SGCWEBSOCKETS · TECHNICAL DOCUMENT

SAML Service Provider

TsgcSAMLServiceProvider: SAML 2.0 single sign-on for Delphi and C++ Builder web applications, with full response validation.

eSeGeCe

WWW.ESEGECE.COM

Overview

Plug your Delphi web application into the corporate identity provider. Users sign in once with Microsoft Entra ID, Okta, AD FS, Google Workspace or Keycloak, and your server receives a signed, fully validated identity.

TsgcSAMLServiceProvider adds SAML 2.0 single sign-on to your web applications. It implements the service provider (SP) side of the SAML 2.0 Web Browser SSO profile: it creates the **AuthnRequest** sent to the identity provider (IdP), publishes the service provider metadata and validates the signed **Response** that the browser posts back to the Assertion Consumer Service (ACS) URL.

The component is not an HTTP server. Host the login, ACS and metadata URLs in any HTTP server, for example **TsgcWebSocketHTTPServer** or a **TsgcHTTPServer**, and call the component from the request handler. It works with Microsoft Entra ID, Okta, AD FS, Google Workspace, Keycloak and any other SAML 2.0 identity provider.

At a glance

COMPONENT CLASS

`TsgcSAMLServiceProvider`

STANDARDS / SPEC

SAML 2.0 (OASIS), Web Browser SSO profile

TRANSPORTS

HTTP-Redirect and HTTP-POST bindings, hosted in your HTTP server

PLATFORMS

Windows, macOS, Linux, iOS, Android

FRAMEWORKS

VCL, FireMonkey

EDITION

Enterprise (also sgcAuth pack)

Features

- **Metadata both ways.** `GetMetadata` produces the SP metadata to register in the identity provider. `LoadIdPMetadata` reads the IdP metadata and fills `IdPEntityID`, `IdPSSOURL`, `IdPSSOBinding` and `IdPCertificates`.

- **Redirect and POST bindings.** `GetAuthnRequestRedirectURL` returns the HTTP-Redirect URL with the deflated AuthnRequest, `GetAuthnRequestPostForm` an auto-submitted HTTP-POST form. `SignAuthnRequests` signs them with `SPCertificate` and `SPPrivateKey`.
- **Trusted keys only.** Signatures are checked only against `IdPCertificates`, a certificate embedded in the message is never trusted. RSA-SHA256, RSA-SHA384 and RSA-SHA512 with exclusive canonicalization. SHA-1 is rejected unless `AllowSHA1` is set.
- **Signature wrapping protection.** The response must hold exactly one assertion as a direct child, and the signature must reference that element, which defeats the XML signature wrapping attacks that broke many SAML libraries.
- **Full assertion checks.** `ProcessResponse` validates issuer, audience, destination, `InResponseTo` and the validity window with `ClockSkew` and `MaxAssertionAge`, and keeps a replay cache of assertion IDs. IdP-initiated sign in stays off until you set `AllowIdPInitiated`.
- **Clean result object.** `TsgcSAMLResult` returns `NameID`, `NameIDFormat`, `SessionIndex`, `AuthnInstant`, the `Attributes` with their friendly names, and a readable `ErrorMessage` when something is wrong.

Technical specification

Standards & specs	SAML 2.0 Core · SAML 2.0 Bindings · SAML 2.0 Profiles · SAML 2.0 Metadata · XML Signature Syntax and Processing (W3C) · Exclusive XML Canonicalization (W3C)
Component class	`TsgcSAMLServiceProvider` (unit `sgcAuth_SAML_SP`)
Frameworks	VCL, FireMonkey
Platforms	Windows, macOS, Linux, iOS, Android
Edition	sgcWebSockets Enterprise, also sold in the sgcAuth pack

Main properties

The published and public properties used to configure and drive the component.

`AllowIdPInitiated`	Accepts unsolicited responses started by the identity provider.
`AllowSHA1`	Accepts RSA-SHA1 signatures and SHA-1 digests.
`AssertionConsumerServiceURL`	URL of the Assertion Consumer Service where the IdP posts the SAML response.
`ClockSkew`	Tolerance in seconds for the clock difference between the IdP and this server.
`EntityID`	Unique identifier of the service provider (SP Entity ID).
`IdPCertificates`	Trusted signing certificates of the identity provider.
`IdPEntityID`	Entity ID of the identity provider, the expected Issuer of the responses.
`IdPSSOBinding`	Binding of the IdP single sign-on URL: HTTP-Redirect or HTTP-POST.
`IdPSSOURL`	Single sign-on URL of the identity provider, destination of the AuthnRequests.
`MaxAssertionAge`	Maximum age in seconds of an Assertion, measured from its IssueInstant.

`NameIDFormat`	Name identifier format requested in the AuthnRequest and published in the metadata.
`OpenSSL_Options`	OpenSSL version and library path used to verify and create signatures.
`SignAuthnRequests`	Signs the AuthnRequests sent with the HTTP-Redirect binding.
`SPCertificate`	PEM certificate of the service provider, published in the metadata.
`SPPrivateKey`	PEM RSA private key used to sign the AuthnRequests.
`WantAssertionsSigned`	Requires the Assertion itself to carry a valid signature.

Main methods

The public methods exposed by the component.

`ClearReplayCache`	Removes every assertion ID from the replay cache.
`GetAuthnRequestPostForm`	Creates an AuthnRequest and returns an HTML page that posts it to the IdP.
`GetAuthnRequestRedirectURL`	Creates an AuthnRequest and returns the IdP URL for the HTTP-Redirect binding.
`GetMetadata`	Returns the SAML metadata of the service provider.
`LoadIdPMetadata`	Configures the identity provider from its SAML metadata document.
`ProcessResponse`	Validates the SAML response posted to the ACS URL and returns the authenticated user.

Events

The events fired by the component.

`OnSAMLError`	Fires when ProcessResponse rejects a SAML response.

Quick Start

Set EntityID and AssertionConsumerServiceURL, load the IdP metadata, then serve the metadata, the login redirect and the Assertion Consumer Service from your HTTP handler.

> **About this scenario.** Three URLs and one validation call. The same code ships in the demo `Demos\26.Authentication\03.SAML_ServiceProvider` .

Delphi (VCL / FireMonkey)

```
uses
  sgcAuth_SAML_SP;

// SAML is a form field: SAML: TsgcSAMLServiceProvider;
procedure TForm1.ConfigureSAML(const aIdPMetadataXML: string);
begin
  SAML := TsgcSAMLServiceProvider.Create(nil);
  SAML.EntityID := 'https://app.example.com/saml/metadata';
  SAML.AssertionConsumerServiceURL := 'https://app.example.com/saml/acs';
  // entity ID, SSO URL, binding and signing certificates of the IdP
  SAML.LoadIdPMetadata(aIdPMetadataXML);
end;

// GET /saml/metadata: return SAML.GetMetadata and register it in the IdP

// GET /saml/login: redirect the browser to the IdP
function TForm1.LoginURL(const aRelayState: string;
  out aRequestID: string): string;
begin
  // keep aRequestID for this RelayState, the ACS needs it
  Result := SAML.GetAuthnRequestRedirectURL(aRelayState, aRequestID);
end;

// POST /saml/acs: validate the signed response posted by the browser
function TForm1.ValidateResponse(const aSAMLResponse, aRelayState,
  aRequestID: string): string;
var
  oResult: TsgcSAMLResult;
begin
  oResult := TsgcSAMLResult.Create;
  try
    if SAML.ProcessResponse(aSAMLResponse, aRelayState, aRequestID, oResult) then
      Result := oResult.NameID // plus oResult.Attributes and SessionIndex
    else
      raise Exception.Create(oResult.ErrorMessage);
  finally
    oResult.Free;
  end;
end;
```

C++ Builder

```
// uses: sgcAuth_SAML_SP
TsgcSAMLServiceProvider *SAML = new TsgcSAMLServiceProvider(this);
SAML->EntityID = "https://app.example.com/saml/metadata";
SAML->AssertionConsumerServiceURL = "https://app.example.com/saml/acs";
SAML->LoadIdPMetadata(IdPMetadataXML);

// GET /saml/login
String RequestID;
String URL = SAML->GetAuthnRequestRedirectURL(RelayState, RequestID);

// POST /saml/acs
TsgcSAMLResult *SAMLResult = new TsgcSAMLResult();
if (SAML->ProcessResponse(SAMLResponse, RelayState, RequestID, SAMLResult))
  ShowMessage(SAMLResult->NameID);
delete SAMLResult;
```

Common scenarios

The following topics come from the online help. Each one explains a part of the component and shows the Delphi code that drives it.

1 · SP-initiated sign in, step by step

1. **Configure the service provider.** Set **EntityID** (the unique name of your application, usually the metadata URL) and **AssertionConsumerServiceURL** (the https URL where the IdP posts the response). Serve **GetMetadata** and register it in the identity provider.
2. **Configure the identity provider.** Call **LoadIdPMetadata** with the IdP metadata, or set **IdPEntityID**, **IdPSSOURL** and **IdPCertificates** by hand.
3. **Send the AuthnRequest.** When an anonymous user opens the login URL, call **GetAuthnRequestRedirectURL** and answer with a 302 redirect (HTTP-Redirect binding), or return the HTML page of **GetAuthnRequestPostForm**, which posts the request to the IdP (HTTP-POST binding).
4. **Keep the request id.** Both methods return the id of the new AuthnRequest in `aRequestID`. Store it on the server, for example in a list keyed by a random RelayState value or by the session cookie. Remove it when the response arrives, so every request can be answered only once.
5. **Process the response at the ACS URL.** The browser posts the form fields `SAMLResponse` and `RelayState`. Look up the stored request id and call **ProcessResponse**. When it returns True the user is authenticated: read **NameID**, **SessionIndex** and **Attributes** from the TsgcSAMLResult and create your own session. When it returns False, **ErrorMessage** tells why and **OnSAMLError** fires.

The RelayState is not covered by the signature of the IdP. Use it only as a key to find your own pending request, never as a URL to redirect to without checking it.

2 · Configure the identity provider

The easiest way is the metadata document of the IdP. **LoadIdPMetadata** reads the entity ID, the `SingleSignOnService` URL (HTTP-Redirect preferred, HTTP-POST otherwise) and every signing certificate, and fills **IdPEntityID**, **IdPSSOURL**, **IdPSSOBinding** and **IdPCertificates**. Download the metadata over https with any HTTP client, or save it to a file. When the document is an `EntitiesDescriptor` with several entities, pass the entity ID of the IdP in the second parameter.

Without metadata, set the same properties by hand with the values shown in the IdP console:

```
Delphi (VCL / FireMonkey)
```

```
oSAML.IdPEntityID := 'https://sts.windows.net/00000000-0000-0000-0000-000000000000/';
oSAML.IdPSSOURL := 'https://login.microsoftonline.com/00000000-0000-0000-0000-000000000000/saml2
oSAML.IdPSSOBinding := samlbHTTPRedirect;
// ... one certificate per item: a PEM or the Base64 body of the certificate
oSAML.IdPCertificates.Text := LoadIdPSigningCertificate;
```

Notes for the most common identity providers. In all of them the two values to register are the **SP Entity ID** (**EntityID**) and the **ACS URL** (**AssertionConsumerServiceURL**). Keep assertion encryption disabled, encrypted assertions are not supported.

- **Microsoft Entra ID.** Enterprise applications, New application, Create your own application (non-gallery). In Single sign-on choose SAML and upload the SP metadata, or fill Identifier (Entity ID) and Reply URL (Assertion Consumer Service URL). Assign users or groups. Load the *App Federation Metadata Url* shown in SAML Certificates. Entra ID signs the assertion with RSA-SHA256 by default. The attributes use claim URIs as names, for example `http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress`.
- **Okta.** Applications, Create App Integration, SAML 2.0. Single sign-on URL is the ACS URL (keep "Use this for Recipient URL and Destination URL" checked) and Audience URI is the SP Entity ID. Choose the Name ID format and add attribute statements such as email, firstName and lastName. Assign people or groups and load the Metadata URL of the Sign On tab.
- **AD FS.** Add a claims aware Relying Party Trust and import the SP metadata file, or enter the SAML 2.0 WebSSO URL (the ACS URL) and the relying party identifier (the SP Entity ID) by hand. AD FS only accepts https endpoints. Add a claim rule that sends a Name ID, for example "Send LDAP Attributes as Claims" (E-Mail-Addresses to E-Mail Address) followed by "Transform an Incoming Claim" (E-Mail Address to Name ID). The IdP metadata is at `https://<adfs-host>/FederationMetadata/2007-06/FederationMetadata.xml`.
- **Google Workspace.** In the Admin console open Apps, Web and mobile apps, Add app, Add custom SAML app. Download the IdP metadata on the Google Identity Provider details page. On the Service provider details page enter the ACS URL and the Entity ID and choose the Name ID format and value (for example EMAIL and Primary email). Turn the app on for the users in User access.
- **Keycloak.** Create a client of type SAML whose Client ID is the SP Entity ID (or import the SP metadata) and set the ACS URL in the valid redirect URIs or the Assertion Consumer Service POST Binding URL. Keycloak signs the whole document by default: enable *Sign assertions* too, because **WantAssertionsSigned** is True by default. When *Client signature required* is enabled, set **SignAuthnRequests**, **SPCertificate** and **SPPrivateKey**, or disable that option. The IdP metadata is at `https://<host>/realms/<realm>/protocol/saml/descriptor`.

3 · Example: service provider hosted in an HTTP server

A complete service provider with three endpoints: `/saml/metadata` returns the SP metadata, `/saml/login` sends the AuthnRequest and `/saml/acs` validates the response. `CreateUserSession`

and `WriteToLog` stand for your own code. In production remove the pending requests which have not been answered after a few minutes.

```
Delphi (VCL / FireMonkey)
```

```
uses
  SysUtils, Classes, SyncObjs, IdContext, IdCustomHTTPServer,
  sgcWebSocket, sgcWebSocket_Types, sgcAuth_SAML_SP;

type
  TMyApp = class
  private
    FLock: TCriticalSection;
    FPending: TStringList; // RelayState=RequestID of the pending AuthnRequests
    FSAML: TsgcSAMLServiceProvider;
    FServer: TsgcWebSocketHTTPServer;
    function NewRelayState: string;
    function TakeRequestID(const aRelayState: string): string;
    procedure OnCommandGet(AContext: TIdContext;
      ARequestInfo: TIdHTTPRequestInfo; AResponseInfo: TIdHTTPResponseInfo);
    procedure OnSAMLError(Sender: TObject; const aError: string);
  public
    constructor Create;
    destructor Destroy; override;
  end;

constructor TMyApp.Create;
var
  oXML: TStringList;
begin
  FLock := TCriticalSection.Create;
  FPending := TStringList.Create;

  // ... service provider identity, published by GetMetadata
  FSAML := TsgcSAMLServiceProvider.Create(nil);
  FSAML.EntityID := 'https://app.example.com/saml/metadata';
  FSAML.AssertionConsumerServiceURL := 'https://app.example.com/saml/acs';
  FSAML.OpenSSL_Options.APIVersion := oslAPI_3_0;
  FSAML.OnSAMLError := OnSAMLError;

  // ... identity provider: entity ID, SSO URL and signing certificates
  oXML := TStringList.Create;
  try
    oXML.LoadFromFile('idp-metadata.xml');
    FSAML.LoadIdPMetadata(oXML.Text);
  finally
    oXML.Free;
  end;

  // ... https server that hosts /saml/metadata, /saml/login and /saml/acs
  FServer := TsgcWebSocketHTTPServer.Create(nil);
  FServer.Port := 443;
  FServer.SSL := True;
  FServer.SSLOptions.Port := 443;
  FServer.SSLOptions.CertFile := 'app.pem';
  FServer.SSLOptions.KeyFile := 'app.pem';
  FServer.SSLOptions.OpenSSL_Options.APIVersion := oslAPI_3_0;
  FServer.OnCommandGet := OnCommandGet;
  FServer.Active := True;
```

```
end;

destructor TMyApp.Destroy;
begin
  FServer.Active := False;
  FServer.Free;
  FSAML.Free;
  FPending.Free;
  FLock.Free;
  inherited;
end;

function TMyApp.NewRelayState: string;
var
  vGUID: TGUID;
begin
  CreateGUID(vGUID);
  Result := GUIDToString(vGUID);
end;

function TMyApp.TakeRequestID(const aRelayState: string): string;
var
  i: Integer;
begin
  // ... every AuthnRequest can be answered only once
  Result := '';
  FLock.Acquire;
  try
    i := FPending.IndexOfName(aRelayState);
    if (aRelayState <> '') and (i > -1) then
    begin
      Result := FPending.Values[aRelayState];
      FPending.Delete(i);
    end;
  finally
    FLock.Release;
  end;
end;

procedure TMyApp.OnCommandGet(AContext: TIdContext;
  ARequestInfo: TIdHTTPRequestInfo; AResponseInfo: TIdHTTPResponseInfo);
var
  vRelayState, vRequestID: string;
  oResult: TsgcSAMLResult;
begin
  if ARequestInfo.Document = '/saml/metadata' then
  begin
    // ... import this document in the identity provider
    AResponseInfo.ContentType := 'application/xml; charset=utf-8';
    AResponseInfo.ContentText := FSAML.GetMetadata;
  end
  else if ARequestInfo.Document = '/saml/login' then
  begin
    // ... 1. AuthnRequest: redirect the browser to the IdP
    vRelayState := NewRelayState;
```

```
    AResponseInfo.Redirect(FSAML.GetAuthnRequestRedirectURL(vRelayState,
      vRequestID));
    // ... 2. keep the request id to check InResponseTo later
    FLock.Acquire;
    try
      FPending.Add(vRelayState + '=' + vRequestID);
    finally
      FLock.Release;
    end;
  end
  else if (ARequestInfo.Document = '/saml/acs') and
    SameText(ARequestInfo.Command, 'POST') then
  begin
    // ... 3. the IdP posts SAMLResponse and RelayState to the ACS URL
    vRelayState := ARequestInfo.Params.Values['RelayState'];
    vRequestID := TakeRequestID(vRelayState);
    if vRequestID = '' then
    begin
      AResponseInfo.ResponseNo := 400;
      AResponseInfo.ContentText := 'Unknown or expired sign in request.';
      Exit;
    end;
    oResult := TsgcSAMLResult.Create;
    try
      if FSAML.ProcessResponse(ARequestInfo.Params.Values['SAMLResponse'],
        vRelayState, vRequestID, oResult) then
      begin
        // ... 4. the user is authenticated
        CreateUserSession(AResponseInfo, oResult.NameID,
          oResult.Attributes.Values['http://schemas.xmlsoap.org/ws/2005/05/identity/claims/email
          oResult.SessionIndex);
        AResponseInfo.Redirect('/');
      end
      else
      begin
        // ... the reason is in oResult.ErrorMessage, log it, do not show it
        AResponseInfo.ResponseNo := 403;
        AResponseInfo.ContentText := 'Sign in failed.';
      end;
    finally
      oResult.Free;
    end;
  end
  else
    AResponseInfo.ResponseNo := 404;
end;

procedure TMyApp.OnSAMLError(Sender: TObject; const aError: string);
begin
  WriteToLog('SAML: ' + aError);
end;
```

4 · Security checks performed by ProcessResponse

A response is accepted only when every check passes. The first failure stops the validation and its reason is returned in **ErrorMessage**.

- **Safe XML parsing.** The response must be valid Base64 and valid XML. `DOCTYPE` declarations are rejected (no DTD, no external entities) and the size and nesting depth of the document are limited.
- **Status.** The root element must be a SAML 2.0 `samlp:Response` with the status `Success`. Any other status is reported with the status code and message of the IdP.
- **Signature.** The Response or the Assertion must be signed and, with **WantAssertionsSigned** (the default), the Assertion must carry its own signature. Signatures are verified only against the certificates in **IdPCertificates**: a certificate embedded in the message (`KeyInfo`) is never trusted. Supported algorithms are RSA-SHA256, RSA-SHA384 and RSA-SHA512 with exclusive XML canonicalization.
- **SHA-1 disabled by default.** RSA-SHA1 signatures and SHA-1 digests are rejected unless **AllowSHA1** is True.
- **Signature wrapping protection.** The response must contain exactly one Assertion, as a direct child of the Response. The signature must reference the ID of the signed element, that ID must be unique in the document, and after the verification only the element covered by the signature is read.
- **Issuer.** The Issuer of the Assertion, and of the Response when present, must be **IdPEntityID**.
- **Destination and recipient.** The Destination of the Response, when present, and the Recipient of the bearer `SubjectConfirmationData` must be **AssertionConsumerServiceURL**.
- **Audience.** The Assertion must have at least one `AudienceRestriction` and every one of them must contain **EntityID**.
- **Time window.** `NotBefore` and `NotOnOrAfter` of the Conditions and of the bearer confirmation, and `SessionNotOnOrAfter` of the AuthnStatement, are checked against the current UTC time with a tolerance of **ClockSkew** seconds. The bearer confirmation must have a `NotOnOrAfter`. Set **MaxAssertionAge** to also limit the age of the Assertion.
- **InResponseTo.** The Response and the bearer confirmation must answer the request id passed in `aExpectedRequestID`. Unsolicited (IdP-initiated) responses are rejected unless **AllowIdPInitiated** is True, and then they must not contain an InResponseTo.
- **Single assertion.** Responses with several assertions, with an `EncryptedAssertion` or with an encrypted NameID are rejected.
- **Replay cache.** The ID of every accepted Assertion is kept until the Assertion expires. The same Assertion posted a second time is rejected. The cache is thread safe and lives in memory in the component. When several servers share the sign in, descend from the component and override the protected virtual method `DoAddToReplayCache` to keep the IDs in a shared store.

5 · OpenSSL 3

The XML signatures of the IdP are verified with OpenSSL, and OpenSSL signs the AuthnRequests when **SignAuthnRequests** is enabled. **OpenSSL_Options.APIVersion** is `oslAPI_3_0` by default: deploy the OpenSSL 3 libraries (`libcrypto-3.dll` and `libssl-3.dll` on Windows) with the application.

6 · Limitations

- Encrypted assertions (`EncryptedAssertion`) and encrypted identifiers (`EncryptedID`) are not supported. Disable assertion encryption in the IdP.
- Single Logout (SLO) is not implemented. **SessionIndex** is returned so the application can implement its own logout.
- The HTTP-Artifact binding is not supported. The AuthnRequest uses the HTTP-Redirect or HTTP-POST binding and the ACS accepts the HTTP-POST binding.
- Signed AuthnRequests are only available with the HTTP-Redirect binding (RSA-SHA256).
- Only RSA signatures are verified.

Sources used to build this document

Every external claim links back to a primary source. The online help references are the canonical pages the company maintains for this component.

Primary standard / spec: SAML 2.0 Core	docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf
Primary standard / spec: SAML 2.0 Bindings	docs.oasis-open.org/security/saml/v2.0/saml-bindings-2.0-os.pdf
Primary standard / spec: SAML 2.0 Profiles	docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf
Primary standard / spec: SAML 2.0 Metadata	docs.oasis-open.org/security/saml/v2.0/saml-metadata-2.0-os.pdf
Primary standard / spec: XML Signature Syntax and Processing (W3C)	www.w3.org/TR/xmldsig-core1/
Primary standard / spec: Exclusive XML Canonicalization (W3C)	www.w3.org/TR/xml-exc-c14n/
Online help: component page	www.esegece.com/help/sgcWebSockets/ (TsgcSAMLServiceProvider)
Delphi demo project (in the sgcWebSockets package)	`Demos\26.Authentication\03.SAML_ServiceProvider`
Component page	www.esegece.com/products/websockets/http/saml/
sgcAuth pack	www.esegece.com/products/sgcauth/
Product page	www.esegece.com/products/websockets/

> **Document scope.** This document covers the publicly documented surface of the SAML Service Provider component shipped with sgcWebSockets Enterprise and the sgcAuth pack. For the full property, method and event reference consult the online help linked above.