaStrengur”) has info about the users browser which was used during authentication. Sample content: „Mozilla/5.0 (Windows NT 6.1; WOW64; rv:26.0) Gecko/20100101 Firefox/26.0“

4.6 Recipient's Icelandic identification number

The DestinationSSN variable (Friendly Name "KennitalaMóttakanda") consists of the Icelandic Id. No. of the Internet service provider.

4.7 IceKey authentication

The KeyAuthentication variable (Friendly Name "VottunÍslykils") is included when a user has authenticated her/himself with an IceKey or a multi-factor IceKey. This variable can express the following values:

- Digital certificate – user who was registered according to a digital certificate when her/his IceKey was generated
- Mailed letter – user who registered her/himself by means of an IceKey sent to her/his domicile when the IceKey was generated
- Online banking documentation – user who registered her/himself by means of an IceKey that was sent to be displayed on her/his personal banking website
- Handed over at Registers Iceland upon the presentation of an official personal identification document
- Handed over by a party collaborating with Registers Iceland, upon the presentation of an official personal identification document
- Letter mailed to an embassy and handed over upon the presentation of an official personal identification document
- Unknown – no information available on which of the above means applied when the IceKey was generated

The subject of the variable remains unchanged unless the user receives a new IceKey by a different one of the above means.

4.8 Legal entity identification numbers

The CompanySSN variable (Friendly Name "KennitalaLögaðila") is included if a user has authenticated her/himself with an employee certificate or a multi-factor employee

certificate. The identification number of the legal entity of the employer is obtained from the user's employee certificate.

4.9 Legal entity names

The CompanyName variable (Friendly Name "NafnLögaðila") is included if a user has authenticated her/himself with an employee certificate or a multi-factor employee certificate. The name of the legal entity of the employer is obtained from the user's employee certificate.

5 SAML message security

A SAML message contains a variety of data concerning both its origin and other aspects that are needed for proper structuring. This data covers for instance the following:

- Issuer
- Validity period (NotBefore and NotAfter attributes which accompany an Assertion or Conditions node)
- Recipient (Audience)
- Authentication (AuthnContextClassRef nodes, which appear under AuthnStatement or AuthnContext)
- Destination (appearing in Response Header and SubjectConfirmationData)
- User IP (the Address attribute of the Subject/SubjectConfirmation/SubjectConfirmationData node)
- Signature information
- Subject Confirmation Data
- Info on the hardware and software the user is using.

The SAML message returned by IAS will be digitally signed with a certificate issued by Auðkenni ehf. (Traustur bunadur). Furthermore, the message will have been transformed with xml-exc-c14n, prior to being digested with SHA256 and signed with a 2048-bit RSA key. The following points must be observed in order to verify message origin:

- That the message was signed with a certificate owned by Registers Iceland (displaying SERIALNUMBER = 6503760649 in the certificate subject)
- That the signature is valid, i.e. that signing was performed with the above certificate
- That the certificate is valid, i.e. that it was issued by Auðkenni ehf. (Traustur bunadur) and has not been revoked.
- That the message is valid, i.e. that its validity period has not expired
- In the original instructions the service providers were encouraged to check if the user IP of the message is the same as that of the user who is authenticating her/himself. This would not work properly for users of big networks, using many proxy servers for different purposes and may very well use one proxy when connecting to the authentication service and another proxy when connecting to the service provider.
- Info on the hardware and software (user agent) of the one connecting to the authentication service and the one connecting to the service provider should match.
- That the recipient is correct, i.e. the service provider that was requesting authentication

Most programming languages can process SAML messages and verify their integrity. In .Net these functions are located in System.Security.Cryptography, while in Java they are in OpenSAML.

6 Development environments on separate domains

If the service provider keeps its development environment at a separate URL (for example under another domain) and wishes to be able to test identification and authentication there, the simplest solution is to request that a special, additional authentication (ID) be set up and used only for testing.

6.1 Example

An entity called Stofnun has a development environment under the domain http://development.stofnun.is and wants to be able to test identification and authentication there. Therefore, a trial authentication is set up, *d.stofnun.is,* which is registered as the return URL http://development.stofnun.is/eydublad (see Example 1 above). This means the link to the authentication page will be http://innskraning.island.is/?id=d.stofnun.is

Note that trial authentications are not required to reflect the actual subordinate domain.

Appendices

1.1 Example of a SAML 2 message

The SAML 2 message with which IAS responds will provide all relevant information on the user and the original authentication. This information is submitted with a digital signature, using the digital certificate of the identification and authentication services. The following is an example of a SAML 2 message:


```
<?xml version="1.0" encoding="UTF-8"?>
<Response xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" ID="_1d62ec4e-ef50-4ca0-ad65-805a126a5e99"
Version="2.0" IssueInstant="2014-01-17T15:15:52.1725761Z"
Destination="https://login.advania.is/islogin.aspx"
xmlns="urn:oasis:names:tc:SAML:2.0:protocol"><Issuer
xmlns="urn:oasis:names:tc:SAML:2.0:assertion">Þjóðskrá Íslands</Issuer><Signature
xmlns="http://www.w3.org/2000/09/xmldsig#"><SignedInfo><CanonicalizationMethod
Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315" /><SignatureMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" /><Reference URI="#_1d62ec4e-ef50-4ca0-
ad65-805a126a5e99"><Transforms><Transform
Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature" /><Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" /></Transforms><DigestMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"
/><DigestValue>uDmYbiiCr3Btq3x2IGvHiXnGeJj0cRCnc9cHreQsFFc=</DigestValue></Reference></SignedI
nfo><SignatureValue>HnFClmDhLmjG/RBeiFiG2XsIB5jhRAAXmAcNKp+uDYnLvwZWYvUAZg3pkOZ9vjy2hpQ8LKY+/A
rXrwL+p6whw8yaHnbvIY9R95GrMhGle5QyDQ9RJB7DvUm4l1Vo/taR4NeB5Y/vpUuaelqeFsz5DY7IzfyfBSkSuydXIPCj
d769uI5Jkpg5O3AhWZ5oS3YjP5bU2ghvgm4lUTckYll5u1OgKGlS/woksl2WrlUcvAw0CFnsz4nlN7HM5DMJ15VMQ96hvh
aGBQiVgf5ZeYXkwFrZjl3P1rVbnp7CQ4pKTsgwj8eE5Canz0ejdeatIpCMdPh1SzbIlIKCmlu2NtrSrg==</SignatureV
alue><KeyInfo><X509Data><X509Certificate>MIIGDDCCBPSgAwIBAgICCbYwDQYJKoZIhvcNAQEFBQAwgZIxCzAJB
gNVBAYTAklTMRMwEQYDVQQFEwo1MjEwMDAyNzkwMRYwFAYDVQQKEw1BdWRrZW5uaSBlaGYuMSMwIQYDVQQLExpVdGdlZmF
uZGkgYnVuYWRhcnNraWxyaWtqYTEWMBQGA1UECxMNTWlsbGlza2lscmlraTEZMBcGA1UEAxMQVHJhdXN0dXIgYnVuYWR1c
jAeFw0xMzA4MTMxMzU2MzFaFw0xNDA4MTMxMzU2MzFaMIHCMQswCQYDVQQGEwJJUzEeMBwGA1UECgwVw55qw7PDsHNrcsO
hIMONc2xhbmRzMRgwFgYDVQQLEw9CdW5hZGFyc2tpbHJpa2kxJTAjBgNVBAsMHFVuZGlycml0dW4gZcOwYSBhdcOwa2Vub
mluZyAxHTAbBgkqhkiG9w0BCQEWDnZlcmtAaXNsYW5kLmlzMRMwEQYDVQQFEwo2NTAzNzYwNjQ5MR4wHAYDVQQDExVJbm5
za3JhbmluZyBJc2xhbmQuaXMwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCkWzKHkw3ckRb55wX9K92Rj4yNv
OIqj9KUYAaO6dmzSRKtZ3APpbsPL+d1f7ObzSyvPZ6hl0H+mgs1l5nsCc4B1E0xR9t7d5AgwF66xYN/zepPfJ95IV3zlmn
nj+CPG4quyJs6HEgyxNH79X9v5lq+xe0WtoHgaY2/vTvceOPOkdksDnCFuD/OvjvVyZ0bTB3Ae0kBNbJRQCAWRpuVSp2I3
xEfXhhqbuZOsv+WUvaOL1januOWYi9d6IoaC8CO3RPRvRcS9q5q/iDu2HYECiqT1ZHXu+u7ZosF0Tn1lt0u//8E6YC051F
+E2XDCtoqiO7QS0sE9Fntxpmbfv0wrlL1AgMBAAGjggI4MIICNDAMBgNVHRMBAf8EAjAAMIIBHAYDVR0gBIIBEzCCAQ8wg
gELBglggmABAgEBBAEwgf0wgcQGCCsGAQUFBwICMIG3GoG0VGhpcyBjZXJ0aWZpY2F0ZSBpcyBpbnRlbmRlZCBmb3IgZGl
naXRhbCBzaWduYXR1cmVzIGFuZCBhdXRoZW50aWNhdGlvbi4gVGhpcyBjZXJ0aWZpY2F0ZSBmdWxmaWxzIHRoZSByZXF1a
XJlbWVudHMgb2Ygbm9ybWFsaXplZCBjZXJ0aWZpY2F0ZSBwb2xpY3kgKE5DUCkgZGVmaW5lZCBpbiBFVFNJIFRTIDEwMiA
wNDIuMDQGCCsGAQUFBwIBFihodHRwOi8vY3AuYXVka2VubmkuaXMvdHJhdXN0dXJidW5hZHVyL2NwMHMGCCsGAQUFBwEBB
GcwZTAjBggrBgEFBQcwAoYXaHR0cDovL29jc3AuYXVka2VubmkuaXMwPgYHYIJgAgFjBoYzaHR0cDovL2NkcC5hdWRrZW5
uaS5pcy9za2lscmlraS90cmF1c3R1cmJ1bmFkdXIucDdiMAsGA1UdDwQEAwIF4DAfBgNVHSMEGDAWgBRv7NsRLAEDaxtgr
rb4aNlAAc2OODBCBgNVHR8EOzA5MDegNaAzhjFodHRwOi8vY3JsLmF1ZGtlbm5pLmlzL3RyYXVzdHVyYnVuYWR1ci9sYXR
lc3QuY3JsMB0GA1UdDgQWBBQPikAxifriUSZrQMrWjxyY2PjEdzANBgkqhkiG9w0BAQUFAAOCAQEABMMaCGABkb40bOrFx
ZIM+YDeYYGne4npdM4QGupkv2aDP2bOhohGw9rCbaxL9lm8BHGHtmWwerwX/3lrtnt367pMc/TRon+vVgnxo9wycmP2u2P
NyXrEI8X297PBJoKlZ/ouDnITVqH/BVvwaPSiYqz78h/kzQRNtFDGLT8O/CqfCRB5qmGNx8I01tcOMq4EHrc/hwHQc1fog
tHJ7JaC5NMO1bg853yl2XC3Dww8OmzAHkCFvKx8aBeEyiesXJHpFj292KohHZhOj1yR5eo9bdqdcjTGkWRn0hN+6vZkdMB
IqhkNS7kyxGpvyniCe3VJZIhaXHIZrbx1aYGlyoCvxg==</X509Certificate></X509Data></KeyInfo></Signatur
e><Status><StatusCode Value="urn:oasis:names:tc:SAML:2.0:status:Success" /></Status><Assertion
Version="2.0" ID="_321427d1-3a6c-4d56-9ada-8d90924f6014" IssueInstant="2014-01-
17T15:15:52.1725761Z" xmlns="urn:oasis:names:tc:SAML:2.0:assertion"><Issuer>Þjóðskrá
Íslands</Issuer><Subject><NameID
NameQualifier="island.is">Innskráningarþjónusta</NameID><SubjectConfirmation
Method="urn:oasis:names:tc:SAML:2.0:cm:bearer"><SubjectConfirmationData Address="127.0.0.1"
NotOnOrAfter="2014-01-17T15:25:52.1745763Z" Recipient="http://localhost:63841/GetBack.aspx"
/></SubjectConfirmation></Subject><Conditions NotBefore="2014-01-17T15:15:52.1745763Z"
NotOnOrAfter="2014-01-
17T15:25:52.1745763Z"><AudienceRestriction><Audience>login.advania.is</Audience></AudienceRest
riction></Conditions><AuthnStatement AuthnInstant="2014-01-
17T15:15:52.1725761Z"><SubjectLocality Address="212.30.225.139"
/><AuthnContext><AuthnContextClassRef>urn:oasis:names:tc:SAML:2.0:ac:classes:TLSClient</AuthnC
```

ontextClassRef></AuthnContext></AuthnStatement><AttributeStatement><Attribute Name="UserSSN"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic"
FriendlyName="Kennitala"><AttributeValue
xsi:type="xsd:string">1234567890</AttributeValue></Attribute><Attribute Name="Name"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic"
FriendlyName="Nafn"><AttributeValue xsi:type="xsd:string">Jón
Jónsson</AttributeValue></Attribute><Attribute Name="Authentication"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic"
FriendlyName="Auðkenning"><AttributeValue xsi:type="xsd:string">Rafræn
skilríki</AttributeValue></Attribute><Attribute Name="IPAddress"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic"
FriendlyName="IPTala"><AttributeValue
xsi:type="xsd:string">127.0.0.1</AttributeValue></Attribute><Attribute Name="UserAgent"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic"
FriendlyName="NotandaStrengur"><AttributeValue xsi:type="xsd:string">Mozilla/5.0 (Windows NT
6.1; WOW64; rv:26.0) Gecko/20100101 Firefox/26.0</AttributeValue></Attribute><Attribute
Name="AuthID" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic"
FriendlyName="AuðkenningarNúmer"><AttributeValue xsi:type="xsd:string">0807DA8D-3299-4FF4-
BEB2-54727A50FFBD</AttributeValue></Attribute><Attribute Name="DestinationSSN"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic"
FriendlyName="KennitalaMóttakanda"><AttributeValue
xsi:type="xsd:string">5902697199</AttributeValue></Attribute></AttributeStatement></Assertion>
</Response>.

1.2 Example using .Net

For a SAML message to be processed in .Net, it must first be downloaded in XmlDocument and from there into SignedXml. Conventional SignedXml methods can then be applied for verifying the message. Please note the token from the authentication service is coded in Bas64.

```
using System.Security.Cryptography.Xml;
using System.Security.Cryptography.X509Certificates;
using System.Xml;
using System.Xml.Serialization;


public static bool verifySamlSimple(string samlString, string ip,
string ua, string authId, out string message)
 {
 message = "";
 XmlDocument doc = new XmlDocument();
 doc.PreserveWhitespace = true;
 doc.LoadXml(samlString);

 SignedXml signedXml = new SignedXml(doc);
 //Sækjum undirritun og skoðum
 XmlNodeList nodeList = doc.GetElementsByTagName("Signature");
 XmlNodeList certList = doc.GetElementsByTagName("X509Certificate");
 signedXml.LoadXml((XmlElement)nodeList[0]);
 byte[] certData = Encoding.Default.GetBytes(certList[0].InnerText);
 X509Certificate2 cert = new X509Certificate2(certData);

 //Hér er mikilvægt að setja ekki true í seinna gildi nema útfærð sé
 //aðgerð sem sannreynir gildi skilríkis sérstaklega
 bool signatureOk = signedXml.CheckSignature(cert, false);
 if (signatureOk)
 message = "Signature OK. ";
 else
message = "Signature not OK. ";
```

```
 bool certOk = false;
 if (cert.Issuer.StartsWith("CN=Traustur bunadur") &&
 cert.Subject.Contains("SERIALNUMBER=6503760649"))
 {
 certOk = true;
 message += "Certificate is OK. ";
 }
 else
 message += "Certificate not OK. ";


 DateTime nowTime = DateTime.UtcNow;
 //Sækjum tíma í Conditions og berum saman
 XmlNodeList condNodeList = doc.GetElementsByTagName("Conditions");
 DateTime fromTime =
DateTime.Parse(condNodeList[0].Attributes["NotBefore"].Value);
 DateTime toTime =
DateTime.Parse(condNodeList[0].Attributes["NotOnOrAfter"].Value);

 bool timeOk = false;
 if (nowTime > fromTime && toTime > nowTime)
 {
 timeOk = true;
 message += "SAML time valid. ";
 }
 else if (nowTime < fromTime)
 message += "From time has not passed yet. ";
 else if (nowTime > toTime)
 message += "To time has passed. ";

 //Skoðum nú IP tölu notanda, notandastreng og auth id úr Attributes
 bool ipOk = false;
 bool uaOk = false;
 bool authidOk = false;
 XmlNodeList attrList =
 doc.GetElementsByTagName("Attribute");
 if (attrList.Count > 0)
 {
 foreach (XmlNode attrNode in attrList)
 {
 XmlAttributeCollection attrCol = attrNode.Attributes;
//Staðfestum að IP tala sé sú sama – athugið að ekki er
 //alltaf hægt að treysta á að IP tala sé sú sama
 if (attrCol["Name"].Value.Equals("IPAddress"))
 {
 ipOk = attrNode.FirstChild.InnerText.Equals(ip);
message += "Correct client IP. ";
 }
//Staðfestum að user agent strengur sé sá sami
if (attrCol["Name"].Value.Equals("UserAgent"))
 {
 uaOk = attrNode.FirstChild.InnerText.Equals(ua);
message += "Correct client user agent. ";
 }
//Staðfestum að auðkenningarnúmer sé það sama
if (attrCol["Name"].Value.Equals("AuthID"))
 {
```

```
 authidOk =
attrNode.FirstChild.InnerText.Equals(authId);
 message += "Correct client auth ID. ";
 }
 }
 if (!ipOk)
 message += "Incorrect client IP. ";
 if (!uaOk)
 message += "Incorrect client user agent. ";
 if (!uaOk)
 message += "Incorrect auth ID. ";
 }
 else
 message += "No Attributes found";

// Skeytið er í lagi ef undirritun, skilríki, tímar eru í lagi
// ásamt ip-tölu, notandastreng eða auðkenningarnúmer.
 return signatureOk && certOk && timeOk &&
(ipOk || uaOk || authidOk);
 }
```

1.3 Example using Java

To process a SAML message in Java, it must be downloaded and unmarshalled in SignableSAMLObject. This will allow the integrity of certificates, etc., to be verified.

```
KeyStore trustStore;

public boolean validateSaml(final String samlString, final String userIP,
final String userAgent, final String authId)
{
 boolean ok = false;

 try {
 SignableSAMLObject signedObject =
 (SignableSAMLObject)this.unmarshall(samlString);

 if (signedObject != null)
 {
 SignableSAMLObject samlObject =
 (SignableSAMLObject)this.validateSignature(signedObject,
trustStore);
 if (samlObject != null)
 {
 Assertion assertion = this.getAssertion((Response)samlObject,
userIP,
 false);
 if(assertion!=null){
 final DateTime serverDate = new DateTime();

 if
(assertion.getConditions().getNotBefore().isAfter(serverDate))
 {
 throw new Exception("Token date valid yet (getNotBefore = " +
assertion.getConditions().getNotBefore()

 + " ), server_date: " + serverDate);
 }
 if
```

```
(assertion.getConditions().getNotOnOrAfter().isBefore(serverDate)) {
 throw new Exception("Token date expired (getNotOnOrAfter = "
 + assertion.getConditions().getNotOnOrAfter()
 + " ), server_date: " + serverDate);
 }
 // Validate the assertions for IP, useragent and authId.
 validateAssertion(assertion, userIP, userAgent, authId);

 ok = true;
 }
 }
 }
 } catch (Exception e) {
 //SAML not verified
 e.printStackTrace();
 }

 return ok;
}

//Unmarshall SAML string
private final XMLObject unmarshall(final String samlString) throws
Exception {
 try {
 byte[] samlToken = Base64.base64ToByteArray(samlString);
 final BasicParserPool ppMgr = new BasicParserPool();
 final HashMap<String, Boolean> features = new HashMap<String,
Boolean>();
 features.put(XMLConstants.FEATURE_SECURE_PROCESSING, Boolean.TRUE);
 ppMgr.setBuilderFeatures(features);
 ppMgr.setNamespaceAware(true);

 Document document = ppMgr.parse(new ByteArrayInputStream(samlToken));
 if (document != null){
 final Element root = document.getDocumentElement();
 final UnmarshallerFactory unmarshallerFact =
 Configuration.getUnmarshallerFactory();
 if (unmarshallerFact != null && root != null)
 {
 final Unmarshaller unmarshaller =
unmarshallerFact.getUnmarshaller(root);
 try {
 return unmarshaller.unmarshall(root);
 } catch (NullPointerException e){
 throw new Exception("NullPointerException", e);
 }
 } else {
 throw new Exception("NullPointerException : unmarshallerFact or
root is
 null");
 }
 } else {
 throw new Exception("NullPointerException : document is null");
 }
 } catch (XMLParserException e) {
 throw new Exception(e);
 } catch (UnmarshallingException e) {
throw new Exception(e);
 } catch (NullPointerException e) {
 throw new Exception(e);
 }
}
```

```
private final SAMLObject validateSignature(final SignableSAMLObject
tokenSaml, KeyStore keyStore) throws Exception {
 try {
 // Validate structure signature
 final SAMLSignatureProfileValidator sigProfValidator =
 new SAMLSignatureProfileValidator();
 try {
 // Indicates signature id conform to SAML Signature profile
 sigProfValidator.validate(tokenSaml.getSignature());
 } catch (ValidationException e) {
 //ValidationException: signature isn't conform to SAML Signature
profile.
 throw new Exception(e);
 }

 String aliasCert = null;
 X509Certificate certificate;

 final KeyInfo keyInfo = tokenSaml.getSignature().getKeyInfo();

 final org.opensaml.xml.signature.X509Certificate xmlCert = keyInfo
 .getX509Datas().get(0).getX509Certificates().get(0);

 final CertificateFactory certFact = CertificateFactory
 .getInstance("X.509");
 final ByteArrayInputStream bis = new ByteArrayInputStream(Base64.
 base64ToByteArray(xmlCert.getValue()));
 final X509Certificate cert = (X509Certificate) certFact
 .generateCertificate(bis);

 // Exist only one certificate
 final BasicX509Credential entityX509Cred = new BasicX509Credential();
 entityX509Cred.setEntityCertificate(cert);

 try {
 cert.checkValidity();
 } catch (CertificateExpiredException exp) {
 throw new Exception("Certificate expired.");
 } catch (CertificateNotYetValidException exp) {
 throw new Exception("Certificate not yet valid.");
 }

 boolean trusted = false;

 for (final Enumeration<String> e = keyStore.aliases();
e.hasMoreElements();)
 {
 aliasCert = e.nextElement();
 certificate = (X509Certificate) keyStore.getCertificate(aliasCert);

 try {
 cert.verify(certificate.getPublicKey());
 trusted = true;
 break;
 } catch (Exception ex) {
 //Do nothing - cert not trusted yet
}
 }

 if (!trusted)
 throw new Exception("Certificate is not trusted.");
 else {
 if
```

```
(cert.getSubjectDN().toString().contains("SERIALNUMBER=6503760649")
 && cert.getIssuerDN().toString().startsWith("CN=Traustur
bunadur"))
 trusted = true;
 else {
 throw new Exception("Certificate is not trusted.");
 }
 }

 // Validate signature
 final SignatureValidator sigValidator = new SignatureValidator(
 entityX509Cred);
 sigValidator.validate(tokenSaml.getSignature());

 } catch (Exception e) {
 e.printStackTrace();
 }
 return tokenSaml;
}

private Assertion getAssertion(final Response samlResponse,
 final String userIP, final boolean ipValidate) throws Exception {
 if (samlResponse.getAssertions() == null
 || samlResponse.getAssertions().isEmpty()) {
 //Assertion is null or empty
 return null;
 }

 final Assertion assertion = (Assertion)
samlResponse.getAssertions().get(0);

 for (final Iterator<SubjectConfirmation> iter = assertion.getSubject()
 .getSubjectConfirmations().iterator(); iter.hasNext();) {
 final SubjectConfirmation element = iter.next();
 final boolean isBearer = SubjectConfirmation.METHOD_BEARER
 .equals(element.getMethod());

 if (ipValidate) {
 if (isBearer) {
 if (StringUtils.isBlank(userIP)) {
 throw new Exception("browser_ip is null or empty.");
 } else if (StringUtils.isBlank(element
 .getSubjectConfirmationData().getAddress())) {
 throw new Exception("token_ip attribute is null or empty.");
 }
 }
 final boolean ipEqual = element.getSubjectConfirmationData()
 .getAddress().equals(userIP);

 // Validation ipUser
 if (!ipEqual && ipValidate) {
 throw new Exception("IPs doesn't match : token_ip ("
 + element.getSubjectConfirmationData().getAddress()
 + ") browser_ip ("+ userIP + ")");
 }
}
 }
 return assertion;
}

/**
 * Validate assertions for IP, user agent and auth ID
 * @param assertion The assertion to validate
```

```
 * @param ip The user IP
 * @param ua The users user agent
 * @param authId The auth ID
 * @throws Exception
 */
private void validateAssertion(final Assertion assertion, String ip,
String ua, String authId ) throws Exception {
final List<XMLObject> listExtensions =
assertion.getOrderedChildren();

 boolean find = false;
 AttributeStatement requestedAttr = null;

 // Search the attribute statement.
 for (int i = 0; i < listExtensions.size() && !find; i++) {
 final XMLObject xml = listExtensions.get(i);
 if (xml instanceof AttributeStatement) {
 requestedAttr = (AttributeStatement) xml;
 find = true;
 }
 }

 if (!find) {
 throw new Exception (
 "AttributeStatement it's not present.");
 }

 final List<Attribute> reqAttrs = requestedAttr.getAttributes();

 String attributeName, tempValue;
 XMLObject xmlObj;
 boolean ipOk = false, uaOk = false, authIdOk = false;

 // Process the attributes.
 for (int nextAttribute = 0; nextAttribute < reqAttrs.size();
nextAttribute++) {
 final Attribute attribute = reqAttrs.get(nextAttribute);

 attributeName = attribute.getName();
 if (attributeName.equals("IPAddress"))
 {
 xmlObj = attribute.getOrderedChildren().get(0);
 tempValue = ((XSStringImpl) xmlObj).getValue();
 ipOk = tempValue.equals(ip);
 }
 if (attributeName.equals("UserAgent"))
 {
 xmlObj = attribute.getOrderedChildren().get(0);
 tempValue = ((XSStringImpl) xmlObj).getValue();
 uaOk = tempValue.equals(ua);
 }
 if (attributeName.equals("AuthID"))
 {
 xmlObj = attribute.getOrderedChildren().get(0);
tempValue = ((XSStringImpl) xmlObj).getValue();
 authIdOk = tempValue.equals(ip);
 }
 }
 if (ipOk || authIdOk || uaOk)
 System.out.println("Assertion valid.");
 else
 throw new Exception
 (String.format("Assertions not valid. IP valid %b, "
```

```
+ "user agent" valid %b, auth ID valid %b",
 ipOk, uaOk, authIdOk));
}
```

1.4 Example using PHP

In order to verify a SAML message in PHP, use xmlseclibs (https://code.google.com/p/xmlseclibs/).

```
        function verifySaml()
        {
                include 'xmlseclibs.php';
                $token = $_POST["token"];
                if ($token != NULL)
                {
                    $xmlDoc = new DOMDocument();
                    $saml = base64_decode($token);
                    $xmlDoc->loadXML($saml);
                    $xmlsec = new XMLSecurityDSig();
                    $objXMLSecDSig = new XMLSecurityDSig();

                    $objDSig = $objXMLSecDSig->locateSignature($xmlDoc);

                    if ($objDSig == NULL) {
                            throw new Exception("Cannot locate Signature Node");
                    }
                    $objXMLSecDSig->canonicalizeSignedInfo();
                    $objXMLSecDSig->idKeys = array('ID');
                    $objXMLSecDSig->idNS           =           array('wsu'=>'http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd');
                    $retVal = $objXMLSecDSig->validateReference();

                    if ($retVal == NULL) {
                            throw new Exception("Reference Validation Failed");
                    }

                    if (!VerifyDate($xmlDoc))
                    {
                        throw new Exception("Conditions not valid.");
                    }

                    if (!verifyConditions($xmlDoc, get_client_ip()))
                    {
                        throw new Exception("Invalid client ip.");
                    }

                    $objKey = $objXMLSecDSig->locateKey();
                    if (! $objKey ) {
                            throw new Exception("Key not found");
                    }
                    $key = NULL;

                    $objKeyInfo = XMLSecEnc::staticLocateKeyInfo($objKey, $objDSig);

                    if (!verifyCert($objKeyInfo))
                    {
                        throw new Exception("Certificate not valid");
                    }

                    if (! $objKeyInfo->key && empty($key)) {
                            $objKey->loadKey(dirname(__FILE__) . '/mycert.pem', TRUE);
                    }
```

```
                if (!$objXMLSecDSig->verify($objKey)) {
                        throw new Exception("Signature invalid!");
                }
                else {
                    return "Signature valid.";
                }
            }
        }

        function locateConditions($doc)
        {
            $xpath = new DOMXPath($doc);
            $xpath->registerNamespace('assertion',
'urn:oasis:names:tc:SAML:2.0:assertion');
            $nodeset = $xpath->query(".//assertion:Assertion/assertion:Conditions", $doc);
            return $nodeset->item(0);
        }

        function locateSubjectConfirmation($doc)
        {
            $xpath = new DOMXPath($doc);
            $xpath->registerNamespace('assertion',
'urn:oasis:names:tc:SAML:2.0:assertion');
            $nodeset                                =                                $xpath-
>query(".//assertion:Assertion/assertion:Subject/assertion:SubjectConfirmation/assertion:Subje
ctConfirmationData", $doc);
            return $nodeset->item(0);
        }

        function verifyConditions($doc, $ip)
        {
            $conditions = locateSubjectConfirmation($doc);
            if (!$conditions)
            {
                    throw new Exception("Unable to locate Conditions");
                    return false;
            }

            try
            {
                    $address = $conditions->getAttribute('Address');
            }
            catch (Exception $e)
            {
                throw new Exception("Exception while locating address");
                return false;
            }

            if (!strcmp($ip, "::1"))
                $ip = "127.0.0.1";

            if (strcmp($address, $ip))
            {
                throw new Exception("Invalid IP address.");
                return false;
            }
            else
                return true;

        }

        function VerifyDate($doc)
        {
            try
            {
                    $conditions = locateConditions($doc);
            }
            catch (Exception $e)
            {
```

```
                    throw new Exception("Exception while locating Conditions");
                    return false;
                }

                if (!$conditions)
                {
                        throw new Exception("Unable to locate Conditions");
                        return false;
                }

                try
                {
                        $start = $conditions->getAttribute('NotBefore');
                        $end = $conditions->getAttribute('NotOnOrAfter');
                }
                catch (Exception $e)
                {
                    throw new Exception("Exception while locating start or end");
                    return false;
                }

                if (!$start || !$end)
                {
                        throw new Exception("Unable to locate start (NotBefore) or end
(NotOnOrAfter)");
                        return false;
                }

                date_default_timezone_set('Atlantic/Reykjavik');
                $startTime = strtotime($start);
                $endTime = strtotime($end);

                if (!is_int($startTime) || !is_int($endTime))
                {
                        throw new Exception("Unable to get time from start (NotBefore) or end
(NotOnOrAfter)");
                        return false;
                }

                $now = date(time());

                $inSpan = $startTime < $now && $now < $endTime;

                if (!$inSpan)
                {
                        throw new Exception("Response is not within specified timeframe");
                        return false;
                }

                return true;
            }

            function verifyCert($objKeyInfo)
            {
                $caFile = file_get_contents("TrausturBunadur.pem");
                $caCert = openssl_x509_read($caFile);
                $caCertParsed = openssl_x509_parse($caCert, true);
                $parsed = openssl_x509_parse($objKeyInfo->getX509Certificate());

                date_default_timezone_set('Atlantic/Reykjavik');
                $dateFrom = date($parsed['validFrom_time_t']);
                $dateTo = date($parsed['validTo_time_t']);
                $nowTime = date(time());
                if ($nowTime < $dateFrom || $nowTime > $dateTo)
                {
                    throw new Exception("Certificate expired or not valid yet");
                }

                $kennitala = $parsed['subject']['serialNumber'];
```

```
    $issuer = $parsed['issuer']['CN'];

    if ($kennitala != "6503760649")
    {
        throw new Exception("Ekki rétt kennitala í undirritunarskilríki");
        return false;
    }

    if ($issuer!= "Traustur bunadur")
    {
        throw new Exception("Ekki réttur útgefandi undirritunarskilríkis");
        return false;
    }

    $subjectKey = $caCertParsed['extensions']['subjectKeyIdentifier'];
    $authKey = $parsed['extensions']['authorityKeyIdentifier'];
    $authKey = str_replace('keyid:', '', $authKey);
    if (!strcasecmp($subjectKey,$authKey))
    {
        throw new Exception("Not correct CA");
        return false;
    }

    return true;
}

function get_client_ip() {
    $ipaddress = '';
    if (!empty($_SERVER['HTTP_CLIENT_IP']))
        $ipaddress = $_SERVER['HTTP_CLIENT_IP'];
    else if(!empty($_SERVER['HTTP_X_FORWARDED_FOR']))
        $ipaddress = $_SERVER['HTTP_X_FORWARDED_FOR'];
    else if(!empty($_SERVER['HTTP_X_FORWARDED']))
        $ipaddress = $_SERVER['HTTP_X_FORWARDED'];
    else if(!empty($_SERVER['HTTP_FORWARDED_FOR']))
        $ipaddress = $_SERVER['HTTP_FORWARDED_FOR'];
    else if(!empty($_SERVER['HTTP_FORWARDED']))
       $ipaddress = $_SERVER['HTTP_FORWARDED'];
    else if(!empty($_SERVER['REMOTE_ADDR']))
        $ipaddress = $_SERVER['REMOTE_ADDR'];
    else
        $ipaddress = 'UNKNOWN';

    return $ipaddress;
}
```