Security Protocols homework

Protopass – Online, browser-based, end-to-end encrypted password manager
source code documentation

Ádám BERECZKI (XKTB7Q)
Zoltán BOGÁROMI (A3H4H8)
Tamás Soma LUCZ (HQC99F)

Last revision: 4 May, 2018

Projects

The application consists of two separate deployment units: the client-side project (protopass-frontend deployed to https://protopass-frontend.azurewebsites.net), and the server-side project (protopass-backend deployed to https://protopass-backend.azurewebsites.net): they communicate via a REST API exposed by the server. We describe these projects separately in this document.

The client

Dependencies

The client-side project is an Angular 5-based frontend, written in TypeScript. Most of the dependencies belong to these two technologies and are generated by the Angular CLI. We used, however, some cryptographic packages during development via the Node Package Manager (npm):

- secure-remote-password by LinusU (https://github.com/LinusU/secure-remote-password) for using the SRP protocol for the login process,
- scrypt-async by dchest (https://github.com/dchest/scrypt-async-js) for key derivation for the cryptography of the user profile.

Both packages have TypeScript typing extensions, which is convenient when using TypeScript.

Angular components

Except the Navbar component (which is shown on all the views if the user is logged in), all components have their own routes (described in app-routing.module.ts). The Dashboard and ChangeLoginPassword components are only accessible for logged-in users, their routes are protected by the Auth route guard. (According to the Auth guard, a user is considered to be logged in if their browser's SessionStorage contains an element with the key "sessionId". This is acquired from the server at login: after authentication, it is stored to the SessionStorage. If the user closes the page or the browser, it is deleted, meaning the user has to log in again. This is an intended security measure.)

Navbar

This component is located on top of the pages, if the user is logged in. There are links to the Dashboard (where the password storage feature is), the change login password feature, and logout.

Register

Available under route /register, this component implements user registration. A form field is shown to the user to give email address and a password (with confirmation) for registration. Valid email address format is checked (appValidateEmailOnInput directive), and the password must match the criteria of a strong password (appValidatePasswordStrengthOnInput directive). Then, registration process is handled by SessionService (described later).

Validate

Validate component has no view and is only responsible for validating email addresses via /validate api endpoint. After registration, an email is sent to the email address the user has given during registration: this email contains a link to the Validate component with an id to verify the address. This Validate component is only responsible for calling the /validate endpoint of the backend with the suitable data. After doing this, it immediately redirects to /login with either a success or an error message.

Login

User credentials are asked to proceed with login. Valid email address and password are required. The login process is handled by the SessionService.

ChangeLoginPassword

This component's route is protected, only users logged in can use its functions. Users can change their login password (for the account). Strong password criteria are checked as by registration. Uses SessionService.

ForgotPassword

If a user forgets the password for the account, an email is sent to the registered email address with a link to verify the user. In this component, the user enters their email address where the email is sent. Uses SessionService.

ResetPassword

This component is responsible for resetting the user's password. It checks if the id in the query parameter is valid (this id is in the email sent by ForgotPassword). If it is, again, strong password criteria are checked to proceed with password reset. Uses SessionService.

Dashboard

This component is the main view of the application, under route /dashboard. All password storage feature is exposed by this component. Only logged in users can access it. For a reminder, the user's profile is a container which is encrypted/decrypted by the client, that contains the user's passwords. This view consists of multiple panels.

If the profile password is not yet entered since the user is logged in, the user is asked to enter their profile password so their user profile can be decrypted afterwards.

If the user has not set up the profile yet, a password is asked for initializing the profile: an empty profile gets created and uploaded to the server. In this case, a password and a confirmation are required, and then the (empty) profile is created. The user is warned that this password cannot be recovered if forgotten.

If the profile already exists, the profile is decrypted with this password (or if the provided password is wrong, an error message is shown).

After the profile password is set up or entered by the user, the Dashboard is shown, with stored password entries and the ability to add new ones. At the initialization of the Dashboard, only the associated data gets displayed, the passwords themselves do not (also, they are not stored in-memory). If the user wants to display a password, they need to click on the Show password button of the wanted password entry. In this case the profile is downloaded, it is opened with the container password (stored by the ContainerPasswordStorageService), and the wanted password entry's value (the password) gets copied to the GUI variable. This prevents that all passwords would sit on the memory unencrypted.

The application also helps to generate secure passwords. Certain conditions can be chosen such as length, whether the password should contain letters, capital letters, numbers or symbols.

The user profile password can be changed on this page, too. This involves downloading the profile, decrypting it with the old container password, encrypting with the new container password, and uploading the new profile encrypted with the new password.

Services

These services contain the application's main business logic, methods of these are called from the components.

ApiService

This service is responsible for communicating with the server API. The **baseUrl** final variable contains the address of the server API. The **call** function wraps HTTPS operations, HTTPS requests are made here and a promise with the server response is returned. If the error code indicates the request was not successful (the response status is not 200-299), an exception is thrown with the returned server error code (e.g. "UserNotExists").

Public methods of the service are responsible for sending requests to the API endpoints of the server (via the call function). These are the following:

- register
- validate
- challenge
- authenticate
- logout
- downloadUserProfile
- uploadUserProfile
- getStorageKey
- changePassword
- resetPasswordRequest
- resetPassword

ContainerPasswordStorageService

This service is responsible for managing the secure storage of the user's profile password.

Without this, the user would have to enter their profile password for every operation on their user profile. This service stores (but does NOT persist) the user's profile password locally, in-memory encrypted with AES-GCM (with a key provided by the server), and decrypts it when it is needed.

The per-user unique password storage key is stored on the server: it is downloaded through getStorageKey endpoint with a valid session. The storage key is invalidated and regenerated every 10 minutes by the server, and cleared on logout and on login as well. The key for decrypting the container password stored in the service is derived from this server key with scrypt. If the user enters their profile password after login, or changes their password, the service encrypts and stores it.

CryptoService
This utility service provides easily accessible cryptographic methods, wrapping the imported scrypt library and the Web Crypto API. AES-GCM is used with a 16-byte authentication tag. Also, it has other functions including generating cryptographically secure random bytes, and random alphabetic character generation from a provided set of characters.

SessionService
SessionService handles user authentication tasks, such as login, logout, changing, resetting user login password. It relies on the SRP protocol, which is utilized via the secure-remote-password library. At the registration, salt and SRP verifier are generated. The authentication at login happens in a challenge-response manner according to the SRP-6a specification, meaning the password does not leave the client machine in any form. Changing and resetting password is similar to registration, with the latter using a verification id sent in email.

UserProfileService
This service is responsible for manipulating the user's profile.

It does not store the profile in any way. Every operation on the profile involves a *download-decrypt-modify-encrypt-upload* cycle. During this cycle, the profile is locked, meaning no other manipulations can be started on it. The necessary profile password for encryption/decryption is provided by the ContainerPasswordStorageService described above: the profile password is entered by the user if needed. The encryption salt and IV are generated locally. Fresh 32-bit salt and IV are generated on every cycle, the salt and IV are uploaded to the server along with the encrypted profile.

It relies on other services such as CryptoService, ContainerPasswordStorageService, ApiService and Convert utility service. After download and before upload, the profile, salt and IV are converted via Convert to/from base64 from/to byte array representation. Byte arrays are then encrypted and decoded using CryptoService AES functions. Password for the profile is accessed via ContainerPasswordStorageService. This service handles functions of the dashboard: profile initialization, displaying stored passwords, adding, removing passwords, and changing password for the profile itself.

The service is stateful in the meaning of being locked or not. The profile is considered to be locked if there is an in-progress manipulation cycle on it. If the service is locked, no other manipulation can happen on the profile, it then rejects all function call. This feature is provided by the mutexCall function, wrapping a private locked variable into a try...finally block.

UtilsService
Methods of this service check if the passwords given by the user meet certain conditions: if they are longer than 12 characters, contain more than one letter, capital letter, numbers, and symbols; and if the format of the email address fields is correct. Regular expressions are used to implement these conditions. These functions are used whenever an email or a password is asked.

Utils
There are two helper services implemented: Convert is responsible for transformation between certain formats such as base64, bytes, hexadecimal, ASCII, which is required for the communication with the server. Utf8 handles encoding UTF-8 strings to byte arrays and vica versa.

Server side

The server side of the application is written in Python, using the Django framework, version 1.1. The pysrp package is used for implementing the SRP-based login process. Server-side functions are separated into two main parts, as the folder structure shows: profile_handler and protopass_auth. These two parts are described int the following chapters.

Profile handler

This unit is responsible for managing the user profile and the encryption key for the container password. It serves the following endpoints as shown in urls.py:

- uploadUserProfile
- downloadUserProfile
- getStorageKey

These functions are implemented in views/profile_handler.py.

Upload user profile

UploadProfileView implements storing the users profile: it reads salt, iv, and the encrypted profile from the request, checks, if salt and iv are fresh (not the same as the stored ones). If no profile exists, it creates a new, and stores data sent by client.

Download user profile

DownloadProfileView serves the user profile in the response body, and if no profile is found, it sends a HTTP 404 response.

Get storage key

DownloadStorageKeyView responds with the stored key for the container password. If it does not exists, or a forceFresh parameter is in the request, a random string is generated and sent in response.

Protopass authentication

This unit handles user authentication related tasks, such as login, registration, password setting, and is responsidle to serve the following API endpoints:

- register
- validate
- challenge
- authenticate
- logout
- changePassword
- resetPasswordRequest
- resetPassword

This unit uses the pysrp library and similar character conversion tasks as described at the client. All endpoints are implemented in authentication.py, except for email validation (email_validator.py).

The SRP's private key is derived using raw SHA-256. We tried to introduce an scrypt-based key derivation for the SRP as well, but we found no libraries supporting it.

Register

RegisterView handles user registration. After error checks, it creates a not yet activated user, authentication information is stored separately in an AuthenticationProfile. Then it sends an activation email to the given email address, which contains the link with activation_id for activating the profile.

Validate

EmailValidatorView checks, if a not yet activated user is providing the correct activation_id via the link sent in email. If it is correct, the profile is activated, and the user can login and use the application.

Challenge

ChallengeView implements the server side of the SRP challenge, which is necessary for user authentication. It receives the clientChallenge and sends the serverChallenge with the salt provided at the registration.

Authenticate

AuthenticateView performs verification of user's data and the challenge information. It receives the clientProof, and calculates the SRP session key on the server side. If successful, a session token is generated and sent to the client. We did not use the SRP's session key as the session ID, but a unique, server-side generated key is used for this.

Logout

LogoutView logs out the user by deleting the session token and container password storage key.

ChangePassword

ChangePasswordView stores the new password for the user sent in the request.

ResetPasswordRequest

ResetPasswordView GET request handles this endpoint. If an existing user has this email address, a reset_id is sent to the user's email address, to verify the user before password reset.

ResetPassword

ResetPasswordView POST request checks, if the reset_id is correct. If it is, it sets the new password salt, and verifier for the user and saves them.