



BRNO UNIVERSITY OF TECHNOLOGY

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

EXCALIBUR SYSTEM - SSO IMPLEMENTATION

SYSTÉM EXCALIBUR - IMPLEMENTACE SSO

MASTER'S THESIS

DIPLOMOVÁ PRÁCE

AUTHOR

AUTOR PRÁCE

Bc. JURAJ CHRIPKO

SUPERVISOR

VEDOUČÍ PRÁCE

Mgr. KAMIL MALINKA, Ph.D.

BRNO 2021

Master's Thesis Specification



Student: **Chripko Juraj, Bc.**
Programme: Information Technology Field of study: Information Technology Security
Title: **Excalibur System - SSO Implementation**
Category: Security

Assignment:

1. Get familiar with Single Sign On (SSO) technologies, focus on Security Assertion Markup Language (SAML) and Fast Identity Online 2 (FIDO2).
2. Get familiar with Excalibur (commercial distributed infrastructure access control system) and other existing access control solutions and used concepts.
3. Design a solution for integrating SAML or FIDO2 into Excalibur system to enable SSO.
4. Implement proposed design for integrating SAML or FIDO2 protocols into Excalibur system.
5. Test correct functionality of implemented solution such as correct behavior from a user point of view, soundness of access control mechanism, and access revocation.

Recommended literature:

- FIDO Alliance (available online <https://fidoalliance.org/fido2/>)
- Security Assertion Markup Language (SAML) V2.0 Technical Overview (available online <http://docs.oasis-open.org/security/saml/Post2.0/sssc-saml-tech-overview-2.0.html>)
- "Excalibur - No More Passwords!" (available online https://getexcalibur.com/docs/Excalibur_pitch_2_4_2018.pdf)

Requirements for the semestral defence:

- Items 1 to 3.

Detailed formal requirements can be found at <https://www.fit.vut.cz/study/theses/>

Supervisor: **Malinka Kamil, Mgr., Ph.D.**
Head of Department: Hanáček Petr, doc. Dr. Ing.
Beginning of work: November 1, 2020
Submission deadline: May 19, 2021
Approval date: November 11, 2020

Abstract

The ultimate goal of the Excalibur system is to move all authentication away from passwords, to the passwordless future. The aim of this thesis is the integration of the Excalibur system with web-based, password-free protocols SAML and FIDO.

SAML standard was integrated into the Excalibur system and successfully tested on multiple major applications. Excalibur is responsible for authentication and user management, and SAML is used to transfer authentication data to third-party applications.

FIDO, on the other hand, is a complete authentication standard, which can be integrated into the Excalibur system in several ways. The most promising way seems to be replacing the Excalibur authentication mechanism with FIDO2, but weak standard support and missing features do not allow it, for now.

Abstrakt

Cieľom systému Excalibur je presunúť autentifikáciu od hesiel používaných v súčasnosti ku bezheslovej budúcnosti. Zámerom tejto práce je integrácia systému Excalibur s webovými bezheslovými protokolmi SAML a FIDO2.

Štandard SAML bol integrovaný do systému Excalibur a úspešne otestovaný s niekoľkými známymi aplikáciami. Excalibur má na starosti samotnú autentifikáciu a manažment používateľov a SAML je použitý na predanie týchto informácií aplikáciám tretích strán.

FIDO2 je, na druhú stranu, kompletný autentifikačný štandard, ktorý môže byť do systému Excalibur integrovaný viacerými spôsobmi. Ako najslubnejší spôsob sa javí výmena autentifikačného mechanizmu systému Excalibur za FIDO2, ale slabá podpora štandardu a chýbajúce funkcie to zatiaľ nedovoľujú.

Keywords

SSO, single sign-on, single sign on, SAML, FIDO, FIDO2, WebAuthn, Excalibur, distributed crypto scheme

Klíčová slova

SSO, single sign-on, single sign on, SAML, FIDO, FIDO2, WebAuthn, Excalibur, distribuovaná kryptoschéma

Reference

CHRIPKO, Juraj. *Excalibur System - SSO Implementation*. Brno, 2021. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Mgr. Kamil Malinka, Ph.D.

Rozšířený abstrakt

Heslá sú momentálne najpoužívanější spôsob autentifikácie na internete, a zároveň sú aj najčastejším cieľom rôznych kybernetických útokov (phishing, hádanie hesiel, atď.). Cieľom práce bola integrácia systému Excalibur s webovými bezheslovými autentifikačnými protokolmi FIDO2 a SAML.

Excalibur slúži ako bezpečnostný token pre autentifikáciu bez hesiel. Používa mobilný telefón na overenie autentifikačných faktorov ako je poloha, PIN kód, odtlačok prsta, Face ID atď. Používateľ je overený pomocou biometrie na svojom vlastnom telefóne, čiže prihlasovanie sa pre neho stáva bezheslové. Avšak aplikácie stále využívajú heslá, preto bola vyvinutá distribuovaná kryptoschéma, ktorá tvorí abstrakciu nad heslami. Heslo je zrekonštruované až v cieľovom systéme a po prihlásení môže byť automaticky zmenené, čo eliminuje množstvo útokov na heslá. V praxi to znamená, že používateľ nepozná aktuálne heslo a prihlásenie je možné iba jeho telefónom po overení všetkých faktorov ako biometria, poloha a pod. Používateľ teda nemôže vyzradiť heslo, ani delegovať prístup inému používateľovi. Distribuovaná kryptoschéma je použitá pri prihlasovaní do operačného systému Windows, no webové technológie ponúkajú niekoľko možností ako zabezpečiť pravú bezheslovú autentifikáciu. Táto práca sa bližšie venuje webovým štandardom FIDO2 a SAML.

FIDO2 umožňuje používateľom využívať bežné zariadenia na ľahkú autentifikáciu pre online služby v mobilných aj desktopových prostrediach. FIDO Alliance stojí aj za starším štandardom Universal 2nd Factor (U2F), ktorý slúži ako 2. faktor pre autentifikáciu na internete. FIDO2 v sebe zahŕňa aj spätnú kompatibilitu so štandardom U2F, keďže U2F je priamy predchodca štandardu FIDO2.

FIDO2 je kompletný autentifikačný štandard, ktorý dokáže nahradiť prihlasovanie pomocou mena a hesla, no môže byť použitý aj na iné účely. Je založený, podobne ako Excalibur, na kryptografii s využitím verejného kľúča, čiže neexistuje žiadne verejné tajomstvo, ktoré by mohol útočník ukradnúť. Štandard FIDO2 je v tejto práci podrobnejšie popísaný v teoretickej časti a výsledkom sú 4 možnosti integrácie so systémom Excalibur: Excalibur vystupujúci ako FIDO2 autentifikátor voči aplikácii, Excalibur vystupujúci ako druhý faktor (U2F), FIDO2 zariadenie ako ďalší faktor voči systému Excalibur, FIDO2 namiesto autentifikačného mechanizmu systému Excalibur.

Všetky tieto možnosti integrácie FIDO2 štandardu so systémom Excalibur majú svoje špeciálne prípady, ktoré sú podrobnejšie popísané v tejto práci. Vzhľadom na to, že zatiaľ nebolo nájdené vhodné využitie, štandard FIDO2 nebol integrovaný so systémom Excalibur. Hlavné dôvody prečo FIDO2 nebol integrovaný je aj slabá podpora FIDO2 štandardu koncovými aplikáciami, slabá podpora rozšírení štandardu FIDO2 prehliadačmi, či fakt, že schéma systému Excalibur dokáže to isté ako FIDO2, ak nie viac.

SAML (Security Assertion Markup Language) je autentifikačný protokol slúžiaci na výmenu autentifikačných údajov medzi poskytovateľom identít – Identity Provider (IDP) a poskytovateľom služieb – Service Provider (SP). Protokol SAML umožňuje presne to, čo bolo zmyslom tejto práce – použiť viacfaktorovú autentifikáciu bez hesla k webovým aplikáciám tretích strán a preto sa zvyšok tejto práce venuje návrhu implementácie, samotnej implementácii a testovaniu integrácie. Excalibur zabezpečuje prihlasovanie, čiže vystupuje ako IDP a autentifikačné údaje posiela aplikáciám, kde sú použité na prihlásenie používateľa. Predtým ako spolu začne IDP a SP komunikovať, musia si dôverovať, čo je zabezpečené výmenou SAML metadát. Tieto metadáta obsahujú informácie o entitách spolu s ich certifikátmi.

Implementáciu SAML časti je možné rozdeliť na implementáciu autentifikačného komponentu a implementáciu samotného SAML komponentu. Pre samotnú autentifikáciu bol použitý komponent `WebSDK`, ktorý nebol navrhnutý na tento účel a musel byť preto upravený. V budúcnosti bude tento komponent vymenený za nový, no momentálne to nie je možné vzhľadom na stav vývoja novej verzie systému Excalibur. Pre prácu so SAML správami bola použitá knižnica `samlify`, no pri testovaní sa zistilo, že jej chýbajú niektoré funkcie ako napríklad možnosť použiť šablónu pre SAML dokumenty a tiež bolo zistené, že obsahuje trhliny v bezpečnosti, ktorých oprava trvá neprimerane dlho. SAML komponent má na starosti aj manažment poskytovateľov služieb (SP). SP sú manažované pomocou ich metadát, preto boli vytvorené prvky na pridanie, zmenu a zmazanie SP v administrátorskom rozhraní – Excalibur Dashboard.

Momentálne je vyvíjaná nová verzia systému Excalibur (v3.5), ktorá používa novú architektúru a tým ponúka aj nové možnosti. Stará verzia (v3) nedokáže zabezpečiť single sign-on (SSO) a zároveň, automatické testovanie je veľmi obmedzené. Súčasný riešenie bolo implementované v starej verzii systému Excalibur, ktorá nedokáže zabezpečiť SSO, no jeho návrh je súčasťou tejto práce. Návrh nového autentifikačného komponentu je takisto súčasťou práce a bude implementovaný hneď, ako to stav novej verzie systému dovoľí.

Výstupom tejto práce je SAML komponent schopný prihlásiť používateľov do známych aplikácií ako Office 365, Google, Ping Identity, Pulse Secure a pod. Toto riešenie je nasadené u partnerov systému Excalibur a je testované na väčšom množstve používateľov. Počas testovania riešenia s rôznymi aplikáciami vznikla aj dokumentácia a návod na konfiguráciu prihlasovania cez protokol SAML do týchto aplikácií, ktorá je verejne dostupná na stránke Excalibur dokumentácie.

Práce na SAML komponente budú pokračovať, pretože sú plánované ďalšie nasadenia systému Excalibur, kde bude využitý aj SAML. Pre novú verziu systému bude potrebné prepísať časť hotového riešenia a takisto otestovať iné SAML knižnice, keďže zvolená knižnica nie je ideálna, ako už bolo spomenuté.

Excalibur System - SSO Implementation

Declaration

I hereby declare that this Master's thesis was prepared as an original work by the author under the supervision of Mgr. Kamil Malinka, Ph.D. The supplementary information was provided by Ing. Ivan Klimek, Ph.D. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Juraj Chripko
May 18, 2021

Acknowledgements

I would like to thank Mr. Kamil Malinka for his thorough guidance and his hints, and Mr. Ivan Klimek for his valuable advice.

Contents

1	Introduction	2
2	Motivation	4
2.1	Password problems	4
2.2	Improving passwords	5
2.3	Password Alternative	6
2.4	Excalibur	7
2.5	SAML	9
2.6	FIDO2	13
3	Excalibur	20
3.1	Excalibur Components	20
3.2	Excalibur PAM	25
3.3	Excalibur system actions	28
4	Integration design	33
4.1	Excalibur – SAML integration design	33
4.2	FIDO2 Case Study	37
4.3	Single sign-on (SSO) design	42
5	Implementation	44
5.1	SAML Server Component	44
5.2	Authentication Component	51
5.3	Single sign-on (SSO)	52
6	Testing and Documentation	53
6.1	Functional Testing	53
6.2	Documentation	57
7	Conclusion	59
	Bibliography	61
	Appendices	66
A	CD Contents	67

Chapter 1

Introduction

Using passwords was for a long time the only way of authentication on the web. Today, a lot of other authentication mechanisms can be used, but passwords stay as the first and main factor. With dozens of both personal and work accounts, there is a serial problem of managing passwords. The average business employee must keep track of 191 passwords and 81% of data breaches were caused by password-based attacks [1].

Password managers can help with remembering passwords and not reusing passwords for multiple accounts, but they can not stop phishing attacks, data breaches nor vulnerability exploitation. A password manager can securely store passwords but in moment of authentication, that password needs to be injected into the website, which means it can, in theory, be stolen by a man-in-the-middle. Industry consensus is to use multiple authentication methods (factors) whenever possible, but most of the second factors can be phished as easily as a password. So passwords are hard to remember, hard to manage, and even dangerous. But our existence is built on top of passwords anyway.

Excalibur acts as a secure token for passwordless authentication using your mobile phone to verify authentication factors such as location, PIN, fingerprint, Face ID, etc. In practice, authentication for the genuine person, authentication is seamless, but it is impossible for any other person. Excalibur created distributed crypto scheme to keep passwords secure. Passwords are bound to the mobile device and protected by biometry, so the mobile device is your Excalibur. Since most systems require passwords, Excalibur creates a new layer on top of existing infrastructure, but the ultimate goal is to a create password-free future.

2020 was an unprecedented year in regards to cybersecurity. Mass remote working has radically changed both how in which people connect and interface with their workplaces as well as how businesses work. With “everyone” working from home, in 2020 more than ever before, attackers have shifted their focus to the outdated and weakly protected remote worker’s devices and techniques such as phishing attacks, ransomware, and data theft have exploded.

Excalibur saw a transition to web solutions in advance, which led to the development of the Excalibur PAM (Privileged Access Management). Excalibur PAM can provide controlled access to most applications by supporting a wide range of protocols. All these applications can be accessed by any web browser. For now, Excalibur was only using abstraction to deliver passwords to the client application, where they were used for authentication. But with the rising use of web applications, passwordless protocols could be used for authentication to third-party applications.

One of the protocols which support passwordless authentication is SAML. This protocol is widely used in the enterprise sector for transferring authentication information between

identity providers and service providers. This thesis aims at designing and implementing SAML integration to the Excalibur system. The distributed crypto scheme is developed in such a way, that when one component is compromised, an attacker gains no access to any resource. By using passwords, this is ensured since part of the password is stored on the user's device and it is secured by biometry. This idea should also be extended to SAML targets.

Another standard for passwordless authentication is FIDO2. FIDO2 enables users to leverage common devices to easily authenticate to online services in both mobile and desktop environments. FIDO2 was chosen because it is built with a similar idea in mind as Excalibur. FIDO2 is a whole authentication standard, so how it can be integrated and what can be gained from that concrete type of integration is also part of this thesis.

The first chapter is an introduction to authentication, mainly on the authentication on the internet and how the most used authentication mechanism (passwords) works in regards to Single sign-on (SSO). Description of how Excalibur tackles security problems of web-based solutions follows. Another 2 subsections are about chosen passwordless protocols: SAML and FIDO2.

The next chapter describes the Excalibur system in more depth. Especially Excalibur components, PAM, actions, the current state of the Excalibur system, and how the Excalibur distributed crypto scheme works. Excalibur PAM is a common name for multiple technologies, so their specifics are also discussed in this chapter. Sensitive information like certificates are still used, so their usage and storage are also described in this chapter.

Fourth chapter starts by combining facts, algorithms, and authentication mechanisms from the previous chapter into a solution that integrates SAML protocol to the Excalibur system. The chapter continues with a description of the FIDO2 use cases since there are multiple ways, how FIDO2 standard can be integrated into the Excalibur system. Integration design also contains basic design decisions for various single sign-on scenarios.

The implementation chapter is focused on SAML implementation, since none of the FIDO2 use cases were good enough to be implemented for now. Single sign-on was also not implemented in this project, because of ongoing efforts to develop a new major version of the Excalibur system. SAML implementation can be divided into 2 components: SAML Component, taking care of all SAML communication, and an Authentication Component, which responsibility is to authenticate a user to the Excalibur system. The implementation chapter also includes a single sign-on section that describes how authentication events from various Excalibur clients can be used for single sign-on functionality once the new version will be released.

Another chapter describes testing and documentation of previously designed and implemented SAML integration. Testing revealed some bugs and security vulnerabilities in the SAML library and in a way how that library is used. During testing and configuration of various service providers documentation was written.

Chapter 2

Motivation

In 2020, the use of video conferencing skyrocketed because of the COVID-19 pandemic and the resulting lockdown. Video conferencing statistics and studies on remote work in 2019 show that the global remote workforce has increased by 140% since 2005 [7]. As a consequence, technology has become even more important in both our working and personal lives. The remote workforce continues to grow and so does the need for cybersecurity. Swiss National Cyber Security Centre (NCSC), stated that the number of cyberattacks reported in Switzerland during the height of the COVID-19 pandemic was up to three times higher than normal [54]. 80% of hacking-related breaches leverage compromised passwords (phishing, brute-forcing, keylogger attacks, credential stuffing, etc.).

This chapter includes basic information about the most used authentication mechanism – passwords. How are they used, what are the password problems and what could be an alternative. This chapter also introduces Excalibur system and 2 passwordless protocols: SAML and FIDO2.

2.1 Password problems

Let's take a closer look at the most used authentication mechanism, **passwords**. Password is a shared secret, a user knows the password and so does the other party, a server. To increase security, passwords are not stored in clear text, but rather hashed. The hash function is a one-way function, a function that is practically infeasible to invert [18]. The hashed password is still a shared secret, since a user needs to give it away, to prove that he knows it.

This creates 2 security problems: data breach problem and fraud problem. **Fraud problem** is oftentimes exploited by a **phishing attack**. Phishing is an example of social engineering techniques used to deceive users. It is the fraudulent attempt to obtain sensitive information or data, such as usernames, passwords, and credit card details, by disguising oneself as a trustworthy website [51]. 51 % of users have experienced phishing attack, 12 % credential theft, and 8 % man-in-the-middle attack based on a survey Ponemon Institute Research Report from 2020 [44].

Data breach exposes confidential information, like credentials, to an unauthorized person, which can lead to another cyberattack, called credential stuffing, by using previously exposed credentials. As long as there is a shared secret stored on the server, there will be something to steal. Verizon showed that hacking attacks (and breaches in general)

are mostly credential theft driven as 80% of them involve brute force or the use of lost credentials [1].

Passwords also suffer from the **usability problem**. A user needs to remember dozens of passwords for both personal and enterprise accounts and the easiest way is just to reuse the same passwords. A report from 2020 sponsored by Yubico showed that 50 % of IT security respondents and 54 % of individual users reuse their passwords. In both categories, respondents reuse passwords on an average of 10 accounts [44].

These problems are caused by password's inherited properties, we cannot use passwords without these inhering these problems.

Password Managers can help manage passwords, but they don't protect from stolen passwords or phishing attacks. Traditionally passwords are stored hashed since the server only needs to compare hashes, but password managers need to store passwords in such a way, that the password can be reconstructed. What makes them password vaults, and that can be attacked [2]. Password manager's ability to protect the user from a phishing attack is based on the ability to detect a fake website. Researches at the University of York fooled 40% of password managers into giving away passwords to malicious apps [13].

2.2 Improving passwords

There are 3 authentication factors categories [41]:

- **Knowledge** – something the user knows (passwords)
- **Possession** – something the user has (tokens – smart card, usb token, software, certificates, etc.)
- **Inherence** – Something the user is or does (biometrics - fingerprint, face recognition, signature, etc.)

When only one of these mechanisms is used it is **Single-factor authentication**. On the other hand, **Multi-factor authentication (MFA)** combines two or more factors.

A fairly new type of authentication is **contextual authentication**. A user interacts with the system in a specific way. Every user has a slightly different style of using the system, how they type, how fast and how much they move mouse, where do the login from (IP address), when do they log in. These factors create context [41].

Using the second factor (SMS, email, other types of One-Time-Passwords (OTP), etc.) eliminates the threat created by stolen passwords and passwords reuse for multiple applications, but only for the application with the configured second factor. An attacker can still create a fake website where the user enters passwords together with the second factor, e.g. one-time password. Users have a hard time identifying the fake website. The best defense against these password attacks is user education about password usage and management, but how that's going is shown in the comics from popular xkcd series (936) [30].

The second factor can prevent phishing, but it cannot be based on a shared secret. All shared secret-based authentication mechanisms (e.g. OTP) can be phished as easily as the passwords. This thesis describes other types of authentication mechanisms that are phishing resistant, they are typically based on public-key cryptography.

While the second factor can improve security, adding more steps to the authentication flow is just making the usability problem worse. A solution could be replacing passwords with a better authentication mechanism.

2.3 Password Alternative

A reasonable requirement is to replace passwords with something hard to guess and simultaneously hard to steal. A long, complex password can be hard to guess, but it still needs to be saved on the target system and in some way send through the network. **Public key infrastructure (PKI)** could be the answer [17].

Public-key cryptography

Public-key cryptography, or **asymmetric cryptography**, is a cryptographic system that uses pairs of keys: public keys, which may be disseminated widely, and private keys, which are known only to the owner [50]. Public keys are stored on target service and since they are public, disclosing them does not harm system security. Public key cryptography best-known uses are:

- **Public key encryption**, in which a message is encrypted with a recipient's public key. The message cannot be decrypted by anyone who does not possess the matching private key, who is thus presumed to be the owner of that key and the person associated with the public key. This is used in an attempt to ensure confidentiality. The scheme is shown in Figure 2.2.
- **Digital signature**, in which a message is signed with the sender's private key and can be verified by anyone who has access to the sender's public key. This verification proves that the sender had access to the private key, and therefore is likely to be the person associated with the public key. This also ensures that the message has not been tampered with, as a signature is mathematically bound to the message it originally was made with, and verification will fail for practically any other message, no matter how similar to the original message. Figure 2.1 shows scheme digital signature. A claim that the user is in possession of a private key can be for authentication. This authentication mechanism is **claim based** because only a claim is sent through the network, not the actual password/private key.

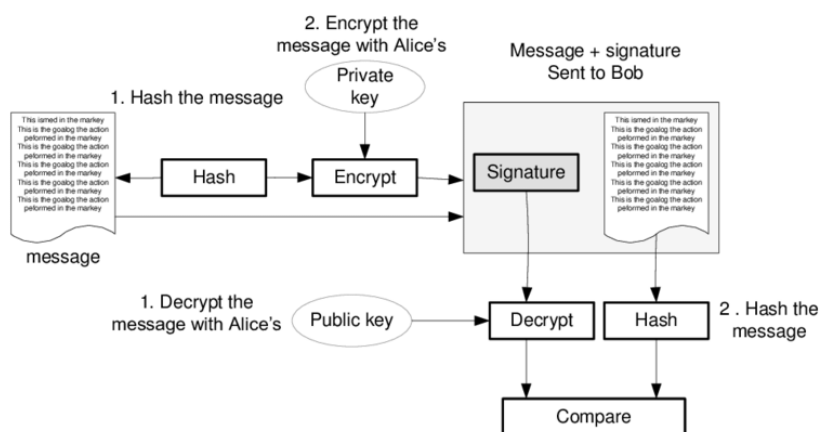


Figure 2.1: Digital signing using private key [26].

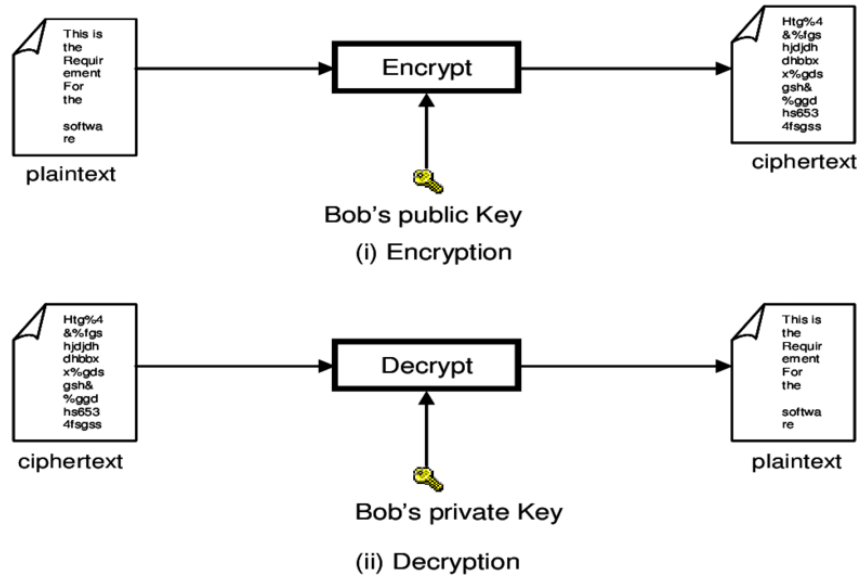


Figure 2.2: Encryption using public key [26].

Biometrics

A device supporting public-key operations is not good enough, it is still just one factor (possession). In case of a lost or stolen device, all the user's accounts would be compromised, since one device can be used for authentication to all the services. Biometry is a perfect second factor in this case since it binds the device to the actual user.

The best way to utilize biometry is access to the private keys on the device. Biometric data together with private keys should never exit the device, since a user has just one face or just one set of fingerprints and they cannot be changed, unlike passwords (unless the user is willing to undergo surgery). This is called on-device biometry. Modern smartphones have biometrics sensors like fingerprint or face scanner and capabilities to generate and store certificates in **Hardware Secure Modules (HSM)**. HSM is responsible for managing certificates and other cryptographic material.

Apple have Secure Enclave ¹, Android have keystore ². Smartphones are also widely popular and one of the few things most of us carry with us everywhere we go. A smartphone application can also be easily distributed and can provide context (IP address, time, location, etc.). It looks like a smartphone could be a key to a more secure future on the internet.

2.4 Excalibur

Excalibur eliminates passwords, moving all your existing password-protected devices and systems seamlessly to smartphone-based multi-factor authentication. Excalibur is not just a password vault. By integration with existing authentication protocols/mechanisms, it is able to inject and even change passwords, so from the user perspective – authentication is instantly password-free. Passwords are stored in a distributed manner, secured by phone

¹<https://support.apple.com/en-us/HT209632>

²<https://source.android.com/security/keystore>

biometry, and can be reconstructed only at the given client the user is authenticating at the moment.

The user only interacts with his/her smartphone – using it to provide authentication factors such as phone-based biometry, location, proximity to other devices, peer verification, PIN code if no phone biometry is present, and of course phone ownership is also a factor. All these factors are combined into a simple and straightforward user experience where the user can't do anything wrong but also can not delegate access in any way.

When there are no more passwords – there is nothing to phish. When the user is unable to delegate access, there is nothing to social engineer. Excalibur authentication always relies on at least on 2 factors: possession (Token) & inherence (biometry) or possession & knowledge (PIN), but more can be configured on per action basis, e.g. location, IP address. Excalibur logo is shown in [Figure 2.3](#).



Figure 2.3: Excalibur logo

By “freeing” authentication from passwords, unique novel authentication/authorization flows are possible – managers/colleagues are able to verify directly from their mobile phone that you are who you claim to be, instead of having to wait for IT Security to react, problems can be solved where they happen – at the branch office. Utilizing physical security and existing organization structures Excalibur makes it possible for users to vouch for each other as an additional authentication factor, exactly as in the physical world, you lose your keys, you ask your manager/colleague to open the doors for you, give you new keys, etc.

In Excalibur every action is cryptographically signed, every session is by default recorded, meaning there is a cryptographically signed record of everything the user/manager/admin does, what policy allowed him to perform that action, what factors have been verified, where why how what the whole context. Malicious actions by authorized users cannot be prevented, but by being open about the level of monitoring the user can be made acutely aware that any malicious action will be without a doubt tied to his identity.

In Excalibur every user has access to the Excalibur Dashboard, which is used to access resources protected behind Excalibur PAM, all user actions and session recordings are visible to the given user – every time the user logs in – he is made aware of the total auditability of everything he does effectively creating a psychological deterrent.

Integration with existing protocols

Excalibur was focusing on the authentication to the operating systems, especially Windows. But with the development of Excalibur PAM, there is a need for passwordless authentication to various web-based applications. There are multiple protocols that can be used for web authentication, but they differ based on used accounts (identities) and how are they managed. Two use cases are:

- **Enterprise accounts** are managed by a company, typically using some **identity provider (IDP)**. Identity providers manage identities and provide authentication services for multiple relying applications. **SAML – Security Assertion Markup Language** is a protocol widely supported by existing identity providers and it does not rely on passwords. How this protocol works is explained in [Section 2.5 – SAML](#).
- **Personal accounts** are not managed, and usually, accounts are not shared between multiple applications. There is a fairly new protocol that aims to replace password-based authentication with PKI-based authentication: **WebAuthn** [12]. It's part of the **FIDO2** [12] standard. More on this standard can be found in [Section 2.6 – FIDO2](#).

OpenID Connect (OIDC) is another protocol that can be used for enterprise accounts instead of SAML. OpenID Connect (OIDC) is an authentication protocol, which introduces an identity layer on top of the authorization framework: OAuth 2.0. In a way, it is an extension of OAuth 2.0. OIDC is a fully developed protocol for both authentication and authorization, making heavy use of JSON security tokens (JSON web token) to communicate user attributes between the service provider and the IdP.

SAML was chosen for this project based on Excalibur user needs. SAML is supported by a larger portion of applications used in enterprise environment ³.

2.5 SAML

Short for Security Assertion Markup Language, an XML-based framework for ensuring that transmitted communications are secure. SAML defines mechanisms to exchange authentication, authorization and nonrepudiation information, allowing single sign-on capabilities for web services [48]. It was developed and continues to be advanced by the Security Services Technical Committee of the open standards consortium, OASIS (Organization for the Advancement of Structured Information Standards [43]). SAML is also [24]:

- A set of XML - based protocol messages [37]
- A set of protocol message bindings [36]
- A set of profiles (utilizing all of the above) [39]

SAML is XML-based, which makes it extremely flexible. Two federation partners can choose to share whatever identity attributes they want in a SAML assertion (aka message) payload as long as those attributes can be represented in XML [43].

Terminology

The SAML specification defines the following terms [57]:

- **Subject** – An entity about which security information will be exchanged. A subject usually refers to a person, but can be any entity capable of authentication, including a software program. For the use cases, we'll discuss, the subject is generally a user of an application.
- **SAML Assertion** – An XML-based message that contains security information about a subject.

³<http://saml.xml.org/wiki/saml-open-source-implementations>

- **SAML Profile** – A specification that defines how to use SAML messages for a business use case such as cross-domain single sign-on.
- **identity provider** – A role defined for the SAML cross-domain single sign-on profile. An identity provider is a server that issues SAML assertions about an authenticated subject, in the context of cross-domain single sign-on.
- **service provider** – Another role defined for the SAML cross-domain single sign-on profile. A service provider delegates authentication to an identity provider and relies on information about an authenticated subject in a SAML assertion issued by an identity provider in the context of cross-domain single sign-on.
- **Trust Relationship** – An agreement between a SAML service provider and a SAML identity provider whereby the service provider trusts assertions issued by the identity provider. Trust is configured by exchanging service metadata [34].
- **SAML Protocol Binding** – A description of how SAML message elements are mapped onto standard communication protocols, such as HTTP, for transmission between service providers and identity providers. In practice, SAML request and response messages are typically sent over HTTPS using either HTTP-Redirect or HTTP-POST, using the HTTP-Redirect and HTTP-POST bindings, respectively.

How It Works

The most common SAML scenario is cross-domain web single sign-on. In this scenario, the subject is a user that wishes to use an application. The application acts as a SAML service provider. The service provider delegates user authentication to a SAML identity provider that may be in a different security domain. The identity provider authenticates a user and returns a security token (SAML assertion) to the application. A SAML assertion provides information on the authentication event and the authenticated subject [57].

To establish the ability to do cross-domain web single sign-on, the organizations owning the service provider (application) and identity provider exchange information, known as metadata. The metadata information contains information such as URL endpoints and certificates with which to validate digitally signed messages. This data enables the two parties to exchange messages. The metadata is used to configure and set up a trust relationship between the service provider and the identity provider and must be done before the identity provider can authenticate users for the service provider (application) [57].

There are 2 ways how the user can start authentication:

- On service provider side thus called **SP-Initiated SSO** [57]. The service provider creates a SAML request, which is transferred through the user's browser to the identity provider. IDP then authenticates the user, generates SAML Response, which is again transferred through the user's browser to the SP. This flow is shown in [Figure 2.4](#).
- On the identity provider side thus called **IDP-Initiated SSO** [57]. IDP authenticates the user without SP to generate an authentication request. This means that the service provider gets an unsolicited SAML response, so SP can't verify for whom SAML Assertions were created. This opens doors for multiple possible attacks, Man-in-the-middle (MITM) for example [3].

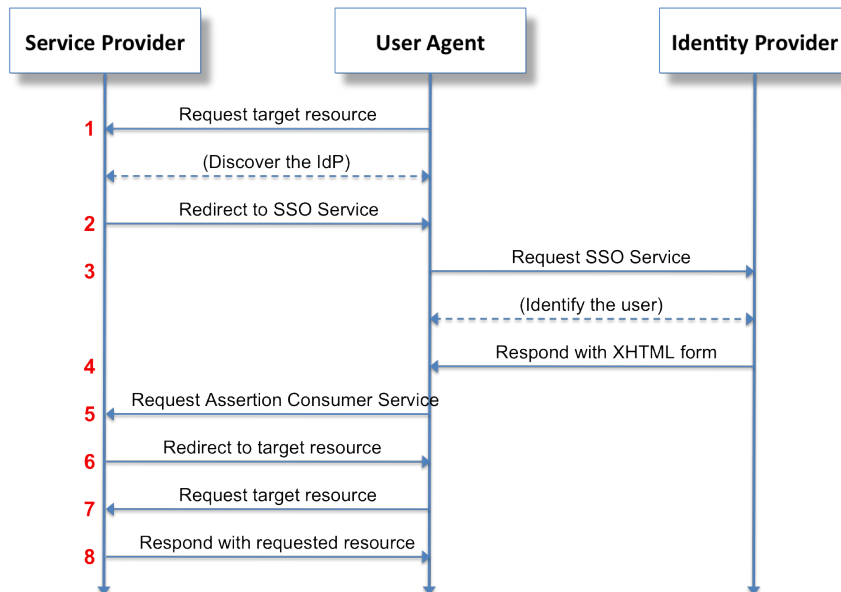


Figure 2.4: Single sign-on using SAML in a Web browser [49].

SP initiated SAML login is much more commonly used and more secure, so we will use it for explaining how SAML authentication looks like. The next chapter will describe SP initiated SSO flow with **Redirect/POST Bindings**, as shown in **Figure 2.4**.

1. User requests target resource.
2. Service provider will determine which identity should be used for this user. Based on username, domain, etc.
3. User is redirected to the IDP SSO URL. SP knows this URL from metadata. Redirect is used in this example as binding. There are multiple ways (bindings) how to access IDP. IDP authenticates user and creates SAML response.
4. Response containing SAML assertion is returned in **XHTML form** to user browser (agent).
5. Form is consequently **POST**-ed to service provider Assertion Consumer Service (ACS). This way of sending SAML Response to SP is called **HTTP-POST binding**.
6. Service provider redirects the user to the target resource. This is made using property called **Relay State**, which is not part of the **SAML Response**, but it's rather send together with **SAML Response**.
7. Target resource is requested.
8. SP responds with the target resource since the user is logged in.

Identity Federation

With SAML, identity federation establishes an agreed-upon identifier used between a service provider (application) and an identity provider to refer to a subject (user). This enables

a service provider to delegate authentication of the user to an identity provider and receive back an authentication assertion with identity claims that include an identifier for the authenticated subject that will be recognizable by the service provider. The identity provider needs to be aware of which service provider is using what identifier. [Figure 2.5](#) shows example with 2 applications (service providers).

SAML - Identity Federation

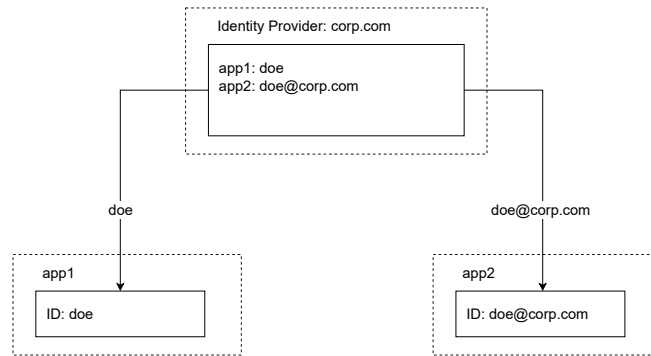


Figure 2.5: Identity Federation

Establishing Trust

Before any communication between Service and identity provider, trust needs to be established. Trust is established by exchanging SAML Entity data, known as SAML metadata. Chosen Identity and service provider metadata elements and their attributes are shown in [Table 2.1](#). Metadata examples can be found in the Excalibur SAML documentation [6], or in the [Listing 5.1](#). Metadata can be **static** or **dynamic**, which refers to the way of exchanging them. Static metadata are exchanged manually and dynamic metadata are exchanged by a trusted third-party service, called **SAML federation**.

Key Points

- SAML is an XML-based framework for exchanging security information between business partners.
- A SAML service provider delegates user authentication to an identity provider.
- A SAML identity provider authenticates a user and returns the results of a user authentication event in an XML message called an authentication response.
- An authentication response contains an authentication assertion with claims about the authentication event and authenticated user.
- Identity federation establishes a common identifier for a user between an identity provider and a service provider.
- New applications should consider using an authentication broker service or SAML library to simplify the task of supporting SAML [57].

2.6 FIDO2

FIDO2 (“**F**ast **I**dentify **O**ne”) is an open authentication standard, hosted by the FIDO Alliance, that consists of the W3C **Web Authentication (WebAuthn)** specification, and the **Client to Authentication Protocol (CTAP)**. CTAP is an application layer protocol used for communication between a client (browser) or a platform (operating system) with an external authenticator such as the security keys or even mobile phone⁴. How CTAP and WebAuthn protocols fit into FIDO2 is shown **Figure 2.6**.

FIDO2 is the latest generation of the U2F (“**U**niversal **2**nd **F**actor”) protocol. U2F is an open authentication standard that enables internet users to securely access any number of online services with one single security key instantly and with no drivers or client software needed⁵.

U2F was created by Google and Yubico, and support by NXP, with the vision to take strong public key crypto to the mass market. Today, the technical specifications are hosted by the open-authentication industry consortium known as the FIDO Alliance. U2F has been successfully deployed by large-scale services, including Facebook, Gmail, Dropbox, GitHub, Salesforce.com, the UK government, and many more⁶.

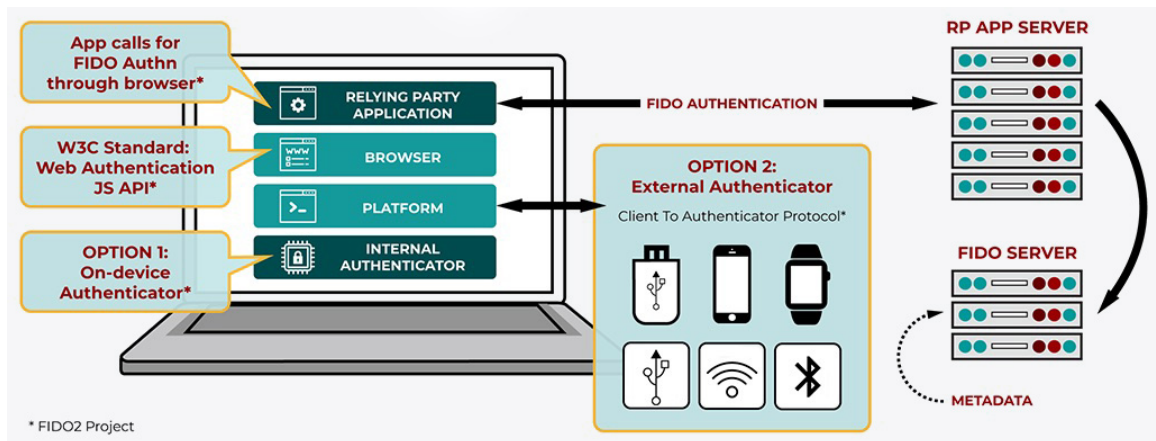


Figure 2.6: FIDO2 WebAuthn + CTAP Flow [12]

Protocols developed by the FIDO Alliance

In 2014, FIDO Alliance published the Universal Authentication Framework (**UAF**), which was intended to implement passwordless authentication through biometrics. They then added Universal 2nd Factor (**U2F**), developed by Google and Yubico as a more secure replacement for traditional OTP-based two-factor authentication (2FA). U2F included its own client-side protocol, Client to Authenticator Protocol (**CTAP**), which could be used to authenticate a token via USB, near-field communication (NFC), or Bluetooth [14].

FIDO2 is a further development of Google and Yubico’s U2F protocol with an expanded version of CTAP, now called **CTAP2**. While U2F was designed to act as a second factor

⁴<https://blog.google/technology/safety-security/your-android-phone-is-a-security-key/>

⁵<https://www.yubico.com/authentication-standards/fido2/>

⁶<https://www.yubico.com/authentication-standards/fido-u2f/>

for passwords, FIDO2's purpose is to allow authentication to become passwordless. It does this via a new web API called Web Authentication (**WebAuthn**) [14].

Summary of discussed protocols and their usage:

- **WebAuthn** defines a standard web API that is being built into browsers and platforms to enable support for FIDO Authentication.
- **CTAP2** can be used for passwordless, second-factor or multi-factor authentication, as shown in [Figure 2.7](#).
- **FIDO U2F** (previously **CTAP1**) is used for a second-factor authentication as shown in [Figure 2.8](#).

PASSWORDLESS EXPERIENCE (UAF standards)

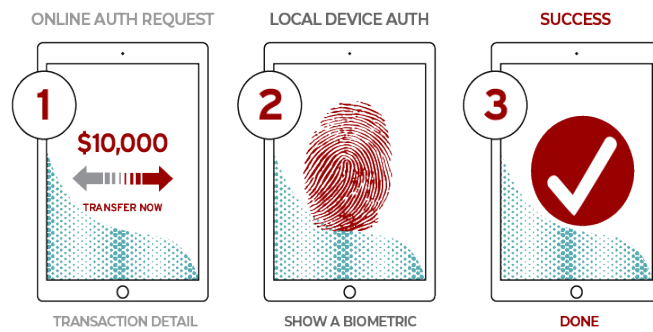


Figure 2.7: Passwordless FIDO Experience [12]

SECOND FACTOR EXPERIENCE (U2F standards)

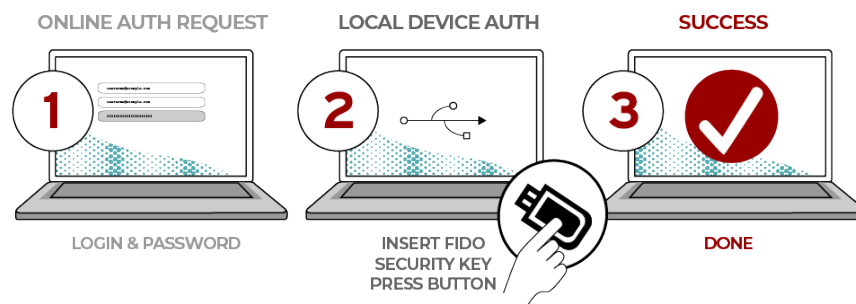


Figure 2.8: FIDO Second Factor Experience [12]

Terminology

FIDO uses slightly different terms same subjects as SAML and introduces some new:

- **Authenticator** – A cryptographic entity, existing in hardware or software, that can register a user with a given Relying Party and later assert possession of the registered public key credential, and optionally verify the user when requested by the Relying Party. Authenticators can report information regarding their type and security characteristics via attestation during registration. A WebAuthn Authenticator could be:
 - roaming authenticator,
 - dedicated hardware subsystem integrated into the client device,
 - or a software component of the client or client device.

Authenticators that are part of the client device as platform authenticators, while those that are reachable via cross-platform transport protocols (USB, NFC, BLE, etc.) are referred to as roaming authenticators. In general, an authenticator is assumed to have only one user. If multiple natural persons share access to an authenticator, they are considered to represent the same user in the context of that authenticator. If an authenticator implementation supports multiple users in separated compartments, then each compartment is considered a separate authenticator with a single user with no access to other users' credentials [21].

- **Client** – an intermediary entity typically implemented in the user agent (in whole, or in part).
- **Client Device** – the hardware device on which the WebAuthn Client runs, for example, a smartphone, a laptop computer, or a desktop computer, and the operating system running on that hardware.
- **Client-Side** – refers in general to the combination of the user's client platform, authenticators, and everything gluing it all together.
- **Relying Party** – The entity whose web application utilizes the Web Authentication API to register and authenticate users. Sometimes referred to as **Server** or service.

Web Authentication (WebAuthn)

Most of the authentication flows performed by end-users are done through web browsers. In a sense, web browsers have become the nexus between credentials and applications on the two major platforms: desktop and mobile. It is natural, then, that changes to authentication flows require support from browsers. The web, however, is built on consensus. This means that changes to the platform need to be implemented by several players. For this reason, the W3C WebAuthn Working Group was formed: to produce a new specification that can be implemented by all parties and that remains interoperable [42].

Web Authentication defines an API enabling the creation and use of strong, attested, scoped, public key-based credentials by web applications, for the purpose of strongly authenticating users.

WebAuthn API

WebAuthn, a core component of FIDO Alliance’s FIDO2 set of specifications, is a web-based API that allows websites to update their login pages to add FIDO-based authentication on supported browsers and platforms. FIDO2 enables users to leverage common devices to easily authenticate to online services in both mobile and desktop environments [21].

Web services and apps can – and should – turn on this functionality to give their users an easier login experience via biometrics, mobile devices, and/or FIDO security keys – and with much higher security over passwords alone.

FIDO’s higher security comes from the use of cryptographic login credentials that are unique across every website, never leave the user’s device, and are never stored on a server. This security model eliminates the risks of phishing, all forms of password theft and replay attacks.

The Web Authentication API (also referred to as WebAuthn) uses asymmetric (public-key) cryptography instead of passwords or SMS texts for registering, authenticating, and second-factor authentication with websites. Similar to the other forms of the **Credential Management API** ⁷, the Web Authentication API has two basic methods that correspond to register and login:

- `navigator.credentials.create()` – when used with the `publicKey` option, creates new credentials, either for registering a new account or for associating a new asymmetric key pair credentials with an existing account.
- `navigator.credentials.get()` – when used with the `publicKey` option, uses an existing set of credentials to authenticate to a service, either logging a user in or as a form of second-factor authentication.

Registration flow is shown in [Figure 2.9](#) and authentication flow in [Figure 2.10](#). More details about individual calls, messages and formats can be found in [Web Authentication standard \[21\]](#) or on [MDN Web Docs](#) ⁸.

Attestation

The attestation is how authenticators prove to the relying party that the keys they generate originate from a genuine device with certified characteristics and establish a hardware root of trust ⁹. Attestation key pair is burned into the device during manufacturing time that is specific to a device model. For example, all YubiKey 4 devices would have the same attestation certificate, or all Samsung Galaxy S8’s would have the same attestation certificate. The attestation is specific to a device model and can be used to cryptographically prove that a user has a specific model of the device when they register. When a user creates the new “credential key pair” mentioned above, the public key that is sent to the service is signed with the attestation private key. The service that is creating the new account for the user can verify that the “attestation signature” on the newly created public key came from the device [46]. Attestation is mainly used registration: if an attacker intercepts a registration message they would not be able to just swap out the new public key with their own, since the attestation signature would not match.

⁷https://developer.mozilla.org/en-US/docs/Web/API/Credential_Management_API

⁸https://developer.mozilla.org/en-US/docs/Web/API/Web_Authentication_API

⁹https://developers.yubico.com/WebAuthn/WebAuthn_Developer_Guide/Attestation.html

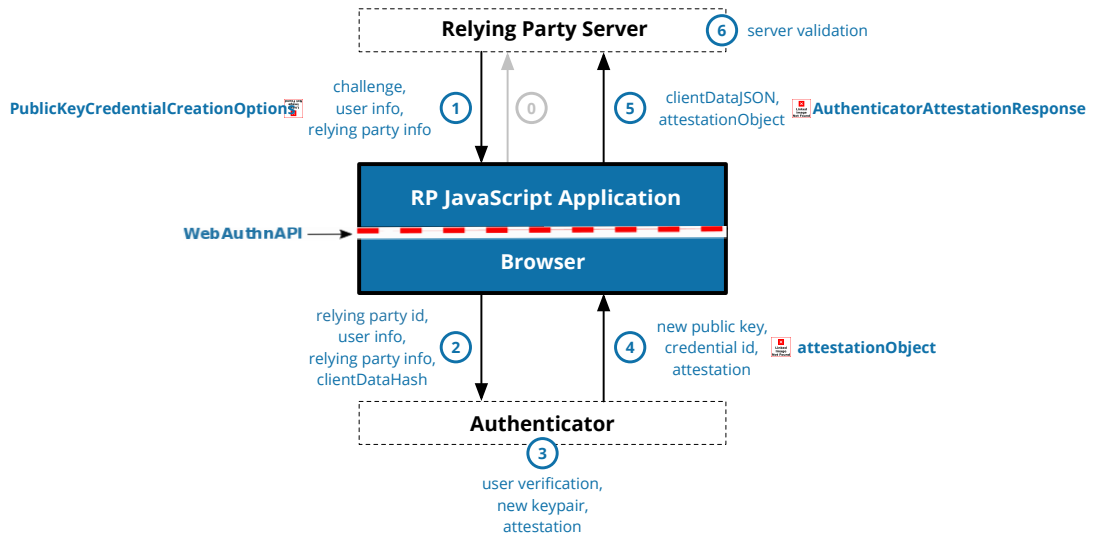


Figure 2.9: FIDO2 registration flow [21].

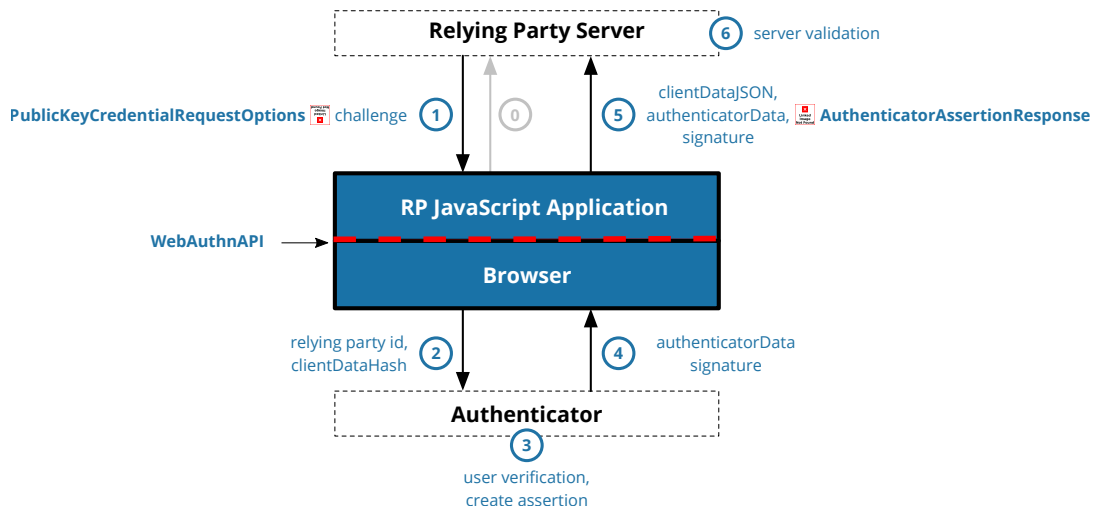


Figure 2.10: FIDO2 authentication flow [21].

Client to Authenticator Protocol (CTAP)

The Client to Authenticator Protocol (CTAP) enables a roaming, user-controlled cryptographic authenticator (such as a smartphone or a hardware security key) to interoperate with a client platform such as a laptop. CTAP is complementary to the Web Authentication (WebAuthn) [21]. CTAP is based upon previous work done by the FIDO Alliance, in particular, the Universal 2nd Factor (U2F) authentication standard.

The CTAP specification refers to two protocol versions, the CTAP1/U2F protocol and the CTAP2 protocol [4]. An authenticator that implements CTAP2 is called a FIDO2 authenticator (also called a WebAuthn authenticator). If that authenticator implements CTAP1/U2F as well, it is backward compatible with U2F.

Extensions

WebAuthn and CTAP protocols both define extensions in their standards [21] [4]. These extensions can serve very different purposes. CTAP standard defines only one extension at the time – `hmac-secret`. It can be used by the platform to retrieve a symmetric secret from the authenticator when it needs to encrypt or decrypt data using that symmetric secret [4].

WebAuthn on the other hand defines 9 extensions at the time [21]:

- **FIDO AppID Extension (`appid`)** – This extension allows WebAuthn Relying Parties that have previously registered a credential using the legacy FIDO JavaScript APIs to request an assertion.
- **Simple Transaction Authorization Extension (`txAuthSimple`)** – This extension allows for a simple form of transaction authorization. A Relying Party can specify a prompt string, intended for display on a trusted device on the authenticator.
- **Generic Transaction Authorization Extension (`txAuthGeneric`)** – This extension allows images to be used as transaction authorization prompts as well. This allows authenticators without a font rendering engine to be used and also supports a richer visual appearance.
- **Authenticator Selection Extension (`authnSel`)** – This extension allows a WebAuthn Relying Party to guide the selection of the authenticator that will be leveraged when creating the credential. It is intended primarily for Relying Parties that wish to tightly control the experience around credential creation.
- **Supported Extensions Extension (`exts`)** – This extension enables the WebAuthn Relying Party to determine which extensions the authenticator supports.
- **User Verification Index Extension (`uvi`)** – This extension enables use of a user verification index.
- **Location Extension (`loc`)** – This extension provides the authenticator’s current location to the WebAuthn WebAuthn Relying Party.
- **User Verification Method Extension (`uvm`)** – This extension enables use of a user verification method. User verification methods can be found in WebAuthn specification, Section 3.1 User Verification Methods [21].
- **Biometric Authenticator Performance Bounds Extension (`biometricPerfBounds`)** – This extension allows WebAuthn Relying Parties to specify the desired performance bounds for selecting biometric authenticators as candidates to be employed in a registration ceremony.

Table 2.1: Selected SAML metadata elements and attributes.

Elements are enclosed by “<>”.

Common metadata attributes	
<code>entityID</code>	the unique identifier of the entity attribute. Note well that the <code>entityID</code> is an immutable name for the entity, not a location.
<code>validUntil</code>	attribute gives the expiration date of the metadata.
<code><Signature></code>	element containing digital signature that ensures the authenticity and integrity of the metadata. The signatory is assumed to be a trusted 3rd party called a metadata registrar.
<code><KeyDescriptor></code>	element provides information about the cryptographic key(s) that an entity uses to sign data or receive encrypted keys, along with additional cryptographic details.

Identity provider specific metadata attributes	
<code><KeyDescriptor use=„signing“></code>	element in which the corresponding public key is included in. The identity provider software is presumably configured with a private SAML signing key.
<code><SingleSignOnService></code>	one of the essential elements with 2 main attributes. <code>Location</code> and <code>Binding</code>
<code>location</code>	attribute of <code>SingleSignOnService</code> element. Used by a service provider to route SAML messages, which minimizes the possibility of a rogue identity provider orchestrating.
<code>binding</code>	also attribute of <code>SingleSignOnService</code> element. Binding are standard URIs specified in the SAML 2.0 Binding specification [36].

Service provider specific metadata attributes	
<code>WantAssertionsSigned</code>	attribute on the <code><SPSSODescriptor></code> element declares that the service provider wants the <code><saml:Assertion></code> element to be digitally signed. This attribute causes a metadata-aware identity provider to auto-configure itself at run time.
<code><KeyDescriptor use=„encryption“></code>	element in which a public SAML encryption key is included. The service provider software is presumably configured with a private SAML decryption key [38].
<code><NameIDFormat></code>	element gives the desired format of the <code><saml:NameID></code> element in the SAML assertion.
<code><AssertionConsumerService></code>	element containing <code>index</code> , <code>binding</code> and <code>location</code> attributes.
<code>Location</code>	attribute where SP will receive SAML authentication responses from identity provider. [38, 57]
<code>Binding</code>	also attribute of <code>SingleSignOnService</code> element. Binding are standard URIs specified in the SAML 2.0 Binding specification [36].

Chapter 3

Excalibur

There's no silver bullet solution with cybersecurity, a layered defense is the only viable defense.

James Scott, Institute for Critical Infrastructure Technology

Excalibur utilizes the user's smartphone to act as a secure hardware token for any and all authentication needs. The ultimate goal is to move all forms of authentication away from passwords, replace them seamlessly with smartphone-based strong but user-friendly multi-factor authentication. Excalibur's unique value is in providing backward compatibility with all the applications, Operating Systems (OS), and services used today thus creating a bridge between the password-based present and password-free future [10].

Excalibur is not just another password manager. It accommodates multiple protocols and tools for integrating any enterprise applications into one distributed system. This chapter names Excalibur Components, how is scheme distributed, and illustrates how actions, like registration and authorization/authentication ¹ are performed.

3.1 Excalibur Components

There are 6 elemental components. These are types of components, they could, in theory, have more implementations, and some do have, e.g. Dashboard is a special case of the client, another client is Windows Client, PAM resources are also clients. The whole Excalibur topology is shown in [Figure 3.1](#).

- **Client** – There are more implementations of the client, but all of them are used to grant access to some resource. The resource can be server accessed by `ssh`, or RDP or Windows account. All types of clients can be found in [Section 3.1 – Excalibur Clients](#).
- **Server** – Essential component in Excalibur scheme. Provides a persistent network and storage point, needs to be reachable by all components, at least at the time of registration.
- **CA** – Certificate authority: Issues certificates for Excalibur components. These certificates are used mostly for signing records and messages. Can be also used for

¹Authentication is implicitly included in authorization flow, so the terms will be used interchangeably.

encryption. More on Excalibur certificates can be found in [Section 3.1 – Certificates used in Excalibur](#).

- **Facade** – Active Directory integration component. Also, act as an identity store. Can be installed on the Active Directory server and integrates Windows Domain accounts or as a lightweight service on any (Windows, Linux) system acting as identity provider service using protocols such as LDAP.
- **Dashboard** – Management interface. The whole Excalibur system can be managed from this web application. PAM is also accessible from Dashboard [Section 3.1 – Excalibur Clients](#).
- **Token** – Smartphone application. The token is based on fact that Excalibur utilizes many of the smartphone functions and sensors, not just the software made by the Excalibur team. Cryptographic material is stored in the smartphone’s HSM (Hardware security module). [Figure 3.2](#) shows factor verification on the Token.

Certificates used in Excalibur

Certificates in Excalibur are used for several purposes. First one is **client authentication** in HTTPS protocol [8]. The server verifies that HTTP client is in possession of the private key of a certificate issued by **Excalibur CA**. This done for all Excalibur components which are connecting to server or CA using HTTPS – **Facade**, **Token**, **Client** (for now). The first connection is made with a built-in certificate. After first connection **Certificate signing request (CSR)** ² is created and submitted to the **Excalibur CA**. From this moment all actions with certificates are made using a newly issued certificate. The only purpose of the built-in certificate is to connect to the CA and creation of the new certificate. This needs to be done because **Excalibur CA** allows connections only to clients with Excalibur signed certificates.

Every Excalibur components have a certificate that is used to verify that message was really sent by that component. This is done by providing **digital signature** of every message. These signatures are then stored, so every action in Excalibur is audited. Another use of certificates in Excalibur systems is **encryption**. Only sensitive data, like passwords, are encrypted using issued certificates.

Overview of Excalibur certificates can be found in [Table 3.1](#).

Table 3.1: Excalibur components certificates.

Component	built-in	issued
Server	✓	×
Facade	✓	×
Token	✓	✓
Client	✓	✓
User	×	✓

²<https://www.globalsign.com/en/blog/what-is-a-certificate-signing-request-csr>

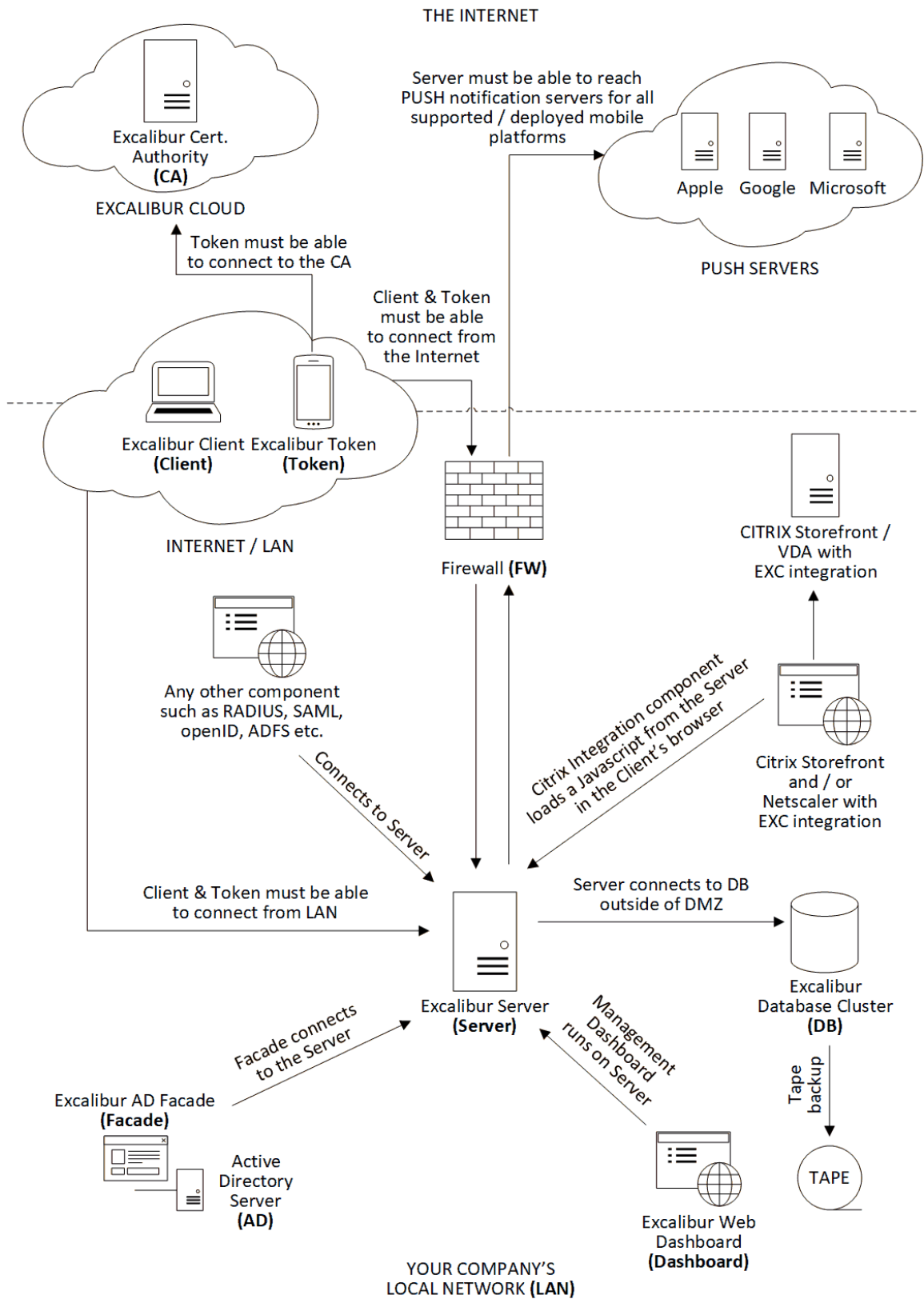


Figure 3.1: Excalibur system Topology

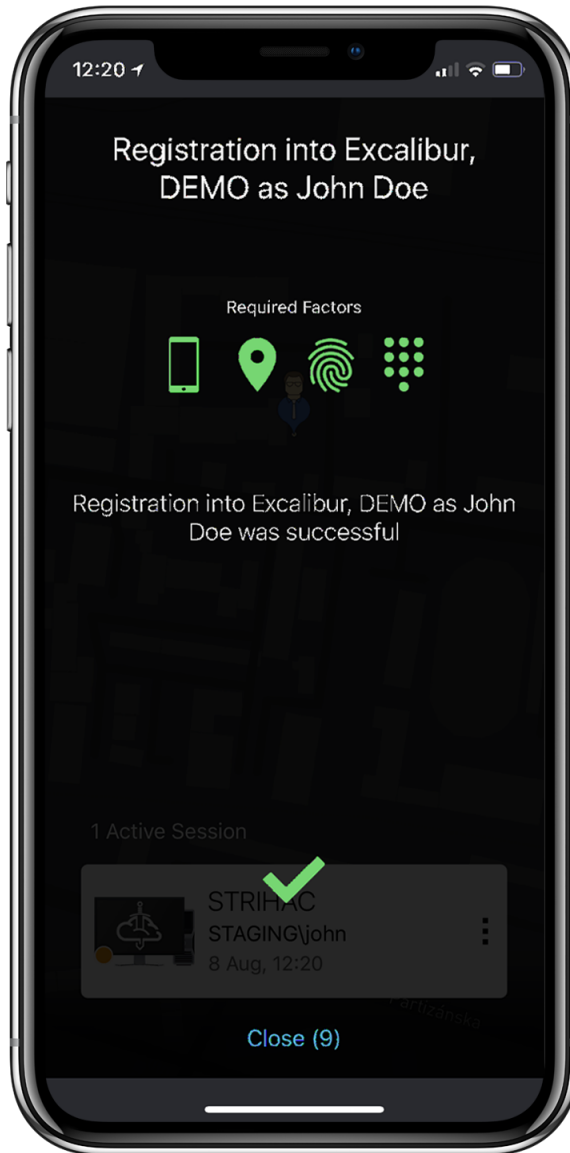


Figure 3.2: Mobile application factor verification (location, biometrics, etc.)

Excalibur Clients

Excalibur Client is any component that can grant access to a resource. It could be some application (**Windows Client**), webpage (**Dashboard**), scripts (**WebSDK Component**) or some service like **Web-Proxy**. Even **Token** can be considered a client since the user can view its password.

- **Windows Client** – Windows application capable of logging into Windows with biometrics-based Excalibur application (**Token**).
- **PAM** – Privileged Access Management: PAM is part of **Excalibur Dashboard**, can be used to access resources using just the web browser. Supported protocols are:
 - **RDP**
 - **SSH**
 - **VNC**
 - **WEB** pages – proxies **WEB (HTTPS)** applications. Built on top of **http-proxy** or **virtual browser**.
- **WebSDK Components** – JavaScript scripts, that can be loaded in the site and act as a gateway to a web resource. User sees QR Code, after scanning it and logging into Excalibur **WebSDK Component** is able to reconstruct password and inject it to **HTML** field. Except for password inject, there are other ways how to deliver passwords to a protected webpage, such as **HTML GET/POST** request. **WebSDK Component** can also be used for registration to the Excalibur, or to find answers in the manual, which is also part of the **WebSDK**. Typical view of the **WebSDK** can be found in **Figure 3.3** and more pieces of information in the Excalibur documentation [11], specifically in the Administrator Dashboard Manual ³ These components can only be injected into the predefined URL. URL of injected website and type of **WebSDK Component** is chosen from **Dashboard**.



Figure 3.3: WebSDK Component as seen by the user.

³<https://docs.xclbr.com/v3/getting-started/excalibur-administrator-dashboard-manual/#components>

3.2 Excalibur PAM

Excalibur PAM is able to provide controlled access to most services/servers/applications by supporting a wide range of protocols. Supported protocols are listed in [Section 3.1 – Excalibur Clients](#). All these applications can be accessed by any web browser supporting HTML5, even mobile web browsers are supported. SSH, VNC and RDP protocols support is achieved using Excalibur heavily customized **Apache Guacamole**⁴. Web resource access is provided by either **HTTP Proxy** or **Virtual Browser (VB)**.

Access to PAM-protected resources (such as HTTP Proxy or Virtual Browser protected resources) is granted only on a whitelist basis and only to strongly multi-factor authenticated users. Excalibur is installed to the existing customer architecture, but it can also be installed in a separate network segment when network access to every protected resource is allowed.

Guacamole

Guacamole is an HTML5 web application that provides access to desktop environments using remote desktop protocols (such as VNC or RDP). Guacamole is also the project that produces this web application and provides an API that drives it. This API can be used to power other similar applications or services.

“Guacamole” is most commonly used to refer to the web application produced by the Guacamole project using their API. This web application is part of a stack that provides a protocol-agnostic remote desktop gateway. Written in JavaScript and using only HTML5 and other standards, the client part of Guacamole requires nothing more than a modern web browser or web-enabled device when accessing any of the desktops served.

The fundamental reason to use Guacamole is constant, worldwide, unfettered access to your computers.

Guacamole allows access to one or more desktops from anywhere remotely, without having to install a client, particularly when installing a client is not possible. By setting up a Guacamole server, you can provide access to any other computer on the network from virtually any other computer on the internet, anywhere in the world. Even mobile phones or tablets can be used, without having to install anything.

Guacamole communicates with the browser through its own protocol, so there needs to be middleware that interprets Guacamole protocol messages and renders HTML5 webpage which is then shown to the user. Guacamole architecture is shown in [Figure 3.4](#). Guacamole supports session recordings, file transfer logging, and input indexing, which means all user activity can be captured. Recordings can be text-based, so they can be efficiently compressed. All of this activity logging is made server-side, so a would-be attacker can't disable them.

Virtual Browser

Virtual Browser is Excalibur's implementation of **Remote Browser Isolation (RBI)**. It works by streaming vector images from a server-based instance of Chromium while allowing full remote control of this browser instance by the user directly from his web browser. This is beneficial mainly for security reasons since the client's browser does not have access to the original HTML page, nor JavaScript or cookies, etc. DOM is executed in the **Virtual**

⁴<https://guacamole.apache.org/>

⁵<https://guacamole.apache.org/doc/gug/guacamole-architecture.html>

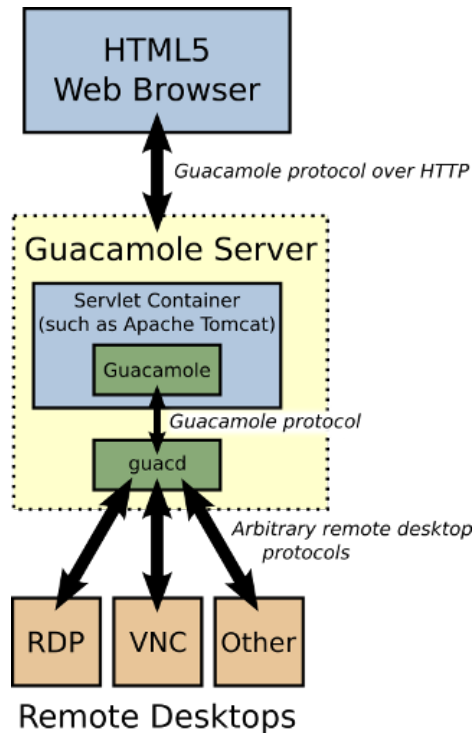


Figure 3.4: Guacamole⁵ architecture.

Browser, so a web page is completely isolated from the client’s browser, thus the name – **Virtual Browser**. In contrast to video codecs - **Virtual Browser** works with vector graphics and only the modified parts of the image are transferred. This means that the image/text quality presented to the end-user is always lossless, no matter the resolution, frame rate, or other factors. As with all Excalibur PAM components - **Virtual Browser** records all sessions. User input is indexed and saved, so it can be used for pattern matching or other analytics purposes.

Virtual Browser also uses **Guacamole**. It defines a new protocol (also based on images and user input) but **webkit** rendering engine is used to create images, which are then sent to the **Guacamole**. Architecture based on the **Virtual Browser** is shown in [Figure 3.5](#).

HTTP Proxy

HTTP Proxy is a light-weight alternative to **Virtual Browser**. It serves a similar purpose as the **Virtual Browser** but does not share the same security advantages such as complete DOM isolation, yet it has its own benefits – mostly compatibility-wise. It terminates **HTTP(S)** connection from the client and creates a new one to the server, effectively acting as a Man-in-the-Middle. Main difference between the **HTTP Proxy** and the **Virtual Browser** is where **HTML DOM**⁶ is executed / interpreted. **Virtual Browser** executes all code on the server-side, which has its security benefits and compatibility limitations. On the other hand, **HTTP Proxy** just proxies traffic, so the code is executed in the client’s browser.

HTTP Proxy injects a special script to the web page, which records user activity. For recording user activity **rrweb** project is used⁷. Recordings are captured directly in the

⁶https://www.w3schools.com/js/js_htmlDOM.asp

⁷<https://www.rrweb.io/>

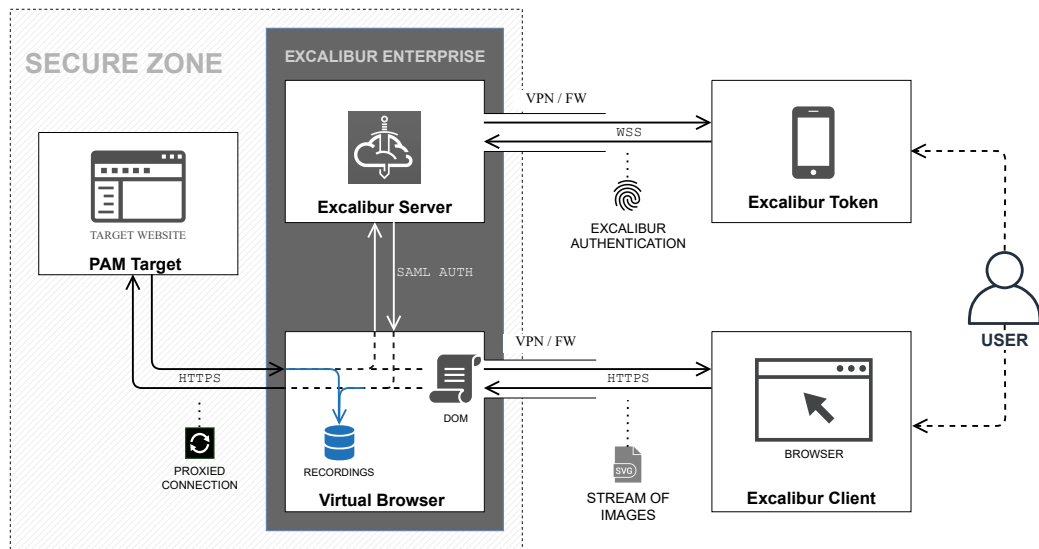


Figure 3.5: Architecture diagram of the Excalibur Virtual Browser.

user's browser and sent to the HTTP Proxy through websocket. The biggest advantage of HTTP Proxy is that because DOM is executed on the client browser – compatibility with legacy SW/HW solutions that need to interact with the webpage is preserved – such as USB security tokens, or other HSM solutions, etc.

HTTP-Proxy does not use **Guacamole**. It serves a similar purpose as **Virtual Browser** but does not share the same security advantages such as layers separations. This doesn't create so much overhead as the virtual browser approach but is also less secure. It can be thought of as a lightweight virtual browser. The architecture diagram of the HTTP Proxy is shown in the [Figure 3.6](#).

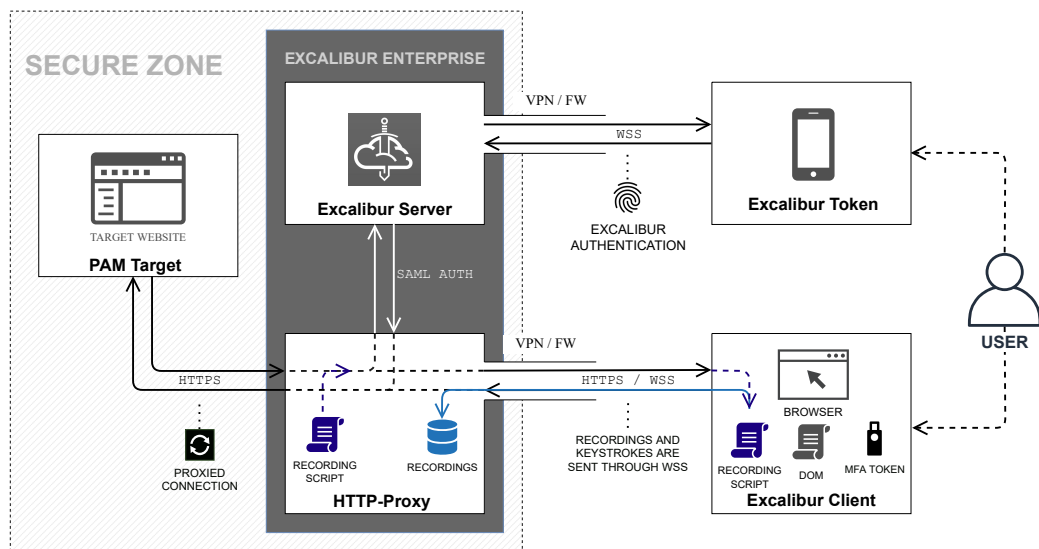


Figure 3.6: Architecture diagram of the Excalibur HTTP Proxy.

3.3 Excalibur system actions

There are 2 main actions in the Excalibur system, i.e., Registration and Authorization. Because credentials are bound to the user's smartphone and their biometrics (face, fingerprint), Excalibur is simultaneously authenticating and authorizing the user. In plain English, the user needs to prove that he is really him (by providing biometrics) every time he wants to access some resource (open `ssh` connection).

Policies

The policy is a set of rules specified for an action performed by an Excalibur user which needs to be fulfilled to allow the action. The policy can specify which factors need to be provided by the user, allows the action to be performed just on some subset of clients, inside specified sets of geofences, and/or at the right time and day of a week [10].

Policies can contain any of the following rules and their combinations:

- Factors
 - Fingerprint
 - PIN
 - Face recognition
- Geofences – a subset of geofences where the user must be physically located to perform the action.
- Clients – a subset of clients on which the action is allowed to be performed.
- Time of the day
- Day of the week
- IP address of the client
- IP address of the token
- Additional verification by manager / admin / support center

When registering, Token generates `privateKey`, `publicKey` pairs for each factor it supports using HW-backed secure enclave, `publicKey` is then signed by Token with `userPrivateKey` and sent to the Server to be stored for Factor Verification.

Fingerprint private-public key pair is generated in such a way that every future signing of the data with a private key requires a fingerprint to be provided to unlock it.

PIN privateKey is only accessible after entering the correct PIN, the rate limit is applied both locally and on the Server.

Location privateKey is used to sign the location sent to the Server [10].

Registration

Registration is a process that binds together user (person) with their smartphone (`token`) with Excalibur account using biometrics. Registration can be, in theory, made on every client, but right now only 2 clients supports registration: `Dashboard` and `Windows Client`. The registration process is shown in [Figure 3.7](#) and the steps are:

1. User enters credentials (username, password) and email address.
2. (a) User's credentials are send to the **Server**, then forwarded to **Facade**,
 (b) where they are confirmed against identity store ⁸.
 (c) Result is returned to **server**.
3. When entered credentials are valid, email with magic link (and fallback OTP – One-Time-Password) is generated and send to entered email address ⁹.
4. (a) User opens this email on any device,
 (b) clicks on “magic” link (or rewrites OTP code ¹⁰) proving he owns entered email address.
 (c) OTP code is validated on the server and the result is sent back to the client.
5. When the email address is validated QR code is generated and shown to the user.
6. User scans registration QR code with Excalibur mobile application.
7. (a) User provides biometrics or PIN ¹¹. **Token** validates biometrics locally.
 (b) At the same time **token** send other factors to the **server**.
 (c) **Server** validates all factors and policies and returns result.
8. When all policies and factors are valid, **token** sends registration data together with login data to the **server**.
9. User is registered and automatically logged in using login data.

Asking users for account and password as the first step has some security benefits. The user needs to prove that he knows something before more system resources are allocated for him. In combination with rate-limiting (Captcha), this acts as sufficient DOS (Denial-of-Service) protection [27].

When a password is verified by **facade** it encrypts the password with a random key and this random key is then encrypted with **token's** public key. Encrypted password is called **server crypto part** and encrypted random is called **token crypto part**. These crypto parts are send back to **server**, where **server crypto part** is stored and **token crypto part** is forwarded to **token**. **Token** stores it's crypto part in HSM.

Authorization

Authorization is simpler than the registration, as shown in **Figure 3.8**, and the steps are:

1. **Client** generates authorization QR code.
2. User scans QR code with Excalibur applications.
3. (a) Mobile application asks user for biometrics

⁸Excalibur does not need any passwords, but existing passwords are the best way how can be Excalibur integrated to existing enterprise infrastructure.

⁹OTP is needed when users cannot access mail client on the same device he is registering, e.g. registration via Windows client – cannot log in to open email

¹⁰When he opened the email on another device where he is registering.

¹¹PIN code is used when biometrics are not available.

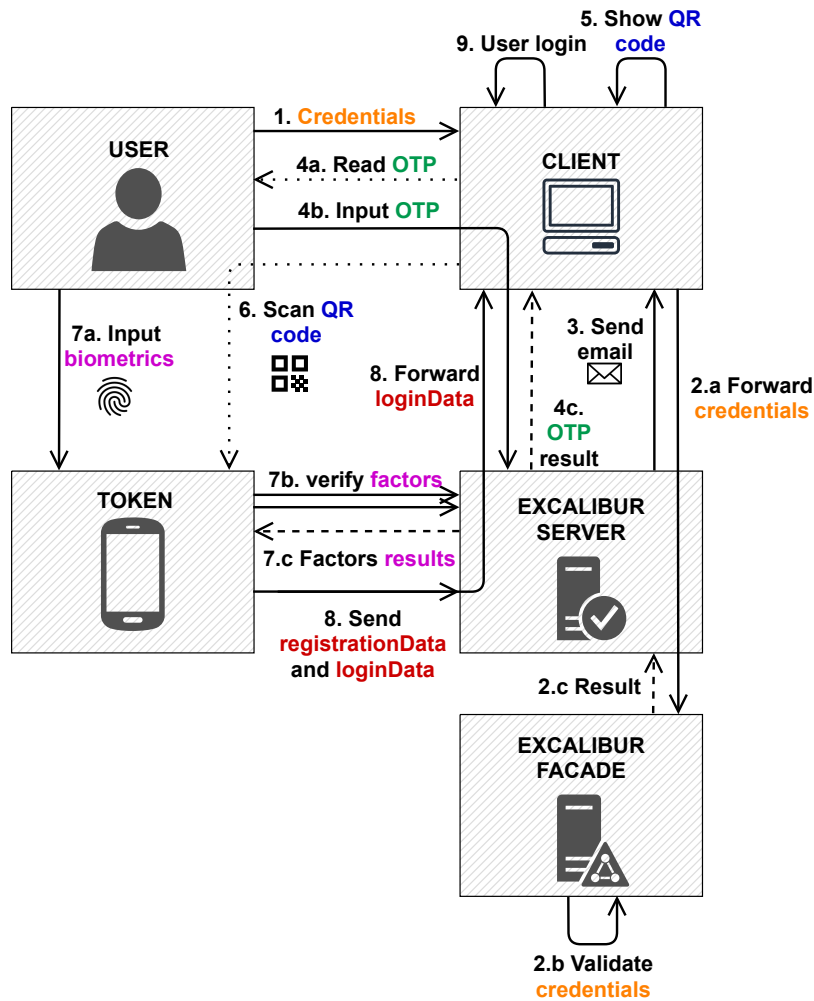


Figure 3.7: Excalibur registration process.

- (b) and simultaneously sends factors (locations, biometrics, etc.) to the **Excalibur** server.
- (c) **Server** verifies factors and policies and return result (c).
- 4. (a) When factors and policies are valid, **token** sends its crypto part to the server. Crypto part is encrypted with **facade's** public key, so only the **facade** can read it.
- (b) This message is forwarded together with **server's** crypto parts to the **facade**.
- 5. **Facade** reconstructs the password and encrypts it with a random key, which is then encrypted with **client's** public key.
- 6. Encrypted random key with encrypted password is send back to **client** through **server**.
- 7. **Client** decrypts password with it's **private** key and uses it for login.

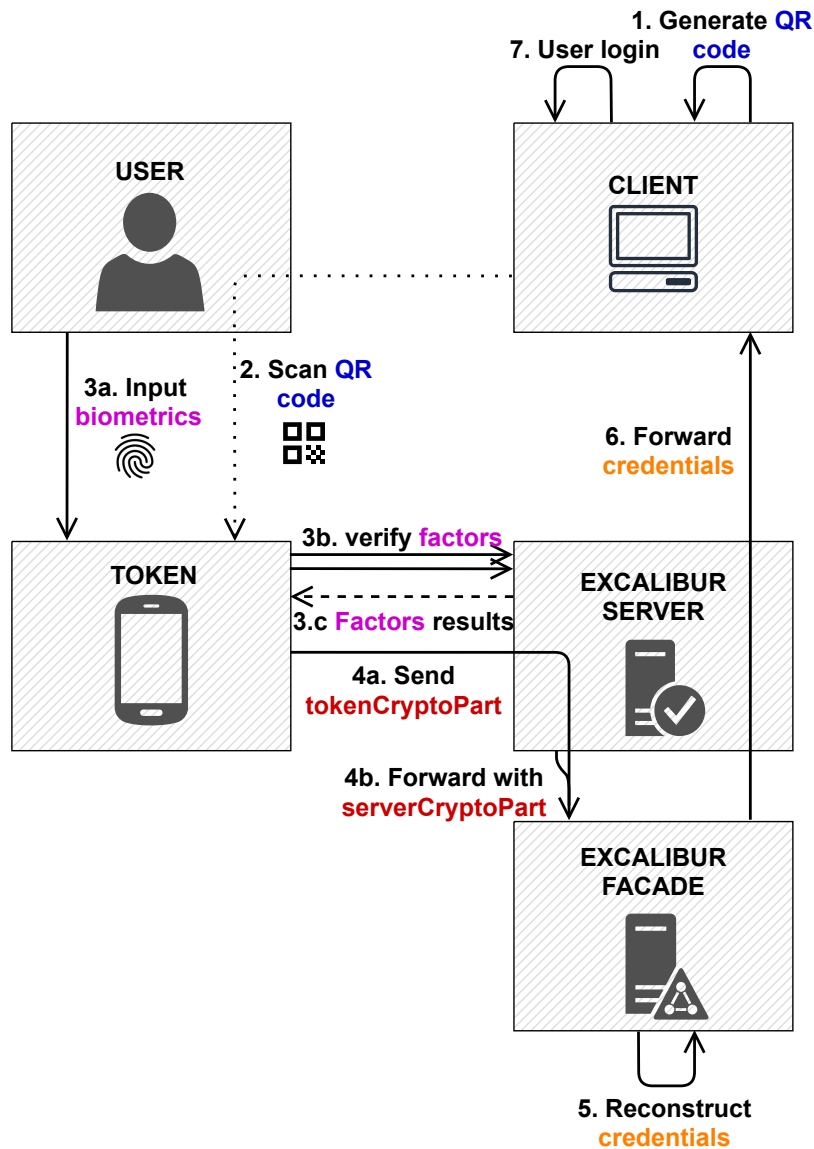


Figure 3.8: Excalibur authorization process.

Current State of the Excalibur Server

Latest deployed version of the Excalibur Server is v3.3.5. Right now we are developing a new major version, which we call v3.5. It is based on a new architecture, but it is based on the same principles. Most of the cryptographic operations in our crypto scheme stay the same and some new ones are added.

Excalibur Server v3 is built on top of the Node.js cluster so a single instance of Node.js runs in a single thread. A single thread is called a *worker*. These workers have a single point of entry and exit for every request. Communication between threads is done by our own implementation of inter-process communication (IPC) [16]. Workers can be started on-demand, or at the start of the Excalibur Server. Multiple workers of the same type can be run simultaneously in which case the IPC decides to which worker will be the request delivered. Some workers are stateless, but some cannot be because of the

other architecture aspects like database connections and connections to the tokens, clients, etc. The bottleneck is not the computation power, but communication pipes, as we find out. **Excalibur Server v3** is not actively developed right now, only bug fixes and minor improvements are developed.

Our focus shifted to the new **Excalibur Server v3.5**. This server is a major overhaul of the existing one. A new database scheme with a new IPC, which we now call **router**, was developed. Workers are now services and are standalone Node.js applications communicating via a router. Outer routers for communication with **Facade**, **Clients** and **Tokens** were also added. This enables us to really scale services. **Excalibur Server v3.5** is more cloud-focused than **Excalibur Server v3**, but it is still a hybrid model since it needs to have some components installed in the customer's infrastructure. A new certificate issuing process was also developed.

Excalibur Server v3.5 is a platform that should developers enable to built secure, testable but open software on a solid foundation. Testing the old solution was nearly impossible since a single action flow through multiple components on multiple platforms and user action was needed. Some components were purely documented and after some time not maintained. Version 3.5 is using various techniques and technologies for better maintainability of the product.

Right now **Excalibur Server v3.5** is still in development and just the basic building blocks were implemented. It is not yet ready for implementing various extensions, which are the focus of this thesis. Implementation is slower than with **Excalibur Server v3**, but it enables us to continuously test our solution and to expand/modify features.

Chapter 4

Integration design

This chapter consists from three main sections: **SAML integration**, **FIDO case study** and **Single sign-on (SSO) integration**. SAML is a standard for exchanging authentication data, which are known to the Excalibur. The first section is divided into subsections about requirements, protocol implementation, and binding with the Excalibur crypto scheme.

FIDO, on the other hand, is a standard for authentication. How it could be integrated into Excalibur is not that simple as with SAML protocol. The FIDO case study section is focused on explaining various use cases, where FIDO could be advantageous to use.

The last section is discussing requirements for implementing SSO to the Excalibur with regards to existing clients and also newly integrated.

4.1 Excalibur – SAML integration design

SAML is a passwordless protocol – there is no shared secret between parties. Traditionally a shared secret is used for establishing trust, so if the other party knows the secret, it can be trusted. In SAML trust is established by exchanging metadata containing certificates with a public key. This key will then be used to validate messages. SAML does not specify the actual method of authentication, it is only used to present SP fact that the user authenticated to the IDP. This is ideal for our use case since users can be authenticated using Excalibur a then SAML assertions can be created. One of the main advantages of Excalibur is its distributed crypto scheme, which ensures that even when an attacker gains control of one component, he will not gain access to any system. Using a password ensures this since passwords can not be reconstructed unless user authorize such action using their phone. This could be a problem with SAML since SAML does not use any shared secret.

As told before, SAML is built on the concept of trust. Trust is established by exchanging metadata, which means that implemented solution needs to be able to generate metadata for the SP and also manage SP metadata. Imported SP metadata are needed for valid SAML communication, so managing service providers means managing SP metadata. IDP metadata needs to be publicly available, so an administrator of the service provider can download them and import them into the SP. Some service providers also require a signing/encryption certificate to be imported since not all IDP includes their certificates in the metadata. So there should also be a way to download these certificates.

SAML is an XML-based framework, so creating and validating actual SAML messages is not a trivial task. Last step should be integrating SAML Component with the Excalibur authentication.

For the sake of recapitulation, SAML implementation **requirements** are:

1. managing and generating SAML metadata,
2. exporting SAML metadata and signing and encryption public keys,
3. creating and validating SAML requests and responses,
4. binding user authentication to the creation of SAML assertions. Only when the user really logs in, assertion is created. Even when the attacker gained control over the server, he can't be able to generate SAML assertion.

SAML library

The first, second, and third requirements can be satisfied by using the SAML library. Most of the Excalibur server code is run in `nodejs`¹, so one of possible libraries to use is `samlify` [32].

New API endpoints will be created: `SingleSignOnService` as named in metadata (Table 2.1). This endpoint will accept only GET and POST HTTP messages containing `SAMLRequest` object. GET and POST are the most commonly used SAML bindings, so it is a good start, artifact binding can be added in the future. In metadata this endpoint will be presented as two `<SingleSignOnService>` elements with different `binding` attribute.

These endpoints will be used for receiving `SAML AuthnRequests`. SAML library will parse and validate this request based on the SP metadata, which were imported beforehand. After validating the SAML request, user authentication follows. User authentication is responsibility of the Authentication component, which will be discussed in the Section 4.1.

After successful authentication user data, given SP metadata and `AuthRequest` will be used to create `SAML Response`. SAML response can be signed and optionally encrypted, depending on the SP metadata and `AuthRequest`. `SAML Response` will be then send to the URL from `<AssertionConsumerService` element from given service provider (SP) metadata. Based on `binding` attribute of `<AssertionConsumerService` element POST or REDIRECT HTTP method will be used. `SAMLResponse` will be either posted in body of the HTTP-POST message, or in `HTML Form`, which will be returned to user browser and automatically submitted. This process is shown in Figure 4.1.

Certificate store

Certificates used in SAML flows can't be stored on the server, since Excalibur is deployed as a Docker container to the customer server, which typically does not have an HSM. Certificates are common for all Excalibur users, so they can't be stored on the user phone either. One of the solutions is to move them deeper into the customer infrastructure, to the `Facade`. `Facade` is installed on customer's Active Directory (AD), where they should stay in HSM. This solution complicates flow by introducing a round trip to the `Facade` for each authentication request.

¹<https://nodejs.org/en/>

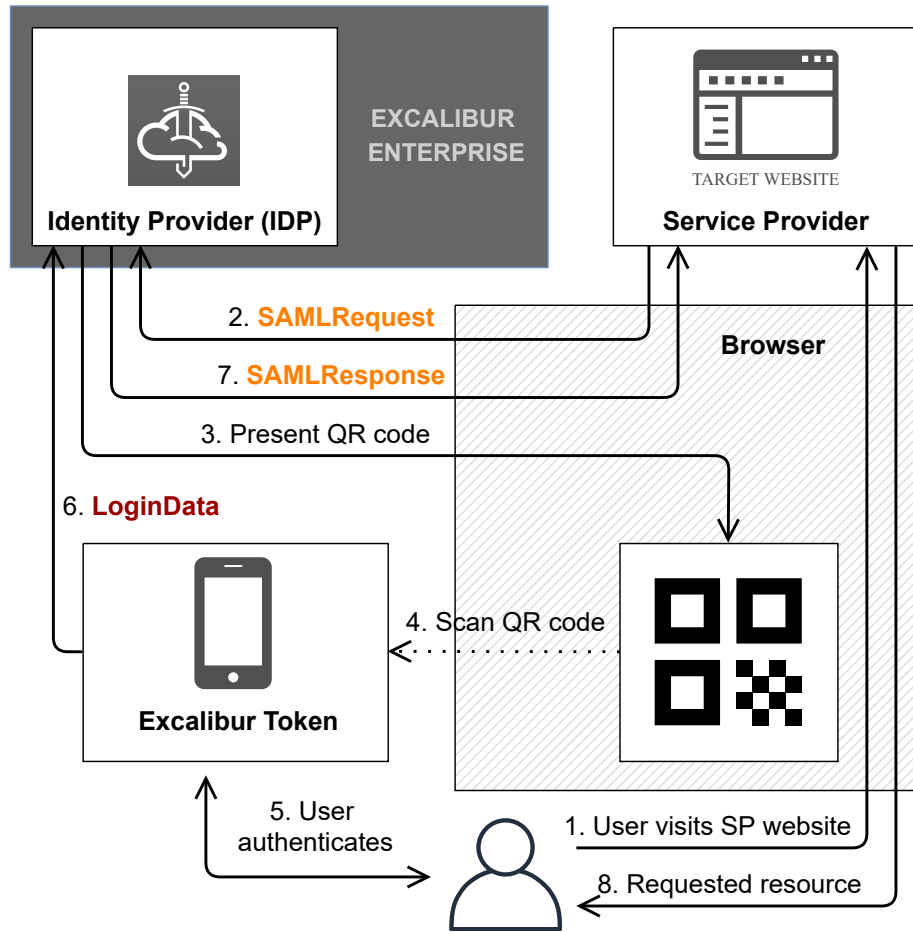


Figure 4.1: Excalibur login via SAML.

Adding Excalibur Authentication to the SAML Flow

After successful validation of the incoming `AuthnRequest`, standard Excalibur authentication will start by presenting the QR code. QR code can be presented to the user by one of the existing Excalibur components – `WebSDK` client, [Section 3.1](#). Diagram showing authentication via SAML using `WebSDK` client is show in [Figure 4.2](#).

This script will be presented to the user with hidden `HTML` form. After the user authenticates, user data will be filled into the form and submitted back to the server. This requires another endpoint where the user data will be sent – `response` endpoint.

`WebSDK` clients don't validate if returned values are really from the server. `WebSDK` clients are intended to inject password in `HTML` form on target applications, so the validation is done by that target system. But since SAML does not need passwords, the server needs to verify that login data submitted back to the server really came from the server and that the user really interacted with the token. This can be solved by including a message signature.

Server-side Authentication Component

The second solution is to present a QR code to the user and wait for authentication on the server-side. For this to work, the server will need to link authentication action with specific

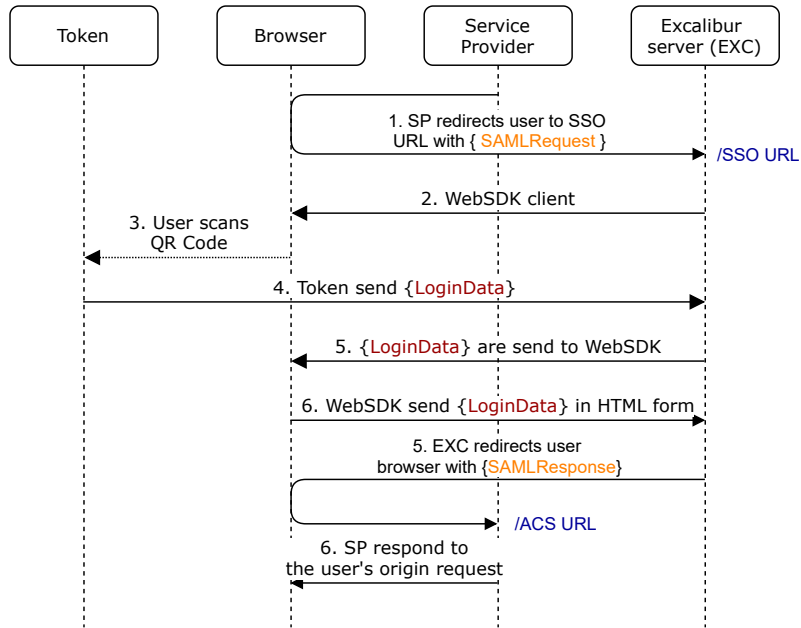


Figure 4.2: Excalibur login via SAML using WebSDK.

browser (QR code changes every 15 seconds). Message and signature validation still need to be done the same way as described in the previous paragraph. This method is slightly more secure because user information will go all the way to the user's browser. At least not until authentication is completed. This information is then embedded in SAML response and possibly send to the user (based on SP binding). Developing a new authentication method can be taken as a negative, but it will simplify the authentication process and would make it a little bit more secure. How would this method work is show in Figure 4.3.

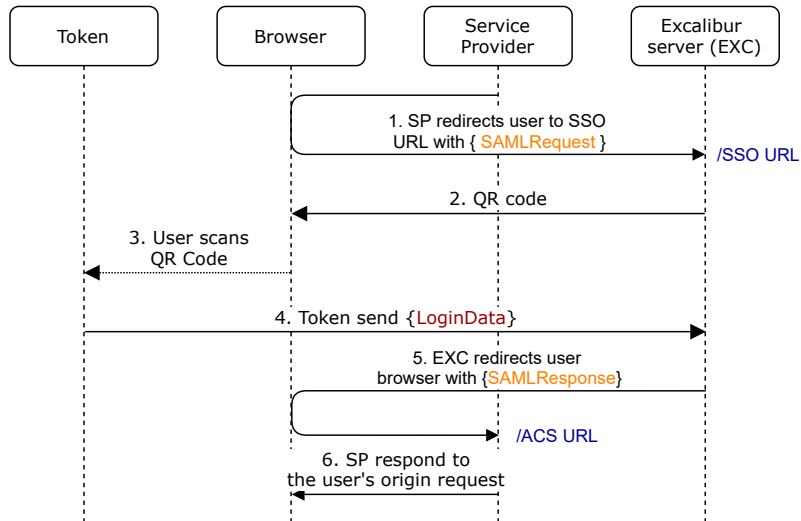


Figure 4.3: Excalibur login via SAML using server-side method.

4.2 FIDO2 Case Study

The goal of the FIDO2 project is to standardize an interface for authenticating users to web-based applications and services using public-key cryptography. Whereas SAML is a protocol for exchanging authentication, so any authentication mechanism can be used. WebAuthn ², on the other hand, specifies how authentication needs to be implemented. This is the main difference between SAML and FIDO2 projects.

In typical FIDO flow website acts as a Relying Party and the device on which the website is rendered acts as a FIDO Authenticator, i.e. website is rendered on the computer, so a computer is a FIDO2 Authenticator. When the computer does not have FIDO capabilities it can utilize CTAP2 protocol and contact some security keys (roaming authenticators) (e.g. smartphone, USB key, smart card).

Here is a brief introduction to the use cases presented in this section.

- **Excalibur as FIDO2 Authenticator.** Excalibur would act as a FIDO2 authenticator. Can be useful for logging in to the web application, that supports FIDO2 with Excalibur.
- **Excalibur as a second factor (U2F).** Similar case as previous one, but utilizing U2F, so Excalibur would only be a second factor, a password is still required.
- **Adding more factors to the Excalibur.** This use case has 2 sub-cases. We can authenticate using security keys to the Excalibur, improving security, or we can retransmit these security keys to the relying party. Retransmitting could be useful when accessing a site via Excalibur PAM.
 - **Authentication to the Excalibur using security keys.**
 - **Retransmitting security keys.** Excalibur PAM can be used to access various websites. When this website requires some security key, it would not always work out of the box. FIDO2 could help retransmit these security keys through PAM.
- **Replacing Excalibur Authentication with FIDO2.** FIDO2 is developed with similar assumptions in mind. This is a direct comparison between the FIDO2 and the Excalibur authentication flow.

Excalibur as FIDO2 Authenticator

Excalibur would act as a FIDO2 (WebAuthn) authenticator and the website as a Relying Party, as shown in [Figure 4.4](#). Implementing this solution would bring true passwordless login to the websites. It is an alternative to the SAML login. The problem with this approach is that really no major website implements WebAuthn [47]. Could be great one day, but right now it is not worth investing time and effort, since it would not be used. Moreover, if the target website would support FIDO2, there would be no need for Excalibur, since Excalibur and FIDO2 have the same requirements for the smartphone. Excalibur could add management capabilities over who is registered and so on, but for this are better-suited protocols, e.g. SAML.

²Web Authentication (WebAuthn) is a core component of the FIDO2 Project.

Excalibur as WebAuthn Authenticator

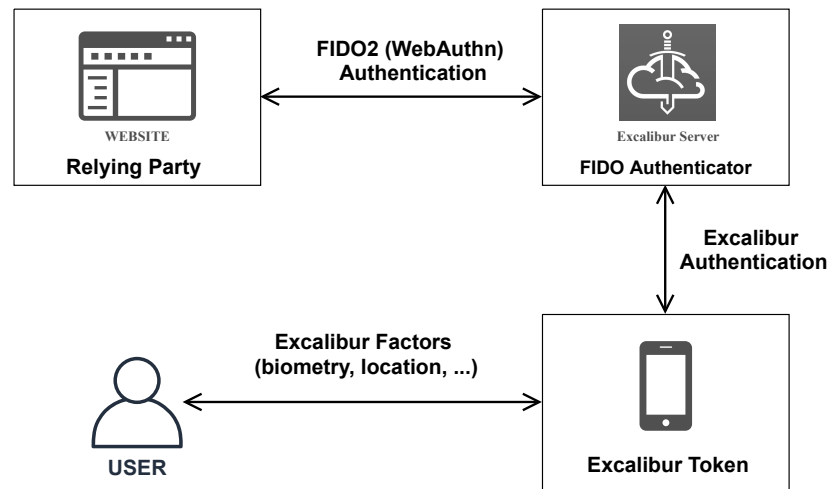


Figure 4.4: Excalibur as WebAuthn (FIDO2) Authenticator.

Excalibur as a second factor (U2F)

Universal 2nd Factor (U2F) is an open standard that strengthens and simplifies two-factor authentication (2FA). FIDO2 is the latest generation of the U2F protocol. It is a very similar case to the previous one, but Excalibur would act as a second factor (password would still be the first one). Unlike WebAuthn, U2F is well supported in both enterprise and personal solutions. Excalibur would act as a security key, that could be used as 2nd factor for any website supporting security as the second factor, as shown in [Figure 4.5](#). A little more on the topic of U2F is discussed in [Section 4.2 – Retransmitting security keys](#). Implementation would include some kind of client: Excalibur client installed on the computer, a mobile application, or a browser extension. This client would implement the U2F (FIDO) protocol, so it would act as a software security key to the websites. Excalibur is trying to shield users from passwords, not just add another step to the already complicated authentication process. We also didn't see demand for this kind of solution from our current nor potential customers.

Adding more factors to the Excalibur

This use case is about utilizing FIDO in the Excalibur authentication. A large portion of companies is already using some kind of the second factor, e.g. USB tokens, smart cards. These security keys have issued certificate, which is used to authenticate the client. This use case can be further divided based on target and type of certificate validation. The target could be either a general Excalibur client or the Excalibur Dashboard, specifically. Client authentication can be done on either HTTPS or the application layer.

Excalibur as U2F Authenticator

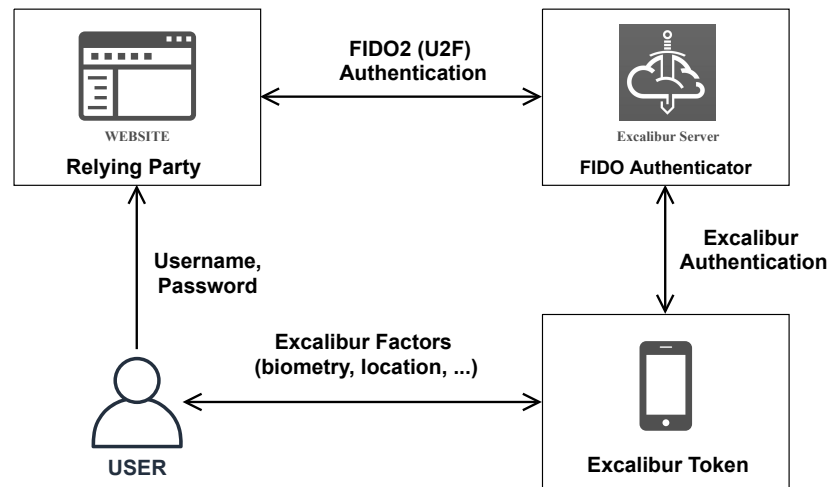


Figure 4.5: Excalibur as second factor using U2F protocol.

Authentication to the Excalibur using security keys

Excalibur Dashboard is a webpage, so standard **HTTPS Client Authentication** can be used. A typical use case would be authentication to the Excalibur PAM since PAM is a part of the Dashboard. **HTTPS Client Authentication** to the Dashboard (PAM) can be used if the certificate root can be exported and subsequently imported to the Excalibur Server. This approach is possible also with already issued security keys. It is also recommended to forbid access on the network level to all the targets that are secured by Excalibur PAM, so users can only access them through Excalibur Server. **HTTPS Client Authentication** is already used for some deployments, so there is no need to implement FIDO or anything else.

On the other hand, **Excalibur Windows Client** is already authenticating against the server with the built-in certificate, as mentioned in [Section 3.1 – Certificates used in Excalibur](#).

FIDO could be implemented to add support for Roaming authenticators in either Excalibur Client, or Excalibur Dashboard, but FIDO2 does not add any value to the authentication flow. Excalibur is already validating many factors, see [Section 3.3 – Policies](#).

Retransmitting security keys

Security keys are often issued by other companies, not the customer. These keys are used to secure federated access from the customer network to the federated network. This means that certificate validation is done on the federated site and can't be done on the Excalibur Server. The typical use case is accessing the federated website from the customer network via Excalibur PAM. There are multiple ways how to authenticate clients in a federated environment:

HTTPS Client Authentication – This is just a special case of previously analyzed [Section 4.2](#). In this instance certificate validation cannot take place on the Excalibur

Server, e.g. because of company policies. Neither HTTP Proxy nor Virtual Browser is supporting this scenario. It is something we are investigating right now, but it has a low priority since we do not see any demand for this feature.

Client Authentication using a thick client. This approach needs some kind of client on the client device. Clients are often custom build for specific use cases. E.g. Slovak government website slovensko.sk is using thick client ³ based on solution from Thales Group [55]. Also, the National Bank of Slovakia (and some other European national banks) is using similar solution for securing access to the Alliance Web Platform [53]. This approach cannot work with the Virtual Browser, since all commands are executed in the Virtual Browser and the security key is plugged in the client device.

On the other hand, this approach can work with HTTP Proxy. HTTP Proxy is retransmitting all HTTP communication, so code is executed on the client device. However we cannot say HTTP Proxy is fully transparent for this approach since there can be, and often are, port forwarding techniques included. Port forwarding is done differently for different solutions, so necessary changes to the HTTP Proxy are deployment-specific.

U2F security keys can be used as an additional method of two-step verification to online services that support the U2F protocol, including Google, Azure [29], Dropbox [19], GitHub [56], GitLab [33], Bitbucket [22], Nextcloud [45], Facebook [20] and others. Again, HTTP Proxy should be transparent for this use case, but it will not work in the Virtual Browser. Virtual Browser could act as a U2F security key. This case was presented in the [Section 4.2](#). For usage with a real user security key, Virtual Browser would need to implement FIDO, which would retransmit FIDO communication from user browser to the Virtual Browser.

We did not encounter any enterprise software that would implement U2F protocol since developers can not specify which can or cannot be used for authentication. Securing major websites as GitHub or Azure with Excalibur PAM does not make sense, since it can be bypassed anytime. These websites can be reached from any device or network in the world, so it is impossible to forbid access to them (unless you live in China). Moreover using U2F is an indicator, that they are taking security seriously, which is especially true for major websites listed above.

Replacing Excalibur Authentication Flow with FIDO2

FIDO2 standard was introduced to enable users to easily authenticate to online services in both mobile and desktop environments [12]. This mechanism can be used to authenticate to the Excalibur. Replacing the Excalibur authentication flow for the FIDO2 flow would standardize the authentication process, but would also remove some of the advantages of the Excalibur authentication flow. How would this flow looked like is shown in [Figure 4.6](#).

One of the biggest advantages of the FIDO2 flow is that the user does not need any application. However, without the application running on the user phone we would lose the ability to store keys and crypto material. This would mean that no password can be reconstructed. There are cases, where this is not a problem, e.g. deployment, where the only way of authenticating is done using passwordless protocols such as SAML. Hopefully, we will do such deployment in the future but for now, it is only a dream.

³<https://techterms.com/definition/thickclient>

Login to Excalibur using FIDO2

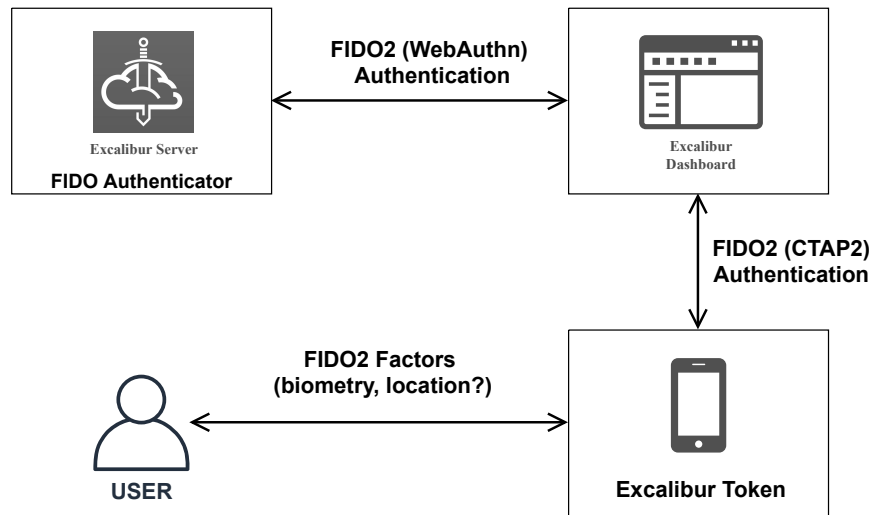


Figure 4.6: App-less Excalibur.

FIDO2 can be also used in the mobile application using various SDKs ⁴. Most FIDO2 implementations do not provide access to a low-level operation like signing or encrypting. The only methods that are usable by developers are those that are described in the [Section 2.6 – FIDO2](#). These methods do not provide any means for supporting encryption, which is vital for Excalibur.

FIDO2 standard introduces extensions, which could be used for encryption and some other functions (e.g. location). `hmac-secret` is an CTAP2 extension which could be used for retrieving a symmetric key for encryption ([Section 2.6 – Extensions](#)). Another extension what could be useful is `Location Extension (loc)`. These extensions are not well supported in neither browsers nor SDKs. As stated on the Mozilla Development Network website:

“As of June 2020, only `appid` is supported by [Chrome](#) and [Edge](#). Firefox does not seem to [support any extension](#). Also Chrome doesn’t plan to support any other extension in [future](#)” ⁵.

Another problem could be the usage of CTAP2 protocol. CTAP2 is used for communication between a computer and a mobile device, utilizing BLE, most probably. Most enterprise does not like solutions built around BLE, because of the manageability and security. Most desktop does not have Bluetooth transmitters out of the box. For this use case, a simple QR code shown on the display is a way better solution.

Key Points

FIDO2 standard is fighting the same war with similar tools as the Excalibur. They are both trying to remove passwords from our lives. Implementing FIDO2 to the existing Excalibur

⁴<https://fidoalliance.org/fido-certified-showcase/>

⁵<https://developer.mozilla.org/en-US/docs/Web/API/PublicKeyCredentialRequestOptions/extensions>

architecture does not make sense. At least, we did not find any use case, where it would be worth it, for now. FIDO2 could be taken as a direct alternative to the Excalibur system, more focused on personal accounts. It can be used by any user who wants to get rid of passwords and use the internet more securely, but it is not a very good option for enterprise solutions. The main reasons are summarized here:

- Weak `WebAuthn` support by the websites.
- Weak FIDO2 extension support by browsers.
- No low-level access to the cryptographic keys used for FIDO2.
- FIDO2 is focused on personal accounts, no way to control which authenticators can be used for authentication/registration.
- Excalibur scheme can do everything that FIDO2 scheme can and much more.
- No or very little added value for FIDO2 implementation.

FIDO is a solid standard, which can be utilized for many use cases. Cloudflare recently announced their plan to use FIDO as CAPTCHA replacement [28]. The flow is very similar to the classic FIDO login, but they are interested in the attestation signature (Section 2.6). However, the FIDO device biometric sensor can be easily deceived and even abused for an automatic attack, which was presented by Yuriy Ackermann [59].

4.3 Single sign-on (SSO) design

Users can authenticate to the Excalibur system from multiple clients. These clients are fundamentally different in platform, persistence, capabilities, etc. This section is written for the `Excalibur Server v3.5`. The ideal client has these properties:

- Can report that user logged in.
- Can report every user logout.
- Can log out the user.

This is true for **Windows Client**, but can not be said for any web client. **Dashboard**⁶ can report user log in, but can't always report logout, e.g. when a connection is ungracefully terminated. Dashboard communicates with the server using `REST API` and login tokens (`JWT`) are also stored on the server-side, so the user could be logged out.

PAM is a web client and behaves the same way as the Dashboard with regards to user sessions with one exception. PAM has a permanent `websocket` connection to the server, so it can be instantly terminated. The only exception is the **HTTP-PROXY**. It's based on `HTTP protocol` (no `websocket`), so it behaves the same way as Dashboard.

When the user logs in via **SAML** to some web applications, Excalibur has only login information. **Single Logout (SLO)** is a feature of the SAML, but only a small portion of applications are supporting it right now. Same as the SSO, SLO can be **SP-Initiated** or **IDP-Initiated** [52]. It can be used for both obtaining information about user logout and for terminating user sessions. SLO should be implemented in the Excalibur server but should be not relied upon, since not all SAML-enabled applications support SLO.

⁶Dashboard is also a client.

Sessions

User session in the **Excalibur Server v3.5** is based on JWT tokens. These tokens are also stored on the server-side and can contain information signed by multiple components. User can be in these states:

- Logged in.
- Logged in but lost access tokens (e.g. device restart) – valid tokens, but user got no access to them.
- Logs out, but a server does not know about it – e.g. lost connection to the internet.
- User logged out.

In which state the user is right now can be tricky to identify. Desktop clients can not be taken into account. When the user which is logged in the OS visits some web resource, e.g. Dashboard, we do not know which user it is. SSO could be determined only based on the web clients. These clients however cannot always report log in / log out events. There are really one 2 fundamental solutions to this problem:

- SSO based only on clients with a permanent connection.
- SSO based on the timeouts.

SSO based only on clients with permanent connection is more straightforward and easier to implement. We should know the user state at every moment. Except for some edge cases like dropped **websocket** connection, that will eventually be restored.

SSO based on the timeouts needs to be well designed. The first issue is to choose which clients will be taken into account and how will the timeouts work for each client. We can distinguish 3 types of clients:

- Web clients with permanent connection (**Excalibur PAM**). This is quite simple. The user is using a web client with a permanent connection – he is logged in.
- Web Client without the permanent connection (**Dashboard, HTTP Proxy**). Since we designed authentication scheme with **JWTs**, which are also stored on the backend, we can at any moment tell, if the given user has a valid token, thus is logged in.
- Web clients used to log in to the third-party application (**WebSDK, SAML**). When the user is logged in to the third-party application using **WebSDK** we cannot make any assumption if she/he is still logged in or no since third party application will typically not report that user logged out. **SAML** does have a mechanism for this, called Single Logout (SLO), but SP does not implement, nor use it. **SAML** is also giving us the option to limit the time for which he will be logged in to the SP, but this again could be ignored by the SP. The best way is to simply ignore this kind of client when determining if the user is logged in.

Chapter 5

Implementation

This chapter is focused purely on SAML implementation since we decided not to implement the FIDO2 protocol based on facts stated in [Section 4.2](#). SAML implementation is divided into:

- [SAML Server Component implementation](#) – responsible for validating and managing SAML service provider (SP) metadata, creating identity provider (IDP) metadata, consuming, validating, and generating SAML messages, i.e. everything SAML related.
- [Authentication Component](#) – `WebSDK` for this implementation, but can be and will be replaced by a new one for SAML implementation to the `Excalibur Server v3.5`. Authentication Component is adding Excalibur authentication to the SAML flow, or any other future integrations.

5.1 SAML Server Component

Since majority of Excalibur Server codebase is written in JavaScript and run by `nodejs`¹ SAML Component will be also written in JavaScript. As said in [Section 4.1 – SAML library](#), `samlify` [32] was used for implementation.

Excalibur Server v3 is using the concept of workers, so SAML Component is a single worker. There is also a possibility to spawn multiple instances of the same worker, so they should be *stateless*, even when this is used only for a few specific workers. More on server architecture in [Section 3.3 – Current State of the Excalibur Server](#).

SAML communication

SAML is an open standard for exchanging authentication and authorization data between parties, in particular, between an identity provider (IDP) and a service provider (SP). What is SAML and how it works is discussed in [Section 2.5 – SAML](#). This chapter is focused on the implementation of the solution proposed in [Section 4.1 – Excalibur – SAML integration design](#).

Right now only SP-initiated login is supported since it is a typical use case for most, if not all the users. IDP initiated flow is on the roadmap, but only for the `Excalibur Server v3.5`, because it does not make sense without fully functional SSO. How IDP initiated flow would look like is discussed at the end of this section.

¹<https://nodejs.org/en/>

This section is describing the authentication process as presented in [Section 4.1 – Excalibur – SAML integration design](#), but in more depth and with added details from implementation. Described flow is visualized in the [Figure 4.1](#), [Figure 4.2](#) or in the [Figure 2.4](#).

SP-initiated flow starts at the SP , where the user enters credentials or chooses that she/he wish to authenticate using Excalibur IDP. **SAML AuthnRequest** is generated and delivered to the IDP. Format and how it is delivered depends on used bindings, more in [Section 2.5 – SAML](#). As shown in example metadata [Listing 5.1](#), Excalibur IDP supports **POST** and **REDIRECT** binding and using different endpoints for both, so the SP needs to choose one of the bindings. `binding` is a attribute of the `SingleSignOn` element in the SAML metadata [38]. If the SP choose to use **POST** binding, it includes `AuthnRequest` in the HTTP POST form submitted to the `location` attribute of the selected `SingleSignOnService` element. On the other hand when SP choose to use **REDIRECT** binding, `AuthnRequest` is part of the `url` parameter to which is user redirected. In both cases `AuthnRequest` is base64 encoded.

Parsing of the SAML AuthnRequest is done by library function

`IdentityProvider.parseLoginRequest(sp, binding, request)`. First parameter is the SP object which is determined based on the `referer` parameter of the HTTP request. After successful parsing and validation of the `AuthnRequest`, `flow` object is created. This is a library-specific object that would be used further in the flow. It represents some kind of library state (information about SP, SAML request, etc.).

The next step is to **present the QR code** to the user. This is done by returning HTML page with `HTML form`, which includes hidden elements and `WebSDK` script. `WebSDK` can be replaced for another authentication component, but for SAML implementation for **Excalibur Server v3**, it was the only choice. It was also modified to support this use case, more in [Section 5.2 – Authentication Component](#).

Since our implementation should be stateless, the flow object is sent to the user’s browser. After authentication, it is sent back to the server. To make sure that `flow` object was not manipulated with, a signature of the object is also included. Before signing the `flow` object, a timestamp is added to prevent reuse of this object.

After user authenticates to the Excalibur , user authentication data with the `flow` object is send to the server via the `HTML form`. Server validates signatures for both user data and `flow` object and timestamp included in the `flow` object.

For **generating SAMLResponse** template is used. Templates grant us the freedom to choose which values are shared with the SP. This template is populated with data from the request and with user data. `samlify` [32] takes care of the signing and encryption of the elements based on the configuration.

SAMLResponse is then included in the `HTML form`, which is presented to the user. This form is auto-submitted to the assertion consumer service location of the SP. HTTP POST via `HTML form` is used for both implemented bindings. For a user, it looks like a simple redirect.

Using SAML all communication goes through the user’s browser, so IDP and SP don’t need to have visibility to each other. They can be in totally different networks, but users need to have access to both IDP and SP. This is true for both **POST** and **REDIRECT** binding.

IDP initiated flow would start in the **Excalibur Dashboard**, where a user needs to be already authenticated. Since right now we are using **WebSDK** client, it cannot utilize SSO, so a user would need to authenticate once more. IDP initiated flow is not supported by every SP and it cannot be read from metadata, so it would need to be configured manually, or it could confuse users. When the user is already logged in to the SP and starts a new session at the IDP it replaces the previous session. Moreover, IDP-initiated flow presents a security risk [3]. When flow starts at the IDP, SP receives unsolicited **SAMLResponse**. This response does not have **InResponseTo** parameter, so it can be reused. Assertions still include validity time, but it needs to be long enough, typically several minutes, so it mitigates attack surface.

Another SAML profile is **ECP (Enhanced Client or Proxy) Profile**. In contrast to the SAML Single Sign-On(SSO) profiles such as web-based SSO and Single Logout, the SAML ECP profile is related to Enhanced Clients and Proxies which have extended capabilities than a normal browser. Simply, an ECP may be a desktop application, a server-side code running in a web application, or a proxy server-WAP(Wireless Access Point) Gateway in front of a mobile device [58]. In practice, it means, that since the authentication is done on the server-side, the user cannot be shown the QR code. For this scenario Excalibur IDP can send push notifications instead of using **WebSDK** to show the QR code, however, the username needs to be specified beforehand authentication.

Push notifications are not using right now, since we did not encounter any application requiring ECP. All of the tested applications use standard web profiles.

5.1.1 Signing and Encrypting SAML Messages

SAML messages: **AuthRequest**, **Response**, **LogoutRequest** and **LogoutResponse** can be all signed. Signature is part of the **xml** document or added as another parameter, based on the binding. **REDIRECT binding** is embedding message in the URL parameters, which have limited length, so the signature algorithm and signature itself are sent in a separate parameter.

SAML Response can have signed individual assertions or the whole document can be signed. These modes can be combined with each other and moreover, assertions can also be encrypted. Which gives us these 8 variants ²:

- An unsigned SAML Response with an unsigned Assertion
- An unsigned SAML Response with a signed Assertion
- A signed SAML Response with an unsigned Assertion
- A signed SAML Response with a signed Assertion
- An unsigned SAML Response with an encrypted Assertion
- An unsigned SAML Response with an encrypted signed Assertion
- A signed SAML Response with an encrypted Assertion
- A signed SAML Response with an encrypted signed Assertion

²Examples of this messages can be also found here: https://www.samltool.com/generic_sso_res.php

SAML response is always sent to the assertion consumer service using POST binding, so a signature is always a part of the xml document.

Our implementation does not require the SP to sign `AuthnRequest`, because of the simplicity and usability. Validating signatures requires IDP to trust certificates used for signing, which means they would need to be imported to the Excalibur since many SPs use untrusted certificates. Excalibur, on the other hand, is always signing both the assertions and the messages as a whole.

Encryption is also implemented but was not fully tested, since there is no need for encryption. All the information included in the message is public and none of the tested service providers(Section 6.1) required this functionality. Most of them didn't even have options for encryption.

Schema validation

Since SAML is XML based protocol, every message, even metadata, are a XML documents that can be validated against XSD scheme. `samlify` [32] library requires developers to setup schema validation. The developer can choose which validator with which schema will be used. `samlify` library author prepared 3 packages for XSD scheme validation. Each one is based on the different package for scheme validation, but all of them have schemes for SAML included:

- [@authenio/samlify-xsd-schema-validator](#)
- [@authenio/samlify-validate-with-xmllint](#)
- [@authenio/samlify-node-xmllint](#)
- [@authenio/samlify-libxml-xsd](#)

Schemes are from official OASIS documentation ³ and are saved locally in the package. My implementation is using `@authenio/samlify-validate-with-xmllint` package for scheme validation, but others should work as well. Validator options from author of the `samlify` library do not validate metadata ⁴, but this feature is on the roadmap.

Proxy cooperation

SAML Component was built with proxy integration in mind. Virtual Browser and HTTP-Proxy are both capable of authenticating users via the SAML protocol. Excalibur acts as the identity provider (IDP), so authentication can be unified across all service providers (SP) providing a secure and seamless Excalibur authentication experience. Virtual Browser provides total isolation, so SAML messages are never sent to the user's browser. SAML redirects users from the service provider URL to the identity provider (Excalibur) site, which in HTTP-Proxy context means that for the time of authentication the user is not connected through HTTP-Proxy. After successful authentication, the user is redirected back to the same HTTP-Proxy session.

SAML Component can detect if the request came from the Excalibur proxy based on the URL. If the URL matches the URL used by the proxy, it is saved together with the `flow` object to the user's browser. After successful authentication `SAMLResponse` should be send to the

³<https://docs.oasis-open.org/security/saml/v2.0/>

⁴<https://github.com/tngan/samlify/issues/371>

AssertionConsumerService location URL. Since the proxy is working by prepending the Excalibur server URL to the actual URL, URL of the assertion consumer service is modified that the user is returned to the previous proxy session. Diagram showing SAML and Proxy cooperation is shown in [Figure 5.1](#), which is fairly similar to the standard SAML flow shown in the [Figure 4.1](#).

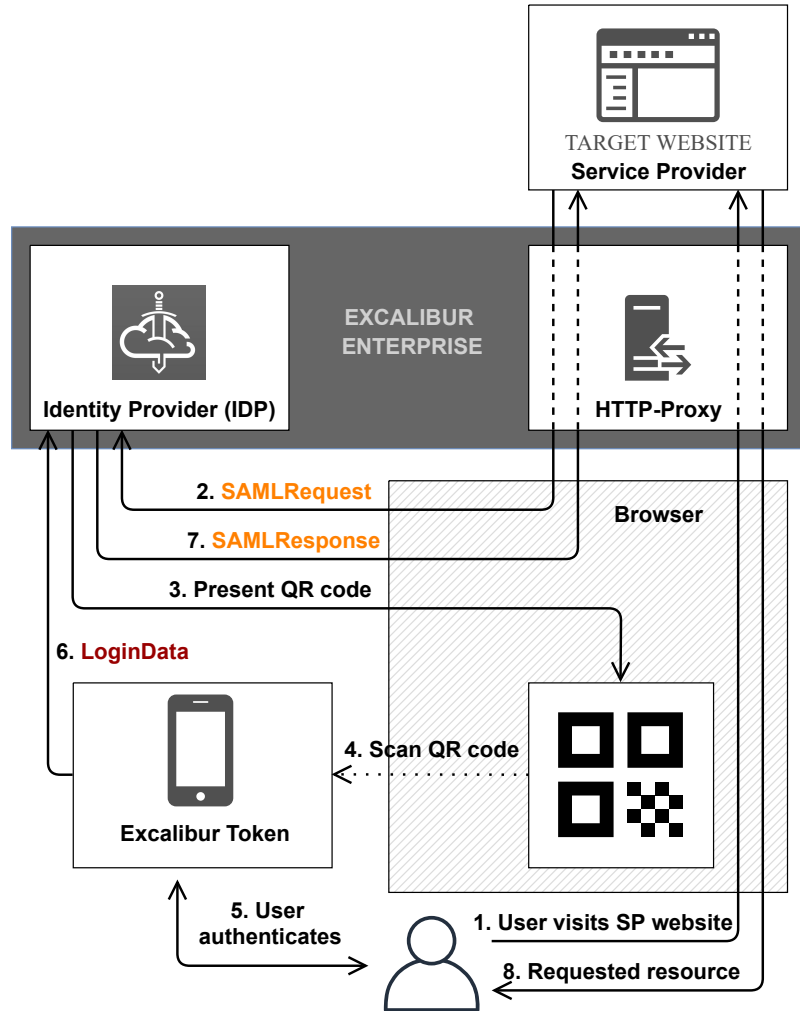


Figure 5.1: Excalibur SAML authentication in proxy session.

Metadata management

SAML provider (SP or IDP) needs to have established trust before any communication takes place. Trust is established by import SAML metadata of each provider [38]. This subsection is divided into **IDP Metadata** and **SP Metadata**. **IDP Metadata** section discuss metadata generation and export. **SP Metadata** section discuss SP metadata import and management process.

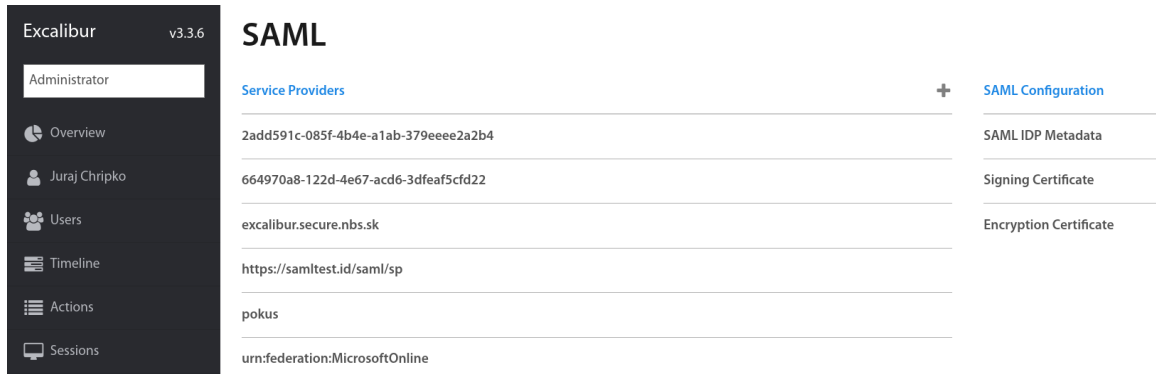


Figure 5.2: Excalibur Dashboard SAML management section.
Image was scaled for better readability.

IDP Metadata

For generating Excalibur IDP Metadata library functions were used. At the start of the SAML worker, `IdentityProvider` object is created. `IdentityProvider` object can be populated with complete metadata file, or with specific values like `id`, `signingCert`, `singleSignOnService` and so on [32]. When metadata file is used for initialization, no options can be changed after that.

We decided to initialize `IdentityProvider` object with values from `config` file, so values can be different for different deployment. Metadata are generated using `IdentityProvider.getMetadata()` function, so metadata generation is solely `samlify` library responsibility. And since there is no option to specify metadata template or manipulate with `xml` elements, generated IDP metadata are not perfect, more in [Table 6.1](#).

IDP Metadata can be exported in several ways:

- `<xclbr-hostname>/saml/metadata` url is a public link for showing metadata.
- `<xclbr-hostname>/saml/metadata.xml` url is a public link for downloading metadata file.
- Button in Dashboard ⁵ can also be used to download metadata file. This button is also shown in the [Figure 5.2](#).

⁵<https://docs.xclbr.com/v3/integrations/excalibur-saml-integration-manual/#get-excalibur-idp-metadata>

```

<EntityDescriptor entityID="xclbr.com">
  <IDPSSODescriptor
    WantAuthnRequestsSigned="false"
    protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">
    <KeyDescriptor use="signing">
      <ds:KeyInfo>
        <ds:X509Data>
          <ds:X509Certificate>
            MII...
          </ds:X509Certificate>
        </ds:X509Data>
      </ds:KeyInfo>
    </KeyDescriptor>
    <NameIDFormat>
      urn:oasis:names:tc:SAML:2.0:nameid-format:persistent
    </NameIDFormat>
    <NameIDFormat>
      urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress
    </NameIDFormat>
    <NameIDFormat>
      urn:oasis:names:tc:SAML:2.0:nameid-format:transient
    </NameIDFormat>
    <SingleSignOnService
      Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect"
      Location="https://xclbr.com/saml/login"/>
    <SingleSignOnService
      Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"
      Location="https://xclbr.com/saml/login-post"/>
    <SingleLogoutService
      Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect"
      Location="https://xclbr.com/saml/logout"/>
    </IDPSSODescriptor>
  </EntityDescriptor>

```

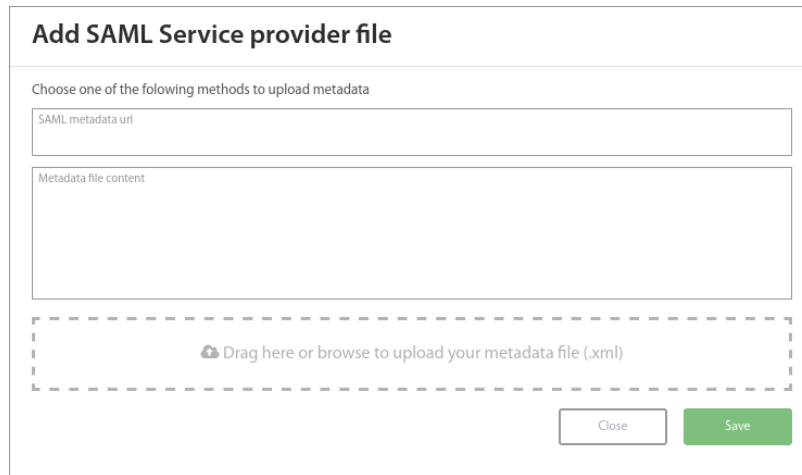
Listing 5.1: Excalibur IDP metadata example

SP Metadata

Each SP that will be used with Excalibur IDP needs to have imported metadata in the Excalibur Dashboard. SP in Excalibur IDP is represented solely by metadata and they need to be persistent. The only way how this could be done in the current version of Excalibur is to save metadata files on disk. Each time SAML Component starts it find all metadata and creates `ServiceProvider` object from `samlify` library [32]. Therefore data are stored as the SAML metadata. For managing service providers user interface was implemented, it is shown in [Figure 5.2](#). Basic CRUD operations are possible, all of them using SAML section in the Excalibur Dashboard. Only admin of the Excalibur Server can manage service providers.

Update of the metadata is done the same way as adding a new SP. Service providers are identified by the ID parameter of the metadata. If metadata with the same ID are already imported in the **Excalibur Dashboard**, a user is asked if he wants to replace metadata [11].

Dialog for adding SP Metadata is shown in **Figure 5.3**.



The image shows a web-based dialog box titled "Add SAML Service provider file". The dialog contains the following elements:

- A title bar with the text "Add SAML Service provider file".
- A subtitle: "Choose one of the following methods to upload metadata".
- A text input field labeled "SAML metadata url".
- A larger text area labeled "Metadata file content".
- A dashed rectangular box for file upload, containing a cloud icon and the text "Drag here or browse to upload your metadata file (.xml)".
- At the bottom right, there are two buttons: a white "Close" button and a green "Save" button.

Figure 5.3: Dialog for adding SP to the Excalibur IDP.

5.2 Authentication Component

In implementation presented in this thesis, **WebSDK** component, **Section 3.1 – Excalibur Clients**, is used for actual user authentication to the **Excalibur Server**. The solution was implemented according to the design presented in the **Section 4.1 – Excalibur – SAML integration design**.

Before **WebSDK** can be used for SAML, it needs to be configured. This configuration is typically done by the Administrator of the Excalibur system, but for the SAML, it is done automatically. SAML Component at its start, checks if the **WebSDK** component for SAML is created and when it is not, it creates it. This is done by inserting values into the database. More on the **WebSDK** can be found in **Section 3.1 – Excalibur Clients**.

There are multiple types **WebSDK** component, SAML is one used for . . . , well . . . , SAML. SAML type of the **WebSDK** have modified action after user login. **WebSDK** automatically enters values to the **HTML form** and sends it after successful use authentication. SAML type **WebSDK** is also signing data send to the user's browser, signature is included in the **password** entry of the **userData**. SAML component is validating this signature at the last step of the algorithm presented in the **Figure 5.4**.

One of the responsibilities of the authentication component is also to report user's actions, authentication in particular.

New Authentication Component

Excalibur Server v3.5 will have new authentication component, which will replace **WebSDK** for this use case, as presented in the **Section 4.1 – Server-side Authentication Component**. A new server-side authentication component could not be implemented because **Excalibur Server v3.5** is not ready for this kind of integration, more in **Section 3.3 – Current State of the Excalibur Server**. The new authentication component will be communicating with

the `Core` service, which will provide SSO capabilities. This server-side component will also need to implement the communication with the server. WebSDK is now using HTTP long polling [15], but there are newer technologies like `websocket` available. However `websocket` is not a clear winner, even Google is still using HTTP long polling for some use cases [23].

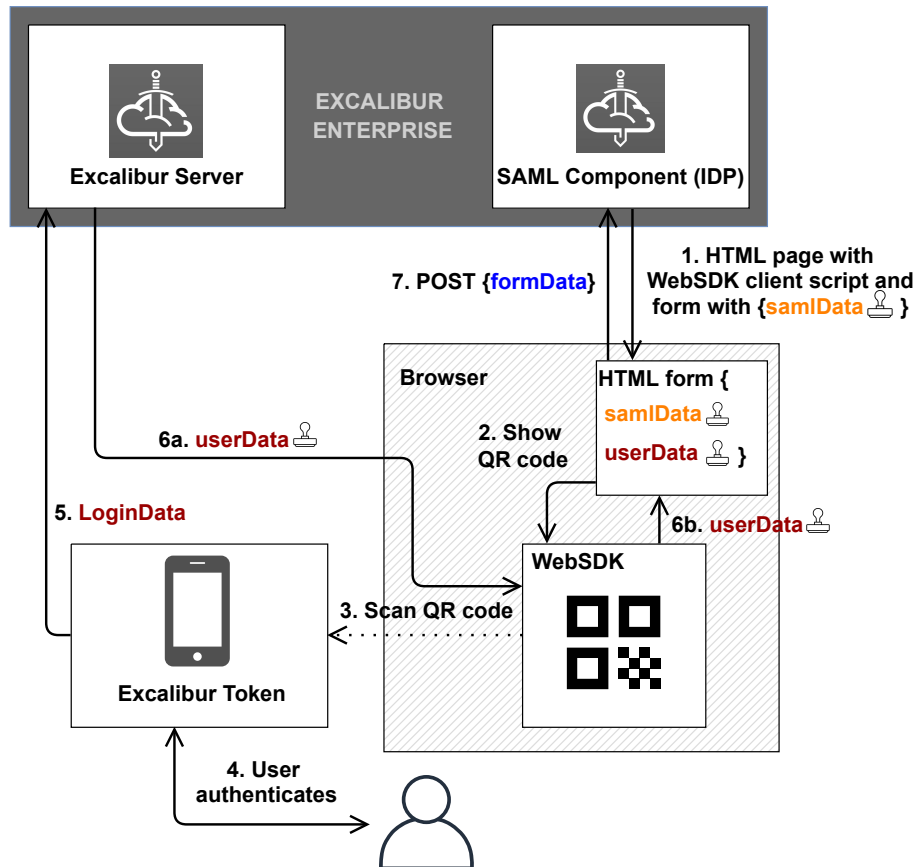


Figure 5.4: Excalibur Authentication to the SAML Component using WebSDK.

5.3 Single sign-on (SSO)

SSO was not implemented because of the current state of the `Excalibur Server`. `Excalibur Server v3.5` is not ready for implementation of this feature and `Excalibur Server v3` does not have architecture capable of doing so.

It is also advised to monitor how different third-party applications act when SAML is used for logging in. Which of the applications honor the maximum length of the session (`SessionNotOnOrAfter` SAML parameter), which support SLO and which are reporting log out events of the user. These events could be simple HTTP call, not just SAML messages.

Chapter 6

Testing and Documentation

Since none of the FIDO protocols have been implemented during this project, this chapter is focused mainly on the SAML implementation. Last entry in the thesis assignment states: “Test correct functionality of implemented solution such as correct behavior from a user point of view, soundness of access control mechanism, and access revocation.”

Access control is a complete responsibility of the Excalibur system, specifically the authentication component, that is `WebSDK`, in this implementation. Access revocation is also an Excalibur system responsibility since access is allowed based on the configured Excalibur policies. To achieve correct behavior from a user point of view, multiple requirements needs to be fulfilled:

- IDP configuration – exporting IDP metadata and importing SP metadata
- SP configuration – exporting SP metadata and importing IDP metadata
- Setting up valid Excalibur policies (default policy will work)
- Some service providers requires additional user configuration, such as creating specific users for SAML authentication, more in [Section 6.2 – User Management](#)

This chapter includes information about testing the actual SAML communication – [Section 6.1 – Functional Testing](#) with the [list of service providers](#) which were used for testing. Functional testing also includes schema validation, [security aspect](#) of the implementation and even [performance testing](#). The second subsection is dedicated to [the documentation](#) and [the user management](#). Actual documentation can be found on our public wiki [11], specifically in the integrations [6]. Documentation includes manuals for managing and configuring Excalibur IDP and configuration manuals for third-party applications, which were used as SP during functional testing. A brief technical API documentation was also created and is in [Appendix A](#). Configuration manual includes basic CRUD ¹ operations on SAML service providers.

6.1 Functional Testing

Testing any software usually starts by testing a smaller part possible, a unit [5]. In our case, the smallest part is a `samlify` library, which already comes with tests, so we need to test how it is used. Because `Excalibur Server v3` have no capabilities to test whole

¹Create, Read, Update, Delete

flows, since they are typically quite complex and flowing through multiple components, so also multiples platforms. Moreover, these tests need user authentication on the Token. Excalibur Server v3.5 was designed to support these complex tests using stubs and improved architecture. All the tests described in this section are done manually, so they are done to describe what the system does, i.e. they are testing functionality [5].

Tests start by initiating login at the SP, diagram of the SAML flow is shown in [Figure 2.4](#), [Figure 4.1](#) or in [Figure 4.2](#). SP generates `SAML AuthnRequest`, which can be observed using `SAML-tracer` browser extension ².

IDP needs to validate `AuthnRequest`, show the QR code for the user and after successful user authentication it needs to validate incoming data, as discussed in the [Section 5.1 – SAML communication](#) and in the [Section 5.2 – Authentication Component](#). When everything is valid, IDP generates `SAML Response` based on the user data and the flow object created after validating `AuthnRequest`. This `Response` is then send to the SP and is also observed using `SAML-tracer`. `Response` need to pass these checks:

- Scheme validation against XSD (XML Schema Definition). Unlike in the [Section 5.1.1 – Schema validation](#) where all incoming messages were validated against the scheme, this chapter describes how outgoing messages were validated since `samlify` is not validating every message it generates. SAML is an XML-based markup language for security assertions, so every SAML message including metadata can be validated against XSD. OneLogin SAML Developer Tools [40] were used for schema, data, and signature validation.
- Data validation against `AuthnRequest`. Schema validates only data structure, not the actual data, so the next step was to validate if the metadata and message contain correct values. `Response` needs to contain number of values based on the `AuthnRequest` and IDP Metadata. These values needed to be entered manually to the validation tool ³.
- Signature validation. Excalibur IDP is by default signing both assertion and message as a whole. Certificates used for testing were issued by the Excalibur CA, so they appeared as untrusted to the other parties. They needed to be manually entered into the validation tool [40].
- User login to the SP. Even when `Response` is valid, SP needs data about the user in a specific format, so not every valid response will end with a logged-in user.

SP metadata needed to be already imported into the Excalibur IDP, but that is discussed in [Section 6.1 – Service Providers Management testing](#).

SAML is quite a big standard with lots of nuances since many features are optional, so each implementation of SAML SP can behave a little differently in various scenarios. Excalibur is an identity provider, so the only SAML messages that it generates are Metadata, SAML Response, and Logout Response. Results from scheme and data validation are in [Table 6.1](#)

Generated metadata does not pass validation against the scheme, as seen in [Table 6.1](#). Wrong element order is the cause. `SingleLogoutService` element must go before `NameIDFormat` element [38]. Simply moving this tag will resolve the problem, but since `samlify` [32] does not have any options to do this, the issue was created on GitHub ⁴.

²<https://addons.mozilla.org/en-US/firefox/addon/saml-tracer/>

³https://www.samltool.com/validate_response.php

⁴<https://github.com/tngan/samlify/issues/429>

Table 6.1: Table showing validation test results.

	Schema Validation	Data Validation
Metadata	×	✓
SAML Response	✓	✓
Logout Response	✓	✓

Testing on Service Providers

Metadata were imported to all of the tested SP and most of them accepted them without any errors. For the SAMLtest.ID metadata element order needed to be manually changed before importing metadata. Actual configuration and log-in were tested against multiple service providers. Some of them are just test applications, which let the developer inspect received assertions and even logs for better debugging. The others are mostly enterprise applications and Identity Access Management (IAM) solutions.

List of tested service providers:

- Fortinet Fortigate ⁵
- Pulse Secure ⁶
- Cisco ASA ⁷
- Alliance Web Platform (SWIFT) ⁸
- RSA SAML Test Service Provider configuration ⁹
- SAMLtest.ID by Signet ¹⁰
- Office 365 ¹¹

Security

SAML is a well-known standard used by a lot of big companies, so it is also a tempting target of cyberattacks. Security evaluation was also part of the testing.

Main source of information were OWASP guidelines [25] and `samlify` GitHub issues ¹². One of the attacks pointed out by OWASP guidelines was the signature wrapping attack. `samlify` was indeed vulnerable to this attack, but after notifying the maintainer of the library vulnerability was fixed [9].

At the time of writing this thesis `samlify` is using vulnerable version of `xmlDOM` library [31]. Issue was raised on the `samlify` GitHub ¹³, but fix was not ready even after 2 months. This vulnerability is also reported from `npm` when installing `samlify` library.

⁵<https://www.fortinet.com/>

⁶<https://www.pulsesecure.net/>

⁷<https://www.cisco.com/c/en/us/products/security/adaptive-security-appliance-asa-software/index.html>

⁸<https://www.swift.com/our-solutions/interfaces-and-integration/alliance-web-platform-se>

⁹<https://sptest.iamshowcase.com>

¹⁰<https://samltest.id/>

¹¹<https://www.office.com/>

¹²<https://github.com/tngan/samlify/issues>

¹³<https://github.com/tngan/samlify/issues/416>

SAML is a complex standard with lots of nuances and it can be quite tricky to implement securely. `samlify` [32] is maintained by one person and most likely only in his free time. Together with the other problems encountered during testing, such as metadata element order problems, or missing validation of the metadata, `samlify` is not an ideal choice for our purposes, but it most definitely was enough for proof of concept SAML IDP implementation.

Performance

Basic load testing was conducted on the SAML Component in order to find out how many users can be served at a given time period. SAML could be also used as an endpoint for Denial-of-Service (DOS) attack since it can be accessed without authorization. `WebSDK` acts as a rate limiter since the user needs to scan the QR code and do the factor verification on the `Token` [27]. 3 cases were tested:

- `saml/metadata` URL – returns metadata as XML document, which should be in memory, so it is only a static website without any prior validation.
- `saml/login` URL with valid arguments – `SAMLRequest` is validated, SP metadata are loaded and static website with `WebSDK` is returned.
- `saml/login` URL with malformed arguments – `SAMLRequest` is malformed, so error should be returned.

All the tests were done by the `ab` (Apache HTTP server benchmarking tool) ¹⁴. 500 requests were done for each URL with 10 concurrent connections at the time and the `Excalibur` server was restarted between tests, so there were no residuals left from previous tests. The next table (Table 6.2) is showing how many concurrent requests can be answered within it one second and how long it takes to answer one request.

Table 6.2: Table showing load test results.

	<code>saml/metadata</code>	valid <code>AuthnRequest</code>	malformed <code>AuthnRequest</code>
Requests per second [# /sec]	42.45	8.79	13.57
Mean time per request [ms]	235.549	1137.426	736.867
Mean time per request [ms] <i>across all concurrent requests</i>	23.555	113.743	73.687

CPU utilization of the SAML worker was over 95% when testing login endpoint with both valid and malformed `AuthnRequest`. RAM utilization started to rise, which even led to worker restart after a few thousand requests. Testing showed, that this implementation can serve up to 10 concurrent requests per second for a short period of time (few minutes). Additional defense against DOS attack should be implemented before production use.

Another limiting factor can be the authentication component. Every entity of the `WebSDK` needs to have opened HTTPS connection for the server, where user results are sent. There is a deployment with around 800 concurrent `WebSDK` sessions and SAML is a potential replacement for `WebSDK`. For SAML to be able to withstand such load, a new authentication component is needed.

¹⁴<https://httpd.apache.org/docs/2.4/programs/ab.html>

Service Providers Management testing

Service providers are added as metadata files via Excalibur Dashboard. When new metadata are added, IDP tries to construct `ServiceProvider` object and when it is built successfully, ID of the SP is compared to the other IDs. If the same ID is found, a user is asked if he wishes to replace SP. As presented in the [Section 5.1.1 – Schema validation](#), `samlify` is not validating metadata right now, so even when SP is added successfully, it does not mean that metadata are correct.

SP Metadata can be added by 3 options: metadata URL, metadata file content, or by uploading metadata file. All of these options were tested against all the service providers, [Section 6.1](#).

6.2 Documentation

Documentation was made during SAML functionality testing, where all of the service providers, [Section 6.1 – Testing on Service Providers](#), needed to be configured. Since only the SAML part was implemented, documentation includes a manual for configuring Excalibur SAML IDP via Dashboard and configuration manuals for applications used during testing [6]. SAML Documentation published on the Excalibur public wiki [11]. This chapter also includes a user management section, since every application takes a slightly different approach regarding user identifiers and user management in general.

User Management

In SAML terminology, Excalibur is an identity provider, meaning, it should provide identities to other services, known as service providers. Correct user mapping a key for correct behavior from a user point of view. User mapping is a method, where user identification used in the `SAML Response` is mapped to the service provider's entity. The most application creates new user when new ID is used, but some require the user to be created beforehand. `SAML Response` contains `NameID` element, which is typically used for user identification, but any other claim can be used. Excalibur IDP is sending 3 values: mail, whole name, and username. However, the real claim count is greater, since various formats are used, especially for the mail. Mail is also used as the `NameID`. `NameID` can also have different formats and they can be required from the SP. Excalibur IDP supports these `NameID` formats:

- `urn:oasis:names:tc:SAML:2.0:nameid-format:persistent` – Should be the same `NameID` for the same user, but also anonymous. Hash function could be used to produce transient `NameID`, but right now mail is used to ease the configuration.
- `urn:oasis:names:tc:SAML:2.0:nameid-format:transient` – New anonymous `NameID` for every user. Excalibur IDP is sending random string when transient `NameID` format is requested in the `AuthnRequest`.
- `urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress` – used by default, typically gained from the mail attribute from the Active Directory.

Another SAML attribute used by the SP is authentication context (`AuthnContext` [35]). This element is used by the SP to determine which authentication mechanism was used. SP can optionally specify which authentication context is required in the SAML request using `RequestedAuthnContext` parameter. Example are:

- `urn:oasis:names:tc:SAML:2.0:ac:classes>PasswordProtectedTransport` – the user should be authenticated through login/password, protected by SSL/TLS.
- `urn:oasis:names:tc:SAML:2.0:ac:classes:MobileTwoFactorContract` – used by default by the Excalibur IDP.

There are plenty of other authentication contexts and more can be made using schema [35]. Different authentication contexts can be compared using a set of rules [37]. Some service providers have options to set required authentication context, some have supported contexts specified in the documentation, others seem like they do not care.

If the SP specifies the required authentication context, Excalibur will use that specified context in the SAML response, since we believe, that Excalibur authentication supersedes most, if not all, typical authentication mechanisms. When the SP does not specify authentication context, `MobileTwoFactorContract` is used since it best represents actual authentication. During testing with all the service providers (Section 6.1), this does not cause any trouble, but there could be SP that requires another authentication context than `MobileTwoFactorContract` but does not specify it in SAML request, which will lead to unsuccessful login.

Chapter 7

Conclusion

Excalibur acts as a security token for passwordless authentication using your mobile phone to verify authentication factors such as location, PIN, fingerprint, Face ID, etc. For now, Excalibur used its distributed crypto scheme to log in to the operation system, Windows specifically. Excalibur is expanding to the web, so new integrations with web-based passwordless standards are needed. The goal of this thesis is to design and implement integration with FIDO2 or SAML passwordless authentication standards.

FIDO2 is a standalone standard for authentication on the web, that can be integrated into the Excalibur system in several ways to provide various functionality. After a detailed analysis of various use cases, it was clear, that FIDO2 is more like an Excalibur alternative than anything else. The most promising use case is to use FIDO2 authentication for Excalibur login, i.e. replacing Excalibur authentication with FIDO2 authentication. Using FIDO2 authentication to the Excalibur would not require an application, however, it cannot be done since FIDO2 lacks capabilities such as encryption support or location reporting. FIDO2 can also be used in smartphone applications, which would allow the application to use location or encrypt messages, but the library implementations are not mature enough. FIDO2 standard defines a variety of extensions and some of them can be used with Excalibur, but additional extensions are needed for a complete replacement of the Excalibur authentication mechanism. Moreover, an overall lack of support is the major problem with the FIDO2 standard. There is no point in integrating FIDO2 into the Excalibur system, but the integrations case study described in this thesis can be used in the future. FIDO2 is a well-designed standard, that could change how we authenticate with our personal accounts in the future, but its adoption is quite slow.

On the other hand, SAML protocol enabled us to do exactly what was the point of this project – use multi-factor passwordless authentication to third-party web applications. For SAML functionality, `samlify` [32] library was used. It worked fine for proof-of-concept implementation, which is currently installed in some of our partner’s deployments. Although, after fully testing this solution, we found out, that it lacks capabilities and even exposes security vulnerabilities. Capabilities can be improved by the library author or us, but for future use, a well tested and maintained library will be needed. For actual authentication a modified **WebSDK Component** is used. This component was not designed for this use case, so a new Authentication Component will be developed for future versions.

Single sign-on (SSO) capabilities were also discussed in this thesis but were not implemented, since **Excalibur Server** does not have infrastructure capable of utilizing SSO, for now. This will change with the deployment of the **Excalibur Server v3.5**, which is not yet ready for this type of integration.

The implemented solution is capable of communication with major SAML service providers, such as Office 365, Pulse Secure, Fortinet Fortigate, CISCO ASA, proving its functionality. During functionality testing with various service providers, configuration manuals for each service providers were made. Documentation also contains configuration manuals for Excalibur IDP as well as notes about the specific behaviour of some service providers and is publicly available [6].

Integration with other service providers will follow as we will continue to deploy SAML to our partners. The current version is proving that SAML integration is doable and even desired since SAML is well supported in the enterprise environment. Major improvements in form of the new authentication component, single sign-on integration, and other security and functionality fixes are needed before production deployment. Testing with other SAML libraries is recommended since the chosen library lacks capabilities and security vulnerabilities are not patched soon enough. SAML Component was developed for the **Excalibur Server v3** and since **Excalibur Server v3.5** is using new architecture, more work is needed to rewrite the SAML component for the new version of the **Excalibur Server**.

Similar to fighting the global pandemic, there is no silver bullet solution in the fight for cybersecurity. More layered defense is the only viable defense.

Bibliography

- [1] BASSETT, G., HYLENDERAND, C. D., LANGLOIS, P., PINTO, A. and WIDUP, S. *Verizon: 2021 Data Breach Investigations Report* [online]. 2021 [cit. 8. May 2021]. Available at: <https://enterprise.verizon.com/resources/reports/2021-data-breach-investigations-report.pdf>.
- [2] BEDNAREK, A. *Password Managers: Under the Hood of Secrets Management* [online]. Feb 2019 [cit. 14. May 2021]. Available at: <https://www.ise.io/casestudies/password-manager-hacking/>.
- [3] BRADY, S. *The Dangers of SAML IdP-Initiated SSO* [online]. June 2019 [cit. 7. January 2021]. Available at: <https://www.identityserver.com/articles/the-dangers-of-saml-idp-initiated-sso>.
- [4] BRAND, C., EHRENSVÄRD, J., JONES, M. B., LINDEMANN, R., KUMAR, A. et al. *Client to Authenticator Protocol (CTAP)*. Proposed Standard. W3C, january 2019. Available at: <https://fidoalliance.org/specs/fido-v2.0-ps-20190130/fido-client-to-authenticator-protocol-v2.0-ps-20190130.html>.
- [5] CHRIPKO, J. *Automatizované testování systému Fitcrack*. Brno, CZ, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Available at: <https://www.fit.vut.cz/study/thesis/20498/>.
- [6] CHRIPKO, J. *Excalibur SAML integration Manual* [online]. 2021 [cit. 28. April 2021]. Available at: <https://docs.xclbr.com/v3/integrations/excalibur-saml-integration-manual/>.
- [7] DECARLO, M. *Remote Work in 2019: Facts, Figures, Tips and Anecdotes* [online]. May 2019 [cit. 8. May 2021]. Available at: <https://getvoip.com/blog/2019/05/06/remote-work-in-2019-facts-figures-tips-and-anecdotes/>.
- [8] DIERKS, T. and RESCORLA, E. *The Transport Layer Security (TLS) Protocol Version 1.2* [Internet Requests for Comments]. RFC 5246. RFC Editor, August 2008. Section 7.4.6. Client Certificate. Available at: <https://tools.ietf.org/html/rfc5246#section-7.4.6>.
- [9] ERLEND OFTEDAL (WEBTONULL). *Samlify is vulnerable to signature wrapping* [online]. May 2018 [cit. 8. May 2021]. Available at: <https://hackerone.com/reports/356284>.
- [10] EXCALIBUR. *Excalibur Whitepaper* [online]. 2018 [cit. 7. December 2020]. Available at: <https://docs.google.com/document/d/1BFMoiEhGkLQJG1jzP-fQ1pEmLhjt4-YeYlyXpdiByeQ>.

- [11] EXCALIBUR. *Excalibur Documentation* [online]. 2020 [cit. 28. April 2021]. Available at: <https://docs.xclbr.com>.
- [12] *FIDO2: WebAuthn & CTAP* [online]. [cit. 28. April 2021]. Available at: <https://fidoalliance.org/fido2/>.
- [13] FORRESTER, N. *Case study: 40% of password managers vulnerable to breach* [online]. May 2020 [cit. 14. May 2021]. Available at: <https://securitybrief.asia/story/case-study-40-of-password-managers-vulnerable-to-breach>.
- [14] FREDERICO, H. Understanding FIDO Standards: Your Go-To Guide. *Okta Security*. January 2019. Available at: <https://www.okta.com/blog/2019/01/understanding-fido-standards-your-go-to-guide/>.
- [15] GANESAN, B. *Polling vs SSE vs WebSocket – How to choose the right one* [online]. July 2018 [cit. 28. April 2021]. Available at: <https://codeburst.io/polling-vs-sse-vs-websocket-how-to-choose-the-right-one-1859e4e13bd9>.
- [16] GRAY, A. *Interprocess Communication in Linux*. Prentice Hall Professional Technical Reference, 2002. ISBN 0130460427.
- [17] HARN, L., HUANG, D. and LAIH, C. Password authentication using public-key cryptography. *Computers & Mathematics with Applications*. 1989, vol. 18, no. 12, p. 1001 – 1017. DOI: [https://doi.org/10.1016/0898-1221\(89\)90028-X](https://doi.org/10.1016/0898-1221(89)90028-X). ISSN 0898-1221. Available at: <http://www.sciencedirect.com/science/article/pii/089812218990028X>.
- [18] HARNIK, D., KILIAN, J., NAOR, M., REINGOLD, O. and ROSEN, A. On Robust Combiners for Oblivious Transfer and Other Primitives. In: CRAMER, R., ed. *Advances in Cryptology – EUROCRYPT 2005*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, p. 96–113. ISBN 978-3-540-32055-5.
- [19] HEIM, P. and PATEL, J. *Introducing U2F support for secure authentication* [online]. August 2015 [cit. 26. April 2021]. Available at: <https://blog.dropbox.com/topics/product/u2f-security-keys>.
- [20] HILL, B. *Security Key for safer logins with a touch* [online]. January 2017 [cit. 26. April 2021]. Available at: <https://www.facebook.com/notes/10157814544346886/>.
- [21] JONES, M., LINDEMANN, R., KUMAR, A., HODGES, J., JONES, J. et al. *Web Authentication: An API for accessing Public Key Credentials Level 1*. W3C Recommendation. W3C, March 2019. Available at: <https://www.w3.org/TR/2019/REC-webauthn-1-20190304/>.
- [22] KELLS, T. *Universal 2nd Factor (U2F) now supported in Bitbucket Cloud* [online]. June 2016 [cit. 26. April 2021]. Available at: <https://bitbucket.org/blog/universal-2nd-factor>.
- [23] KILBRIDE SINGH, K. *Google: Polling Like It's the 90s* [online]. October 2019 [cit. 28. April 2021]. Available at: <https://dzone.com/articles/google-polling-like-its-the-90s>.

- [24] MALER, E. *Minutes of 9 January 2001 Security Services TC telecon* [online]. January 2001 [cit. 28. April 2021]. Security-services at oasis-open (Mailing list). Available at: <https://lists.oasis-open.org/archives/security-services/200101/msg00014.html>.
- [25] MANICO, J. and J., M. *OWASP Cheat Sheets* [online]. [cit. 5. May 2021]. Available at: <https://cheatsheetseries.owasp.org/index.html>.
- [26] MBANASO, UCHE and COOPER. Privacy Enhancement Technologies in Access Control. *IRIS Postgraduate Journal*. october 2005, p. 7–15.
- [27] MEHRA, M., AGARWAL, M., PAWAR, R. and SHAH, D. Mitigating denial of service attack using CAPTCHA mechanism. In: . January 2011, p. 284–287. DOI: 10.1145/1980022.1980086. ISBN 978-1-4503-0449-8.
- [28] MEUNIER, T. *Humanity wastes about 500 years per day on CAPTCHAs. It's time to end this madness* [online]. May 2021 [cit. 14. May 2021]. Available at: <https://blog.cloudflare.com/introducing-cryptographic-attestation-of-personhood/>.
- [29] *Passwordless authentication options for Azure Active Directory* [online]. February 2021 [cit. 26. April 2021]. Available at: <https://docs.microsoft.com/en-us/azure/active-directory/authentication/concept-authentication-passwordless#fido2-security-keys>.
- [30] MUNROE, R. *Password Strength* [online]. August 2011 [cit. 7. January 2021]. Available at: <http://www.xkcd.com/936>.
- [31] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. *CVE-2014-0160* [online]. March 2021 [cit. 8. May 2021]. Available at: <https://nvd.nist.gov/vuln/detail/CVE-2021-21366>.
- [32] NGAN, T. *SAMLIFY – Node.js SAML2 API* [online]. [cit. 28. April 2021]. Available at: <https://samlify.js.org/#/>.
- [33] NWAIGWE, A. *Support for Universal 2nd Factor Authentication* [online]. June 2016 [cit. 26. April 2021]. Available at: <https://about.gitlab.com/blog/2016/06/22/gitlab-adds-support-for-u2f/>.
- [34] *Security Assertion Markup Language (SAML) v2.0* [online]. Standard. Organization for the Advancement of Structured Information Standards, March 2008 [cit. 7. December 2020]. Available at: <http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-tech-overview-2.0.pdf>.
- [35] *Authentication Context for the OASIS Security Assertion Markup Language(SAML) V2.0* [online]. Standard. Organization for the Advancement of Structured Information Standards, March 2005 [cit. 7. December 2020]. Available at: <https://docs.oasis-open.org/security/saml/v2.0/saml-authn-context-2.0-os.pdf>.
- [36] *Bindings for the OASIS Security Assertion Markup Language (SAML) v2.0* [online]. Standard. Organization for the Advancement of Structured Information Standards, March 2005 [cit. 7. December 2020]. Available at: <http://docs.oasis-open.org/security/saml/v2.0/saml-bindings-2.0-os.pdf>.

- [37] *Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) v2.0* [online]. Standard. Organization for the Advancement of Structured Information Standards, March 2005 [cit. 7. December 2020]. Available at: <http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf>.
- [38] *Metadata for the OASIS Security Assertion Markup Language (SAML) v2.0* [online]. Standard. Organization for the Advancement of Structured Information Standards, March 2005 [cit. 7. December 2020]. Available at: <http://docs.oasis-open.org/security/saml/v2.0/saml-metadata-2.0-os.pdf>.
- [39] *Profiles for the OASIS Security Assertion Markup Language (SAML) v2.0* [online]. Standard. Organization for the Advancement of Structured Information Standards, March 2005 [cit. 7. December 2020]. Available at: <http://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf>.
- [40] *OneLogin SAML Developer Tools* [online]. 2015 [cit. 26. April 2021]. Available at: https://www.samltool.com/online_tools.php.
- [41] PAUL MADSEN. *The Top 6 Authentication Mechanisms* [online]. [cit. 30. December 2020]. Available at: <https://www.pingidentity.com/en/company/blog/posts/2016/the-top-6-authentication-mechanisms.html>.
- [42] PEYROTT, S. *Introduction to Web Authentication: The New W3C Spec* [online]. June 2018 [cit. 9. January 2021]. Available at: <https://auth0.com/blog/introduction-to-web-authentication/>.
- [43] PING IDENTITY. *SAML 2.0: How It Works* [online]. [cit. 7. December 2020]. Available at: <https://www.pingidentity.com/en/resources/client-library/articles/saml.html>.
- [44] PONEMON INSTITUTE LLC. *The 2020 State of Password and Authentication Security Behaviors Report* [online]. February 2020 [cit. 2. January 2021]. Available at: https://resources.yubico.com/53ZDUYE6/as/q9ugik-5mv8o0-dkloty/The_2020_State_of_Password_and_Authentication_Security_Behaviors_Report.pdf.
- [45] POORTVLIET, J. *Nextcloud 11 sets new standard for security and scalability* [online]. December 2016 [cit. 26. April 2021]. Available at: <https://nextcloud.com/blog/nextcloud-11-sets-new-standard-for-security-and-scalability/>.
- [46] POWERS, A. *FIDO TechNotes: The Truth about Attestation* [online]. July 2018 [cit. 9. January 2021]. Available at: <https://fidoalliance.org/fido-technotes-the-truth-about-attestation/>.
- [47] SAMI, L. *WebAuthn Is Great and It Sucks*. *Okta Security*. april 2020. Available at: <https://sec.okta.com/articles/2020/04/webauthn-great-and-it-sucks>.
- [48] *SAML* [online]. April 2021 [cit. 28. April 2021]. Available at: <https://www.webopedia.com/definitions/saml/>.
- [49] SCAVO, T. *Single sign-on using SAML in a Web browser* [online]. 2011 [cit. 7. December 2020]. Available at: <https://en.wikipedia.org/w/index.php?curid=32521419>.

- [50] STALLINGS, W. *Cryptography and Network Security: Principles and Practice*. Prentice Hall, 1999. ISBN 9780138690175. Available at: <https://books.google.sk/books?id=Dam9zrViJjEC>.
- [51] STAVROULAKIS, P. and STAMP, M. *Handbook of Information and Communication Security*. Springer Berlin Heidelberg, 2010. Handbook of Information and Communication Security. ISBN 9783642041174. Available at: <https://books.google.sk/books?id=I-9P1EkTkigC>.
- [52] SUNDAS CHOUDRY. *The Challenge of Building SAML Single Logout* [online]. May 2020 [cit. 10. January 2021]. Available at: <https://www.identityserver.com/articles/the-challenge-of-building-saml-single-logout>.
- [53] SWIFT. *Alliance Web Platform SE* [online]. 2021 [cit. 26. April 2021]. Available at: <https://www.swift.com/our-solutions/interfaces-and-integration/alliance-web-platform-se>.
- [54] *Jump in cyber attacks during Covid-19 confinement* [online]. June 2020 [cit. 8. May 2021]. Available at: <https://www.swissinfo.ch/eng/jump-in-cyber-attacks-during-covid-19-confinement/45818794>.
- [55] THALES. *Banking & Payment* [online]. 2021 [cit. 26. April 2021]. Available at: <https://www.thalesgroup.com/en/markets/digital-identity-and-security/banking-payment>.
- [56] TOEWS, B. *GitHub supports Universal 2nd Factor authentication* [online]. October 2015 [cit. 26. April 2021]. Available at: <https://github.blog/2015-10-01-github-supports-universal-2nd-factor-authentication/>.
- [57] WILSON, Y. and HINGNIKAR, A. *Solving Identity Management in Modern Applications: Demystifying OAuth 2.0, OpenID Connect, and SAML 2.0*. Apress, 2019. ISBN 9781484250952. Available at: <https://books.google.sk/books?id=EJXFDwAAQBAJ>.
- [58] WINMA HEENATIGALA. *SAML ECP (Enhanced Client or Proxy) Profile* [online]. July 2018 [cit. 8. May 2021]. Available at: <https://medium.com/@winma.15/saml-ecp-enhanced-client-or-proxy-profile-97f8fd051c6>.
- [59] YURIY, A. *Why Cloudflare's CAPTCHA replacement with FIDO2/WebAuthn is a really bad idea* [online]. May 2021 [cit. 14. May 2021]. Available at: <https://herrjemand.medium.com/why-cloudflares-captcha-replacement-with-fido2-webauthn-is-a-really-bad-idea-d5487f6c7566>.

Appendices

Appendix A

CD Contents

Attached CD contains:

- `xchrip00dp.pdf` – this thesis,
- `tex` – directory with sources for \LaTeX
- `saml` – directory with source codes for SAML Authentication Component
 - `saml.js` – main JavaScript file
 - `README.md` – internal (technical) documentation
 - `client.html` – HTML file with `WebSDK` script used for authentication
 - `push_client.html` – HTML file, which can be used instead of `WebSDK` for push notification instead of showing the QR code, so a `WebSDK` alternative
 - `actions.html` – HTML file for auto-submitting responses
 - `login_response_template.xml` – XML file including login response template
- `doc` – sources for Excalibur SAML public documentation [6]
- `websdk` – directory containing sources for the `WebSDK` component ([Section 3.1 – Excalibur Clients](#)), only minor changes were made to this component