SGCWEBSOCKETS · TECHNICAL DOCUMENT

TOTP Authenticator

TsgcTOTPAuthenticator: time-based and counter-based one-time passwords for two-factor authentication in Delphi and C++ Builder.

eSeGeCe

WWW.ESEGECE.COM

Overview

Add a second factor to any login with the six digit codes of Google Authenticator, Microsoft Authenticator or Authy. Create the secret, show it as a QR code and verify the codes on your server.

The **TsgcTOTPAuthenticator** component implements time-based one-time passwords (**TOTP**, RFC 6238) and HMAC-based one-time passwords (**HOTP**, RFC 4226), the codes shown by authenticator apps such as Google Authenticator, Microsoft Authenticator or Authy. Use it to add a second authentication factor to a login form, a REST API or a WebSocket server.

The component is non-visual and never opens a connection: it creates secrets, builds the `otpauth://` provisioning URI shown as a QR code, generates and verifies codes, and creates one-time recovery codes. Storing the secrets is the job of the application.

At a glance

COMPONENT CLASS `TsgcTOTPAuthenticator`	STANDARDS / SPEC **TOTP, RFC 6238 · HOTP, RFC 4226**
TRANSPORTS **None, codes are computed locally**	PLATFORMS **Windows, macOS, Linux, iOS, Android**
FRAMEWORKS **VCL, FireMonkey**	EDITION **Enterprise (also sgcAuth pack)**

Features

- **Secrets and QR provisioning.** `GenerateSecret` returns a random Base32 secret of `SecretLength` bytes (20 by default). `GetProvisioningURI` builds the `otpauth://totp/` URI with `Issuer`, `Algorithm`, `Digits` and `Period`, ready to render as a QR code.
- **Clock drift tolerance.** `VerifyCode` accepts the current time step and `Window` steps before or after it (1 by default), so a phone whose clock is a few seconds off still signs in.

- **Replay protection.** The `VerifyCode` overload with `aLastTimeStep` only accepts a time step greater than the last one used and returns the matched step, so a code can never be used twice.
- **HOTP counters.** `GenerateHOTP` and `VerifyHOTP` implement the counter-based variant for hardware tokens, with a look-ahead window that resynchronises the counter on success.
- **Recovery codes.** `GenerateRecoveryCodes` fills any `TStrings` with unique one-time codes, the fallback when the user loses the device with the authenticator app.
- **Algorithms and digits.** HMAC-SHA1 (the default every app supports), HMAC-SHA256 or HMAC-SHA512 through `Algorithm`, codes of 6 to 8 `Digits` and any `Period`.

Technical specification

Standards & specs	RFC 6238 · RFC 4226 · RFC 4648 · Key URI Format
Component class	`TsgcTOTPAuthenticator` (unit `sgcAuth_TOTP`)
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.

`Algorithm`	HMAC hash function used to compute the codes: SHA-1, SHA-256 or SHA-512.
`Digits`	Number of digits of every code, from 6 to 8.
`Issuer`	Name of the service shown by the authenticator app next to the account name.
`Period`	Duration in seconds of a TOTP time step.
`SecretLength`	Number of random bytes of the secrets created by GenerateSecret.
`Version`	Read-only string exposing the sgcWebSockets library version.
`Window`	Number of time steps accepted before and after the current one when a TOTP code is verified.

Main methods

The public methods exposed by the component.

`GenerateCode`	Returns the TOTP code of a secret for the current UTC time or for a given Unix time.
`GenerateHOTP`	Returns the RFC 4226 HOTP code of a secret for a given counter.

`GenerateRecoveryCodes`	Clears a list and fills it with unique one-time recovery codes.
`GenerateSecret`	Returns a new random secret, Base32 encoded without padding.
`GetProvisioningURI`	Builds the otpauth://totp/ URI that the authenticator app reads from a QR code.
`VerifyCode`	Verifies a TOTP code, optionally with replay protection based on the last accepted time step.
`VerifyHOTP`	Verifies a HOTP code for a counter and an optional look-ahead range, advancing the counter on success.

Quick Start

Call GenerateSecret once per user, show GetProvisioningURI as a QR code, store the secret, and check every code typed at sign in with VerifyCode.

> **About this scenario.** Enrol the user, then verify the codes. The same code ships in the demo `Demos\26.Authentication\02.TOTP` .

Delphi (VCL / FireMonkey)

```
uses
  sgcAuth_TOTP;

var
  TOTP: TsgcTOTPAuthenticator;
  vSecret, vURI: string;
begin
  TOTP := TsgcTOTPAuthenticator.Create(nil);
  TOTP.Issuer := 'Example App';

  // enrolment: a new Base32 secret and the otpauth:// URI for the QR code
  vSecret := TOTP.GenerateSecret;
  vURI := TOTP.GetProvisioningURI('alice@example.com', vSecret);
  // save vSecret with the user record, render vURI as a QR code

  // sign in: check the code typed by the user
  if TOTP.VerifyCode(vSecret, edtCode.Text) then
    ShowMessage('Code accepted');

  // one-time recovery codes for a lost phone
  TOTP.GenerateRecoveryCodes(memoRecovery.Lines, 10);
end;
```

C++ Builder

```
// uses: sgcAuth_TOTP
TsgcTOTPAuthenticator *TOTP = new TsgcTOTPAuthenticator(this);
TOTP->Issuer = "Example App";

String Secret = TOTP->GenerateSecret();
String URI = TOTP->GetProvisioningURI("alice@example.com", Secret);

if (TOTP->VerifyCode(Secret, edtCode->Text))
  ShowMessage("Code accepted");

TOTP->GenerateRecoveryCodes(memoRecovery->Lines, 10);
```

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 · How TOTP works

1. The server creates a random secret for every user and shares it once with the authenticator app, usually as a QR code.
2. Both sides divide the current Unix time by **Period** (30 seconds) to get the same time step counter.
3. Both sides compute the HMAC (SHA-1 by default) of that counter keyed with the secret.
4. A dynamic truncation turns the HMAC into a decimal code of **Digits** digits (6 by default).
5. The server accepts the code when it matches the current time step or one of the **Window** steps before or after it.

2 · Quick Start: enrolment

Create a secret, show the provisioning URI as a QR code and confirm the enrolment with a first code typed by the user. Save the secret only when the first code is valid.

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

```
oTOTP := TsgcTOTPAuthenticator.Create(nil);
oTOTP.Issuer := 'My Company';

// 1. new secret for the user and the URI rendered as a QR code
vSecret := oTOTP.GenerateSecret;
vURI := oTOTP.GetProvisioningURI('john@example.com', vSecret);
ShowQRCode(vURI); // any QR code library

// 2. the user scans the QR code and types the first code
vLastStep := -1;
vNow := StrToInt64(GetDateTimeUnix(Now, False)); // UTC Unix time, sgcBase_Helpers
if oTOTP.VerifyCode(vSecret, vCode, vNow, vLastStep) then
begin
  SaveUserSecret('john@example.com', Encrypt(vSecret), vLastStep);
  // 3. recovery codes: show them once, store only their hashes
  oCodes := TStringList.Create;
  try
    oTOTP.GenerateRecoveryCodes(oCodes, 10);
    ShowRecoveryCodes(oCodes);
    SaveRecoveryCodeHashes('john@example.com', oCodes);
  finally
    oCodes.Free;
  end;
end;
```

3 · Quick Start: verification with replay protection

A TOTP code stays valid for the whole time step, and with **Window** = 1 for the steps before and after it. Store the last accepted time step of every user and pass it to **VerifyCode**: only a code for a later step is accepted, so a code which has already been used, or which was captured by an attacker, can not be used again.

Delphi (VCL / FireMonkey)

```
function TMyServer.CheckSecondFactor(const aUser, aCode: string): Boolean;
var
  vSecret: string;
  vLastStep, vNow: Int64;
begin
  LoadUserSecret(aUser, vSecret, vLastStep); // vSecret decrypted, vLastStep = -1 the first time
  vNow := StrToInt64(GetDateTimeUnix(Now, False));
  Result := oTOTP.VerifyCode(vSecret, Trim(aCode), vNow, vLastStep);
  if Result then
    SaveLastTimeStep(aUser, vLastStep); // updated with the matched step
end;
```

4 · Security notes

- **Store the secrets encrypted.** The TOTP secret works like a password shared between the server and the app. Anyone who reads it can generate valid codes, so keep it encrypted at rest and never log it.
- **Store the recovery codes hashed. GenerateRecoveryCodes** only creates the codes. Show them to the user once, save a hash of every code (for example SHA-256), compare the hash when a code is used and delete it after a successful login, because every recovery code is valid only once.
- **Persist the last time step.** Replay protection only works when the value returned in `aLastTimeStep` is saved for the user after every successful verification.
- **Limit the attempts.** A 6 digit code has one million values, lock the account or add a delay after a few failed codes.
- **Delphi 7.** On Delphi 7 HMAC-SHA256 and HMAC-SHA512 are computed by OpenSSL. When **Algorithm** is `totpSHA256` or `totpSHA512`, deploy the OpenSSL 3 libraries with the application and keep OpenSSL 3 selected (`OPENSSL_API_VERSION := opSSL_3_0`, the default value in `sgcIdSSLOpenSSLHeaders`).

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: RFC 6238	datatracker.ietf.org/doc/html/rfc6238
Primary standard / spec: RFC 4226	datatracker.ietf.org/doc/html/rfc4226
Primary standard / spec: RFC 4648	datatracker.ietf.org/doc/html/rfc4648
Primary standard / spec: Key URI Format	github.com/google/google-authenticator/wiki/Key-Uri-Format
Online help: component page	www.esegece.com/help/sgcWebSockets/ (TsgcTOTPAuthenticator)
Delphi demo project (in the sgcWebSockets package)	`Demos\26.Authentication\02.TOTP`
Component page	www.esegece.com/products/websockets/http/totp/
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 TOTP Authenticator component shipped with sgcWebSockets Enterprise and the sgcAuth pack. For the full property, method and event reference consult the online help linked above.