Návrh autentizačního protokolu za využití bezpečnostních tokenů

Bc. Peter Bátora

Diplomová práce
2020

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně

Fakulta aplikované informatiky

Ústav elektroniky a měření

Akademický rok: 2019/2020

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Peter Bátora**
Osobní číslo:	**A17309**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Bezpečnostní technologie, systémy a management**
Forma studia:	**Kombinovaná**
Téma práce:	**Návrh autentizačního protokolu za využití bezpečnostních tokenů**
Téma práce anglicky:	**The Design of an Authentication Protocol Using Security Tokens**

Zásady pro vypracování

1. Rozeberte a nastudujte problematiku autentizace pomocí tokenů.
2. Popiště aktuálně využívané principy autentizace za využití tokenů.
3. Navrhněte vlastní řešení protokolu využívající bezpečnostních tokenů.
4. Implementujte navržené řešení ve vhodném programovacím prostředí.
5. Vhodně reprezentujte výsledky řešení a implementace.

Rozsah diplomové práce:
Rozsah příloh:
Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. https://oauth.net/books/
2. https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=83893
3. LEK, Kamol a Naruemol RAJAPAKSE. Cryptography: protocols, design, and applications. New York: Nova Science Publishers, c2012, ix, 242 s. Cryptography, steganography and data security. ISBN 978-1-62100-779-1.
4. BOYD, Ryan. Getting started with OAuth 2.0. Sebastopol, CA: O'Reilly, c2012, x, 66 s. ISBN 978-1-449-31160-5.
5. BAIER, Dominick. A guide to claims-based identity and access control: authentication and authorization for services and the web. Redmond: Microsoft, c2010, xxiv, 148 s. Patterns & practices. ISBN 978-0-7356-4059-7.

Vedoucí diplomové práce: **Ing. Petr Žáček**
Ústav informatiky a umělé inteligence

Datum zadání diplomové práce: **9. prosince 2019**
Termín odevzdání diplomové práce: **29. května 2020**

L.S.

doc. Mgr. Milan Adámek, Ph.D.
děkan

Ing. Milan Navrátil, Ph.D.
ředitel ústavu

Ve Zlíně dne 9. prosince 2019

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 14.8.2020

PETER BÁTORA
.....
podpis diplomanta

ABSTRAKT

Tato diplomová práce se zabývá návrhem autentizačního protokolu s využitím bezpečnostních tokenů a jeho následnou implementací. V teoretické části jsou čtenářům přiblíženy problematiky kryptologie, autentizace a také jsou osvětleny nejznámější a nejpoužívanější autentizační protokoly: SAML 2 a OAuth 2. V praktické části řeším tvorbu webové aplikace, popisuji a jaké nástroje jsem k tomu použil. Další kapitoly praktické části se zabývají vlastní tvorbou a použitím nově navrženého bezpečnostního tokenu. Následuje vyhodnocení tokenu, kde nový token porovnávám s aktuálně používanými tokeny (SAML Assertion a JWT). Zmiňuji se zde také o možnostech vylepšení protokolu. Poslední část se věnuje závěru, kde zopakuji, co bylo cílem práce, co a jak jsem dělal a shrnutí toho, na co jsem přišel.

Klíčová slova: Autentizace, autorizace, token, SAML 2, OAuth2, JWT, Kryptologie, Flask, Python

ABSTRACT

This diploma thesis deals with the design of an authentication protocol using security tokens and its subsequent implementation. In the theoretical part readers are introduced to the issues of cryptology, authentication and also sheds light on the best known and most used authentication protocols: SAML 2 and OAuth 2. In the practical part I am creating a web application and I am describing what tools I used. The next chapters of the practical part deal with the creation and practical use of a new token in a newly created web application. The following part is an evaluation of the token, where I am comparing the new token with the currently used tokens (SAML Assertion and JWT). I am also mentioning here the possibilities of improving the protocol. The last part deals with the conclusion, where I am repeating what was the goal of the work, what and how I did and a summary of what I came up with.

Keywords: Authentization, authorization, token, SAML 2, OAuth2, JWT, Cryptology, Flask, Python

Rád bych touto formou poděkoval svému vedoucímu diplomové práce panu Ing. Petrovi Žáčkovi za cenné rady, postřehy a konzultace, které vedly k dokončení diplomové práce. Dále bych také chtěl poděkovat všem, kteří mě podporovali, motivovali a také tolerovali, své přítelkyni, její rodině a v neposlední řadě celé své rodině, především svému strýci Ing. Oldřichu Vykydalovi, který je pro mě velkým vzorem a motivací.

OBSAH

ÚVOD

V naší moderní době 21. století neboli v digitálním světě se s pojmy autentizace, autorizace, autentizační protokoly, tokeny aj. setkáváme každý den. Většina z nás si toho ani nevšimne, ale přesto je tomu tak. Jako příklad lze uvést přihlašovaní se na web citacepro.com. Každý, kdo na tento webový portál přijde se může přihlásit svým uživatelským jménem a heslem anebo se může přihlásit přes svoji instituci. Po volbě přihlášení přes instituci nás webový prohlížeč přesměruje na webovou stránku s přihlašovacím formulářem, který ale nepatří stránce, ze které jsem přišel. Patří instituci, přes kterou se hodlám přihlásit. Po zadání přihlašovacích údajů a jejich ověření (autentizaci) serverem instituce nám autentizační protokol přidělí bezpečnostní token a přesměruje nás zpět na původní web, kde se prokážeme token a server nás přihlásí. Tuto funkcionalitu má na starost jeden z nejpoužívanějších autentizačních protokolů SAML 2 a token, který předává potřebné informace se jmenuje SAML Assertion. Obdobně to funguje i v prostředí sociálních sítí Facebooku a Googlu, kde se o toto stará autentizační protokol OAuth 2 a jeho dobře známý JWT.

Hlavním důvodem, proč jsem si toto téma vybral je nejen, že mě problematika bezpečnosti ve webovém prostředí zajímá, ale také pokusit se navrhnout a implementovat nový autentizační protokol, a hlavně nový způsob tvorby bezpečnostních tokenů. Jak jsem zmínil výše, nejznámější a nejpoužívanější autentizační protokoly jsou SAML 2 a OAuth 2. Oba se používají již nějakou dobu a prošly modernizací, jak můžeme vidět na jejich názvech. Nicméně se od sebe liší oblastí, kde působí, formátem, zabezpečením, a hlavně bezpečnostním tokenem, který používají.

V teoretické části jsem čtenáře seznámil se základními pojmy jako jsou kryptologie, šifrovací algoritmy a jejich příklady. Nezapomněl zmínit co je to autentizace, autorizace, ale hlavně se budu věnovat popisu autentizačních protokolů SAML 2 a OAuth2 a vysvětlil, jak vlastně fungují. V první kapitole praktické části jsem popsal použité nástroje pro vytvoření webové aplikace, kde byl nový protokol implementován. Následuje tvorba a použití nového bezpečnostního tokenu. V této časti čtenáři uvidí zdrojové kódy webové aplikace a souborů, které tyto tokeny generují a pracují s nimi. Celá webová aplikace i logika za novým protokolem byla napsána v programovacím jazyce Python 3.X. V kapitole *5.2* a *5.3* jsou popsány serverová a klientská strana protokolu. V první zmíněné kapitole obsahuje naprogramovacné API endpointy pro získání, kontrolu, obnovení a smazání aktivního tokenu. Na straně klienta je zase napsán skript, který otestuje veškerou logiku (tvorba tokenu, šifrování klíči a

dotazování na nově vytvořené API endpointy). Na závěr bude nový protokol a jeho tokeny porovnány a zhodnocen a vůči aktuálně používaným řešením.

I. TEORETICKÁ ČÁST

1 KRYPTOLOGIE

1.1 Základní pojmy

Kryptologie je vědní disciplína zabývající se šifrováním a dešifrováním zpráv. Zahrnuje také pojmy Kryptografie a Kryptoanalýza. Kryptografie neboli šifrování se zabývá zašifrováním odesílané zprávy jinými slovy převodem zprávy do podoby, která je čitelná jen se správným klíčem. Kryptografii nejde o utajení přenosu zprávy, ale o nemožnosti jejího přečtení, bez klíče. Kryptoanalýza se zase zabývá prolamováním šifer – řešením šifry bez jejího klíče.

1.2 Cíle kryptografie

V souvislosti s definicí pojmu Kryptografie je také nutné zmínit její cíle:

- Důvěrnost,
- celistvost dat,
- autentizace,
- autorizace,
- nepopiratelnost.

Důvěrnost – confidentality – je nejdůležitějším cílem kryptogfrafie. Zjednodušeně jde o zachování obsahu zprávy v utajení.

Celistvost dat – data integrity – klade důraz na zamezení neoprávněných manipulací dat. Mezi tyto úpravy můžeme zařadit vložení nových dat, smazání části původních dat nebo jejich substituci. Součástí zamezení těchto manipulací také vzniká povinnost tyto změny detekovat.

Autentizace – authentification – proces ověření identity žadatele, že jde o tu osobu, za kterou se vydává.

Autorizace – authorization – potvzení původu dat. Ověření, zda data vytvořil ten, který se za autora vydává.

Nepopiratelnost – non-repudiation – souvisí s autorizací – jedná se o důkaz, že autor dat nemůže popřít své autorství. [1;2;3;4]

1.3 Šifrování

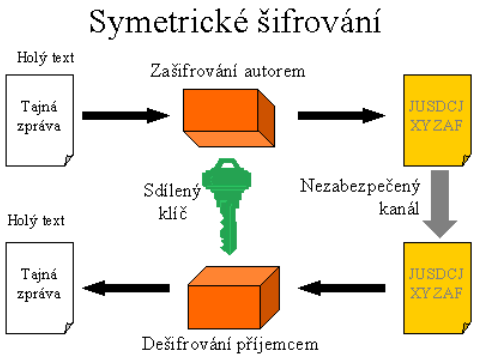
Pod pojmem šifrování se skrývá kryptografický algoritmus založený na matematických funkcích. Jinými slovy, tento algoritmus převádí nezabezpečené, čitelné zprávy na data zašifrovaná. Čitelná pouze pro majitele dešifrovacího klíče. Parametrem šifrování není přímo ten algoritmus, ale klíč. Zpětný převod zašifrovaných dat na čitelné se nazývá dešifrování. Podle použitých klíčů šifrování dělíme na:

- Symetrické šifrování,
- asymetrické šifrování. [3;5;6]

Obrázek č. 1 – Obecné šifrování [3]

1.4 Symetrické šifrování

V případě tohoto typu šifrování se stejný klíč používá nejen pro šifrování, ale také pro dešifrování. Postup je poměrně jednoduchý. Odesílatel a příjemce se nejprve domluví na klíči, tedy na sekvenci znaků (ideální je, když jsou znaky náhodné bez jakéhokoliv významu). Odesílatel zprávu zašifruje, odešle ji příjemci a ten si ji pomocí výše zmíněného klíče dešifruje. Výhodou je nízká výpočetní náročnost a s tím související rychlost šifrování a dešifrování. Nevýhodou naopak je nutnost domluvy na stejném klíči. Tento klíč musíme zasílat v zašifrované komunikaci, protože kdyby ho útočník získal, tak by následně mohl dešifrovat naše další zprávy. [5;6]



Obrázek č. 2 – Symetrické šifrování [7]

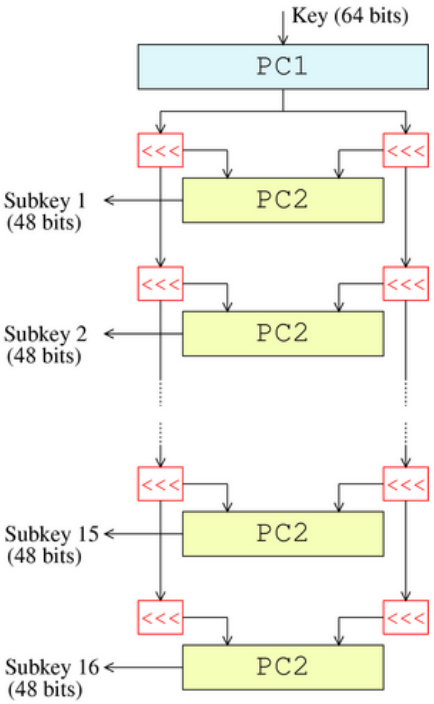
1.4.1 Typy symetrických šifer

V této kapitole vyjmenuji a stručně popíši některé typy symetrických šifer.

1.4.1.1 DES

Data Encryption Standard (DES) je druh blokového symetrického šifrovacího algoritmu se soukromým klíčem používaný k ochraně soukromých dat. Byl to první šifrovací algoritmus, který byl uznán vládou USA a zveřejněn pro všeobecné užívání. V současnosti je tato šifra považována za nespolehlivou, protože používá klíč pouze o délce 64 bitů, z toho 8 je kontrolních a 56 efektivních. Data jsou rozdělena do 64bitových bloků a každý z nich je jednotlivě transformován šifrovacím algoritmem. Aby opakované použití transformace na všechny bloky nebylo slabinou vedoucí ke snadnějšímu prolomení šifry, jsou zde navíc uplatněny tzv. “modes of operation”, díky nímž zašifrovaná data působí jako náhodné sekvence. DES byla založena na Feistelově blokové šifře a funguje na principu lavinového

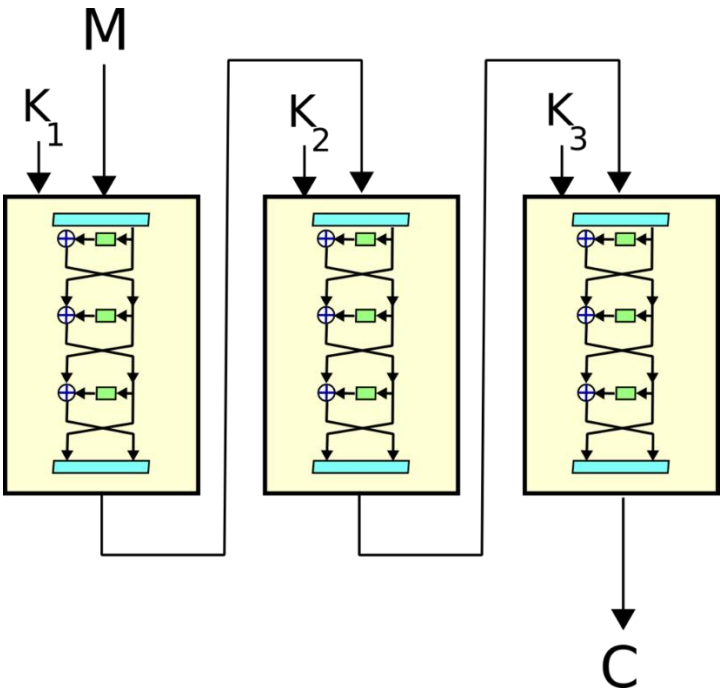
efektu - tzn. malá změna v původního textu způsobí obrovskou změnu v textu zašifrovaném. V roce 1999 společnost Electronic Frontier Foundation vytvořila přístroj DES Cracker, který v té době stál 250 000 dolarů. Dokázal rozluštit 88 miliard klíčů za vteřinu a šifru prolomil za 56 hodin. O půl roku později se jim to podařilo prolomit za 22 hodin. Už minimálně z tohoto důvodu bylo jasno, že se musí vytvořit lepší a bezpečnější standard. Výsledkem byl Triple DES. [8;9;10]



Obrázek č. 3 – DES šifra [11]

1.4.1.2 Triple DES

Triple DES (3DES nebo TDES) je bloková šifra přímo vycházející ze šifrovacího algoritmu DES (zmíněného v kapitole 1.4.1.1). Jediným rozdílem je, že DES aplikuje 3krát, čímž zvyšuje odolnost vůči prolomení. 3DES byl nejjednodušším způsobem, jak zvýšit odolnost DESu bez nutnosti vývoje nového algoritmu. [12]



Obrázek č. 4 – 3DES šifra [13]

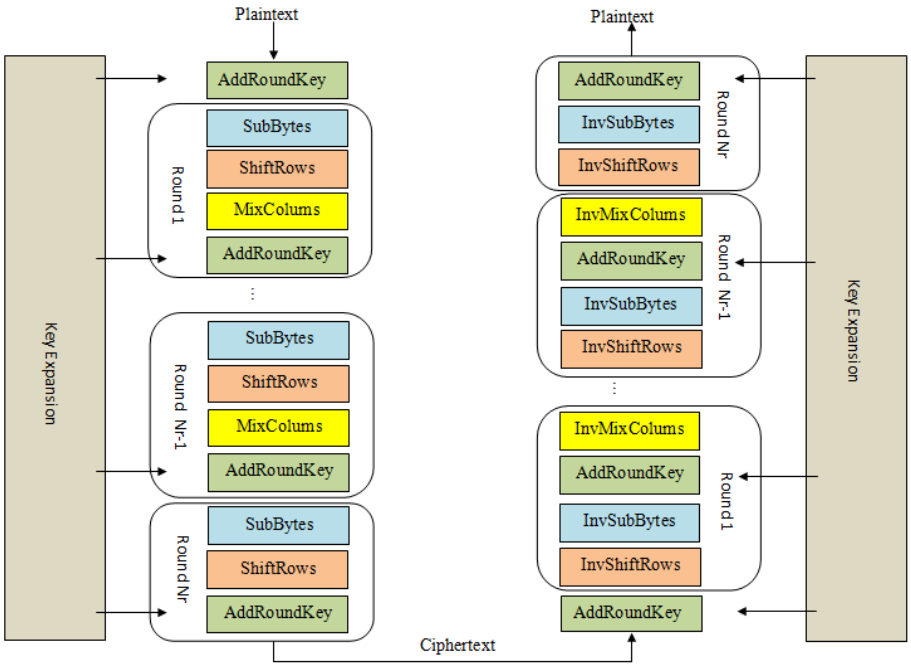
1.4.1.3 AES

Advanced Encryption Standard (AES) je v současné době nejvíce používaná symetrická šifra. Původní název šifry byl Rijndael podle přesmyčky jmen svých autorů (Vincent Rijmen a Joan Daemen) Doposud se neobjevil žádný způsob, jak tuto šifru prolomit, vyjma útoku hrubou silou. AES je symetrická bloková šifra, která zpracovává data v blocích o velkosti 128 bitů a za použití šifrovacích klíčů o délce 128, 192 nebo 256 bitů. Algoritmus AES byl navržen tak, aby bych schopen používat i jiné velikosti šifrovaných bloků a klíčů, nicméně nejsou zahrnuty ve standardu AES.

Principem AESu je použití substituce a permutace k převedení bloků čitelné zprávy na šifrovanou a naopak. U této šifry dochází na rozdíl od šifry DES, kde se šifrovalo pouze 32bitů, k zašifrování všech 128 bitů. Na rozdíl od starších algoritmů nepracuje s vektory, ale se stavovými maticemi 4x4 bajty, pro které jsou definované čtyři transformace a k nim příslušné inverzní funkce. Šifrovací i dešifrovací transformace jsou:

- AddRoundKey – přidání opakovacího(iteračního) klíče
- SubBytes – substituce bajtů – inverzní fce: InvSubBytes
- ShiftRows – posun řádků – inverzní fce: InvShiftRows
- MixColumns – posun sloupců – inverzní fce: MixColumns

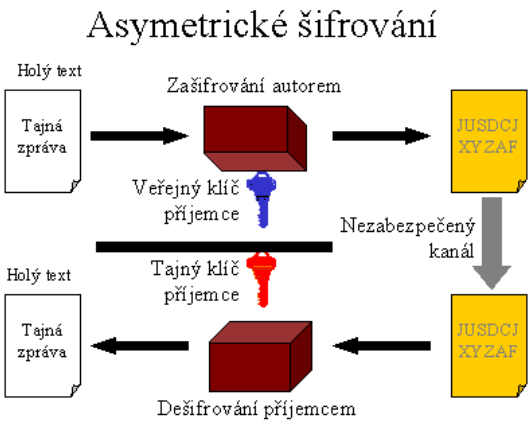
Jedním z hlavních důvodů, proč byl AES zvolen jako standard, je i jeho efektivita, malá náročnost na počítačovou paměť a vysoká výkonnost při aplikaci v softwaru a hardwaru, a to jak u osobních počítačů, tak i v komplexnějších vládních či komerčních systémech.[14]



Obrázek č. 5 – AES šifra [14]

1.5 Asymetrické šifrování

Tato metoda je novější a více používanější než symetrické šifrování. Princip asymetrického šifrování spočívá v existenci dvou různých klíčů – veřejného a soukromého. Veřejným klíčem se data zašifrují a dešifrovat je pak lze pouze klíčem soukromým. Odesílatel zašifruje zprávu veřejným klíčem příjemce a odešle ji. Pokud se zprávy zmocní někdo jiný, nemůže ji dešifrovat, protože nezná soukromý klíč příjemce. Jediný, kdo si zprávu může přečíst, je jen skutečný příjemce. Veřejný klíč a soukromý klíč jsou vzájemně spojeny a tvoří tzv. klíčový pár. Ze soukromého klíče lze odvodit veřejný klíč, ale opačné odvození možné není. Komunikují-li dvě osoby obousměrně, tak si vzájemně vymění své veřejné klíče, kdežto soukromé klíče si ponechávají jen pro sebe. Možnost zahájení bezpečného šifrovaného přenosu i bez předchozí výměny klíčů je velkou výhodou asymetrického šifrování. Nevýhodou je naopak řádově vyšší výpočetní náročnost. [5;6;15]



Obrázek č. 6 – Asymetrické šifrování [16]

1.5.1 Typy asymetrických šifer

V této kapitole stručně popíšu některé typy asymetrických šifer.

1.5.1.1 RSA

Asymetrická šifrovací funkce RSA patří mezi první šifrovací systémy používající veřejný klíč a byla poprvé publikována v roce 1977. Jméno funkce se skládá z počátečních písmen autorů – Ronald **R**ivest, Adi **S**hamir a Leonard **A**dlemann (**RSA**). RSA je sice pomalejší než symetrické kryptosystémy, ale našla si uplatnění v oblasti digitálních podpisů a pro bezpečné uchování nebo transport klíčů. Šifrovací algoritmus je založen matematicky

náročném prvočíselném rozpadu. RSA klíče mají typicky 1024 až 4096 bitů, ale experti věří, že 1024bitové klíče mohou být v blízké budoucnosti prolomeny.

Tvorba klíčového páru pomocí šifry RSA:

1. Zvolíme dvě různá náhodná prvočísla **p** a **q**.
2. Spočítá jejich součin **n = p*q**.
3. Spočítá hodnotu Eulerovy funkce **φ(n) = (p − 1)(q − 1)**.
4. Zvolíme celé liché číslo **e** menší než **φ(n),** které je s **φ(n)** nesoudělné.
5. Nalezne číslo **d** tak, aby platilo **de ≡ 1 (mod φ(n))**, kde symbol ≡ značí kongruenci zbytkových tříd.
6. Jestli **e** je prvočíslo, tak **d = (1+r*φ(n))/e**, kde **r = [(e-1)φ(n)^(e-2)]**. [17]

1.5.1.2 ElGamal

ElGamal patří mezi asymetrické šifrovací algoritmy. Je založen na výpočtu diskrétního logaritmu. Díky těmto výpočtům se zvětší šifrovací data až na dvojnásobek své původní délky. Toto je možná jeden z důvodů, proč se nedočkal masového využití.

Popis šifrovacího algoritmu

Generování klíče:

- Veřejné parametry: Generátor **g** multiplikativní cyklické grupy **G** řádu **q**.
- Tajný klíč: Náhodné číslo **x** z {0, 1, . . . , q − 1}.
- Veřejný klíč: $\mathbf{h = g^x}$. Zveřejněn bude veřejný klíč h a také informace o grupě (tzn. **G,g,q**).
- Tajný klíč **x** je třeba ponechat v tajnosti.

Šifrování:

- Zašifrujeme zprávu **m** pomocí veřejného klíče (**h, G, g, q**).
- Vybereme náhodné **y** z {0, 1, . . . , q − 1}.
- Spočteme $\mathbf{c_1 = g^y}$.
- Spočteme $\mathbf{s = h^y}$ (tzv. ephemeral key = jepičí klíč).
- Pozn. Tyto kroky bylo možné provést před samotným šifrováním.
- Převedeme zprávu **m** do zprávy $\mathbf{m^1 \in G}$ (např. pomocí OAEP).
- Spočítáme $\mathbf{c_2 = s \cdot m^1}$ (= $\mathbf{h^y \cdot m^1}$).

- Odešleme šifrový text ($\mathbf{c_1}$, $\mathbf{c_2}$).
- Pozn. Dále je nutné dobře utajit (popř. zničit) ephemeral key **s**.

Dešifrování:

- Obdržíme šifrový text ($\mathbf{c_1}$, $\mathbf{c_2}$).
- Spočteme $\mathbf{m^1 = c_2 \cdot ((c_1)^x)^{-1}}$.
- Převedeme na původní zprávu **m**.
- Postup funguje, protože:
  - $\mathbf{c_2 \cdot (c_1^x)^{-1} = s \cdot m^1 \cdot ((g^y)\, x)^{-1} =}$
  - $\mathbf{= h^y \cdot m^1 \cdot (g^{xy})^{-1} = g^{xy} \cdot m^1 \cdot (g^{xy})^{-1} = m^1}$. [18;19]

2 AUTENTIZACE

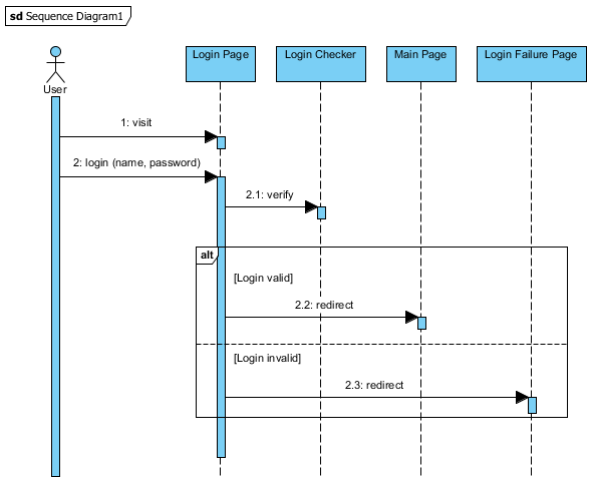
Termín autentizace (anglicky Authentication) vznikl z řeckého slova *authentikos,* neboli skutečný či pravý. Jedná se o proces ověřování identity uživatele, objektu, za který se vydává. Autentizace přímo navazuje na **identifikaci** neboli proces předložení identifikačních informací. Např. se může jednat o zadání uživatelské jména, identifikačního čísla, certifikátu atd. Autentizace a identifikace nemůžou fungovat jeden bez druhého – ani jeden proces není sám o sobě užitečný. Po úspěšné autentizaci přichází proces **autorizace** – akt přidělování příslušných práv k využívání různých služeb v konkrétním systému, např. možnost zápisu do databáze, čtení určitých souborů atd. Existují tři základní formy autentizace:

- Autentizace znalostí (znám heslo, pin),
- autentizace vlastnictvím (mám přístupovou kartu, klíč, token),
- autentizace vlastností (biometrika – otisk prstu, sítnice). [20;21]

2.1 Autentizace znalostí

Autentizace znalostí je nejjednodušší a v současné době nejrozšířenější forma autentizace. Probíhá zadáním uživatelského jména a hesla. Zakládá se na faktu, že uživatel svoje heslo zná nebo ho má někde uloženo. Při procesu přihlášení uživatel zadá své uživatelské jméno i heslo a server tyto údaje porovná se svou databází a pokud proces autentizace proběhne úspěšně, pustí uživatele dál. Autentizaci znalostí (heslem) rozdělujeme na:

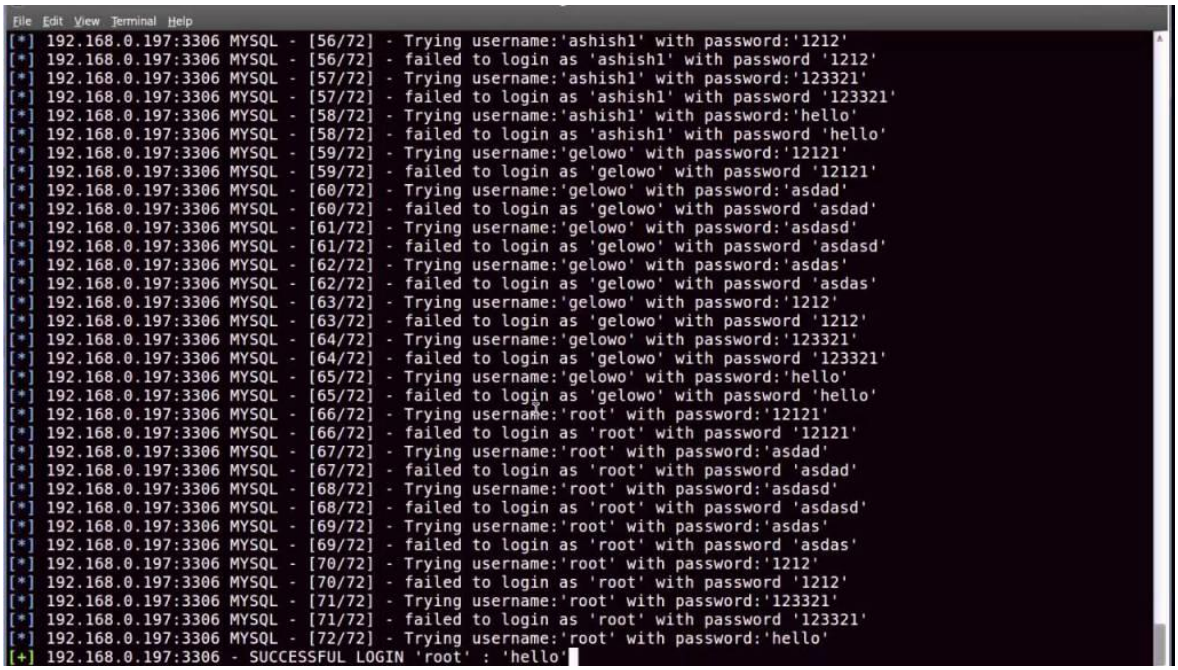
- Statická,
- dynamická,
- jednorázová. [22;23]



Obrázek č. 7 – Autentizace znalostí [24]

2.1.1 Statická hesla

Statická hesla jsou historicky nejstarší a také nejméně bezpečný autentizační údaj. Důležitou vlastností těchto hesel je jejich kvalita – celková délka hesla a počet alfanumerických znaků. Doporučovaným způsobem zvyšování kvality hesel je jejich časté obměňování a nepoužívání stejných hesel do různých informačních systémů. Hesla bývají typicky dlouhá 6-12 znaků a obsahují kombinaci velkých, malých písmen a čísel. Nemělo by se skládat pouze z konkrétního slova a k němu přidanému čísla, protože taková hesla bývají náchylnější k prolomení pomocí slovníkového útoku (při tomto útoku útočníci mají k dispozici slovníky ukradených hesel a ty pak zkoušejí použít k prolomení přihlašovacích údajů) nebo útoku hrubou silou (Brute Force Attack) – jedná se o pokus rozluštění šifry bez znalosti klíče k dešifrování neboli systematické testování všech možných kombinací znaků. [22;23]



Obrázek č. 8 – Brute Force Attack [25]

2.1.2 Dynamická hesla

Dynamická hesla částečně odstraňují nevýhody výše zmíněných statických hesel, kdy při každém přihlášení je ověřovatelem vygenerováno nové heslo podle předem definovaného algoritmu, který jej zašle žadateli o přihlášení. Toto heslo poté nelze použít při příštím pokusu o přihlášení – má limitovanou životnost. Lidé se s touto formou autentizace mohou setkat například při pokusech o přihlášení do internetového bankovnictví (IB). Po zadání svého uživatelského jména a hesla přijde uživateli SMS zpráva s vygenerovaným číselným kódem. Uživatel potom tento kód zadá do formuláře v IB a pokud tento kód zadal správně – systém ho pustí dál.

2.1.3 Jednorázová hesla

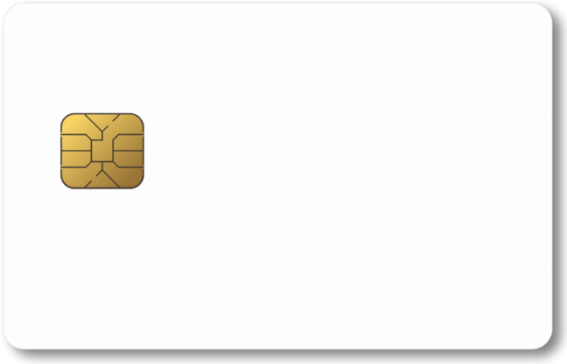
Jednorázová hesla neboli OTP (One-time passwords) bývají zpravidla používány k jednorázovému přihlášení nebo transakcím. Na rozdíl od dynamických hesel se tajná informace generuje na obou stranách – žadatel i ověřovatel. Příkladem může být mobilní autentifikátor do různých onlinových her – při pokusu o přihlášení se na mobilní aplikaci vygeneruje náhodný osmimístný číselný kód, který je poté potřeba zadat do formuláře přihlášení. Tento typ autentizace se nejvíce hodí k přihlašování na nezabezpečené počítače. [25;29]

2.2 Autentizace vlastnictvím

Žadatel o autentizaci prokáže svoji identitu nějakým přenosným předmětem, nejčastěji nazývaným jako **token**. Tyto předměty v sobě obsahují tajnou informaci, například heslo nebo kryptografický klíč anebo mají typický tvar, elektrický odpor nebo kapacitu. Složitější tokeny také mohou provádět kryptografické výpočty. Mezi nejčastější tokeny patří čipové a paměťové karty, USB tokeny nebo autentizační kalkulátory. Tokeny nabízejí obecně bezpečnější a efektivnější metodu autentizace a kombinací vícero typů autentizace se tato bezpečnost ještě zvyšuje. Příkladem může být platební karta do bankomatu – kombinace autentizace vlastnictvím (mám kartu u sebe) a autentizace znalostí (znám svůj PIN). Velkou výhodou oproti autentizaci znalostí je, že uživatel si nemusí pamatovat žádná složitá hesla. Také se dá jednoduše zjistit, zda byl tento token odcizen a je velmi složité tyto tokeny

duplikovat. Mezi hlavní nevýhody lze zařadit nemožnost se autentizovat při ztrátě tokenu. [22;23;26]

..



Obrázek č. 9 – Čipová karta [27]

2.3 Autentizace vlastností

Autentizací vlastností (biometriky) je založena na měření fyziologických vlastností člověka (otisk prstu, ruky nebo sítnice) nebo na chování jedince (vzorek hlasu, dynamický způsob podpisu). Fyziologických autentizace je zpravidla spolehlivější, protože není tak moc ovlivněna psychickým nebo fyziologickým stavem (stres, nemoc). Základní procesy biometrického zabezpečení jsou sběr dat, analýza, extrakce a jejich uchovávání. Nejvýznamnějším rozdílem mezi biometrickým způsobem autentizace a klasickým je ve způsobu vyhodnocení odpovědi systému. Při vyhodnocování biometrické autentizace ověřovatel neumí odpovědět se 100% jistotou. Není to jako v případě hesla, které se porovnává v databázi a je tu jasný stav souhlasí/nesouhlasí. Takže aby biometrické ověření bylo použitelné, musíme nastavit určitou odchylku biometrických charakteristik. Na druhou stranu čím vyšší odchylku nastavíme, tím vyšší šanci dáváme útočníkovi k prolomení. [22;23;26]

Obrázek č. 10 – Čtečka otisku prstů [28]

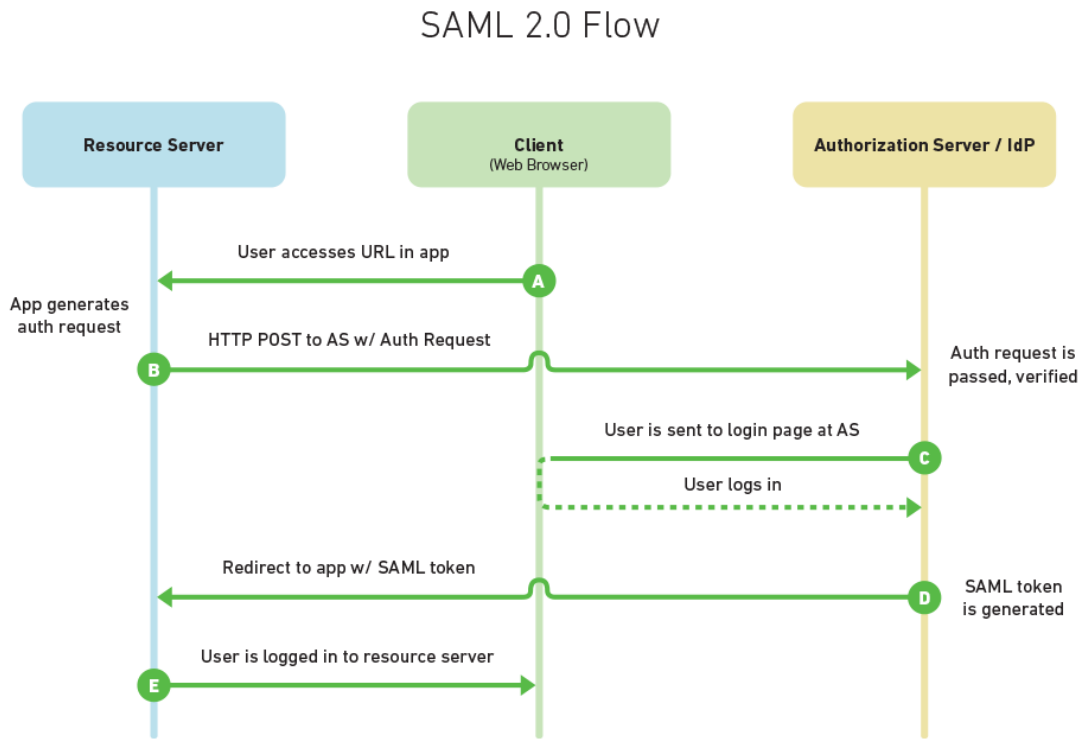
3 AUTENTIZAČNÍ PROTOKOLY

V této kapitole se zabývám představením a srovnáním různých autentizačních protokolů, podle kterých následně navrhnu vlastního řešení autentizačního protokolu využívající bezpečnostní tokeny.

3.1 SAML 2

Security Assertion Markup Language (SAML) je otevřený standard, který umožňuje poskytovateli identit (PID) předávat uživatelské údaje poskytovateli služeb (PS), jinými slovy se stará o centralizovanou autentizaci uživatele. Tyto údaje dále spravuje PID. Je mnohem jednodušší spravovat pouze jedny společné přihlašovací údaje do vícero služeb než spravovat každou z nich zvlášť.

Pro komunikaci mezi PID a PS SAML používá značkovací jazyk XML (Extensible Markup Language). SAML umožňuje SaaS (Software as a Service) – jedná se o hostování aplikace provozovatelem služby a také uživatelsky oblíbenou funkci SSO (Single-Sign On) – proces umožňující uživateli přihlásit se jednou do konkrétní webové aplikace a poté tyto samé přihlašovací údaje použít na přihlášení jiných PS. [29; 30; 31; 32]



Obrázek č. 11 – SAML 2 Flow [29]

Proces komunikace uživatele, PS a PID můžeme vidět na obrázku výše. Vše začíná požadavkem uživatele na konkrétní stránku spravovanou PS. Údaje uživatele spravuje PID. PS vytvoří SAML požadavek, podepíše ho a zašifruje. Poté PS uživatele přesměruje na web PID spolu s požadavkem na přesměrování a jeho vlastním identifikátorem. Pokud uživatel nemá "živou" session, tak uživatele vyzve k zadání přihlašovacích údajů. PID požadavek zpracuje, autentizuje uživatele ze zadaných údajů vůči údajům, které si drží někde ve své databázi. Po úspěšné autentizaci PID vytvoří SAML Token obsahující uživatelské údaje a podepíše jej svým certifikátem a pošle ho zpět PS. PS, který zná PID a má obtisk jeho certifikátu, obdrží tento token, dešifruje ho a získá z něj přihlašovací údaje uživatele a případně jeho práva, pokud nějaké má, a poté uživatele přihlásí. [29; 30; 31; 32]

3.1.1 SAML Assertion

SAML Assertion je typ tokenu, který protokol SAML 2 používá. Je ve strikním XML formátu a pro šifrování na úrovni odesílané zprávy používá XML Encryption. Vůči JWT je mnohem složitější, hlavní pole působnosti bylo podnikové SSO, jehož je SAML již nějakou dobu standardem. Je to balíček informací, který může obsahovat od jednoho až po tři statements (prohlášení) a to:

- Authentication – nese informaci, že tento konkrétní subjekt byl autentifikován určitým způsobem v určitém čase. Většinou je generován PID.
- Attribute – v tomto statementu má subjekt přidružené dodané atributy
- Authorization decision – informace, zda byl subjektu udělen nebo zamítnut přístup k dané službě. [33;34]

3.2 OAUTH 2

OAuth2 je otevřeným standardem, protokolem i frameworkem pro autentizace uživatele a autorizace aplikací. Specifikuje způsob, jak vlastník zdroje udělí souhlas klientovi, aby mohl přistoupit k jeho zdrojům, Dále také definuje, jak může klient přistoupit k jeho zdrojům, aniž by mu musel poskytnout svoje přístupové údaje. Deleguje ověření uživatele, která je hostitelem uživatelského účtu a autorizuje aplikaci třetích stran pro přístup k tomuto účtu. Využívá se například na Facebooku, GitHubu, Twitter atd.

3.2.1 Základní role

Oauth2 definuje čtyři základní role:

- Resource owner – vlastník zdroje
- Client – klient
- Resource Server – server zdrojů
- Authorization Server – autorizační server

3.2.1.1 Resource owner – uživatel

Vlastník zdroje je uživatel, který autorizuje aplikaci třetí strany k použití jeho účtu. Přístup aplikace k tomuto účtu je limitován právy.

3.2.1.2 Client – aplikace

Klient je vlastně aplikace, která žádá o přístup k účtu vlastníka zdroje. Předtím, než mu je to dovoleno ho musí uživatel autorizovat a musí být autentizován programovacím rozhraním API.

3.2.1.3 Resource / Authorization server – API

Server zdrojů uchovává zabezpečené uživatelské účty, autorizační server zase ověřuje identitu uživatele a vydává příslušné tokeny.

3.2.2 Způsoby získávání tokenu

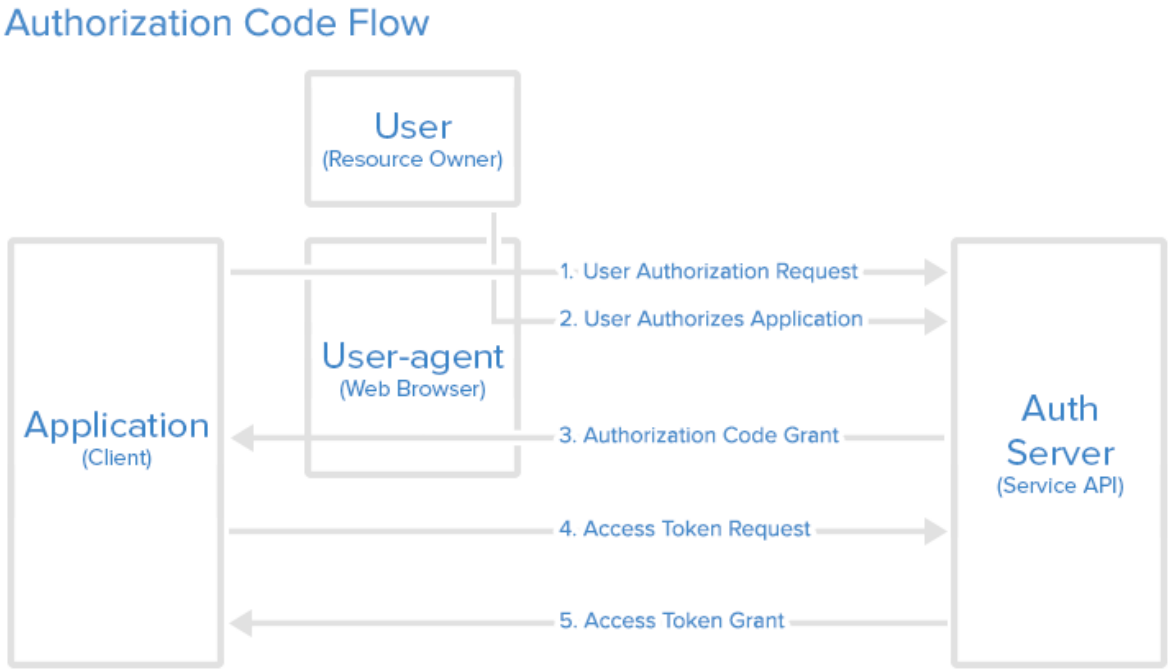
V současné době existuje 5 způsobů získání tokenu, pro naše potřeby popíšu pouze 2 pro serverové aplikace. Nicméně je všechny aspoň můžeme vyjmenovat.

- Authorization code grant

- Implicit grant
- Resource owner credentials grant
- Client credentials grant
- Refresh token grant

3.2.2.1 Authorization code grant

Tento způsob získání tokenu je nejčastěji používaný, protože je optimalizován pro aplikace na straně serveru, kde zdrojový kód není veřejně vystaven a lze tedy zachovat důvěrnost klienta. Jedná se o tok založený na přesměrovávání, což znamená, že aplikace musí být schopna interagovat s uživatelským agentem (tj. webovým prohlížečem uživatele) a přijímat autorizační kódy API, které jsou směrovány prostřednictvím uživatelského agenta.



Obrázek č. 12 – Authorization Code Flow [35]

Jak můžeme vidět v obrázku výše, klient nejdříve pošle autorizační dotaz – dostane autorizační odkaz, který vypadá takto:

```
xxx.com/authorize?re-
sponse_type=code&client_id=CLIENT_ID&redirect_uri=CALLBACK_URL&scope=
read
```

Níže popíšu důležité parametry z tohoto dotazu:

Client_id=CLIENT_ID – jedná se ID klienta, jak ho identifikuje dotazované API

Redirect_uri = CALLBACK_URL – URL, na kterou služba přesměruje webový prohlížeč potom, co klient obdrží autorizační token.
Response_type = code – ukazuje, že klient žádá autorizační token.
Scope=read – specifikuje rozsah práv uživatele.

Po kliknutí na tento odkaz se bude muset uživatel přihlásit (pokud již teda není přihlášen). Poté bude vyzván k autorizování nebo zamítnutí aplikace k využití jeho účtu. Pokud uživatel povolil přístup, tak ho server přesměruje na předem specifikovanou URL i s autorizačním tokenem. Představme si, že předem specifikovaná URL byla yyy.com a tak by naše přesměrování vypadalo nějak takto:

```
yyy.com/callback?code=AUTHORIZATION_CODE
```

Aplikace poté zažádá o přístupový token API tím, že pošle požadavek, který obsahuje výše zmíněný autorizační token a přístupové údaje. Požadavek může vypadat takto:

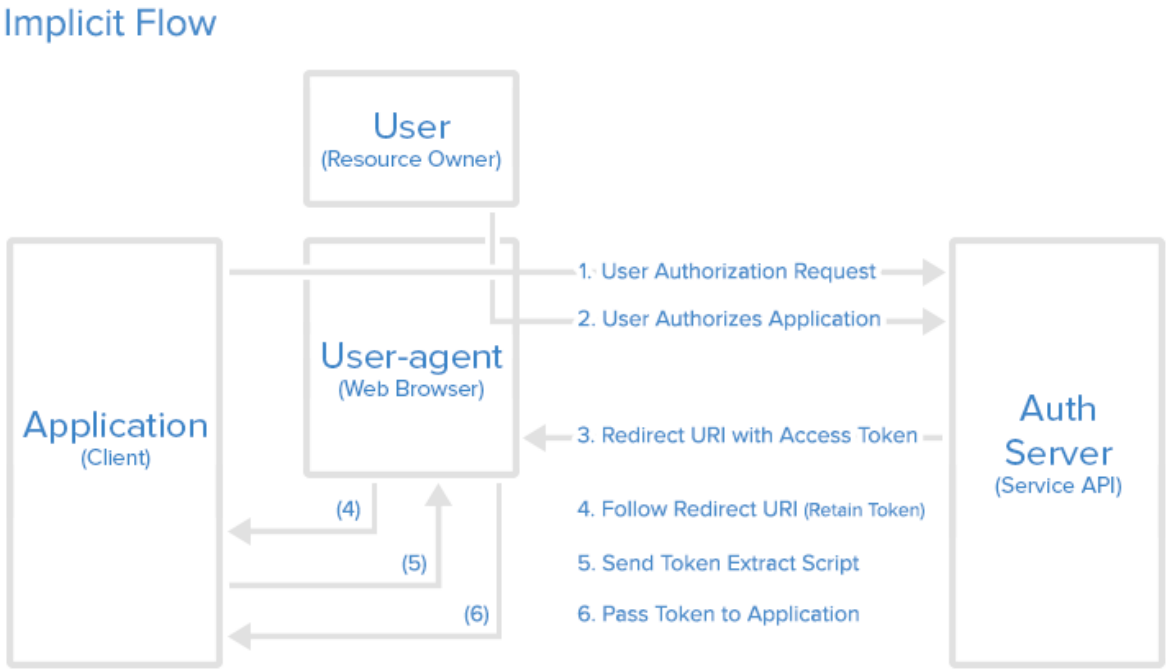
```
xxx.com/to-
ken?client_id=CLIENT_ID&client_secret=CLIENT_SECRET&grant_type=author
ization_code&code=AUTHORIZATION_CODE&redirect_uri=CALLBACK_URL
```

Pokud autorizační server ověří přístupové údaje vygeneruje přístupový token, který následně tento token vrátí jako odpověď. Token má v sobě zadaný rozsah práv a také časový údaj, po kterém tento token vyprší.

```
{"access_to-
ken":"ACCESS_TOKEN","token_type":"bearer","expires_in":50400,"refresh
_token":"REFRESH_TOKEN","scope":"read","uid":123456,"info":
{"name":"PB","email":"p@b.com"}}
```

3.2.2.2 Implicit grant

Implicitní typ grantu se používá pro mobilní a webové aplikace, kde není zaručena důvěrnost klienta. Implicitní typ je také založený na přesměrovávání, ale přístupový token je přidělen uživateli-agentovi, aby jej předal klientovi, takže může být předán uživateli a dalším aplikacím v zařízení uživatele. Tento tok také neověřuje identitu aplikace a k dosažení tohoto účelu se spoléhá na přesměrování URI (které bylo registrováno u služby). Implicitní typ grantu nepodporuje obnovovací tokeny.



Obrázek č. 13 – Implicit Flow [36]

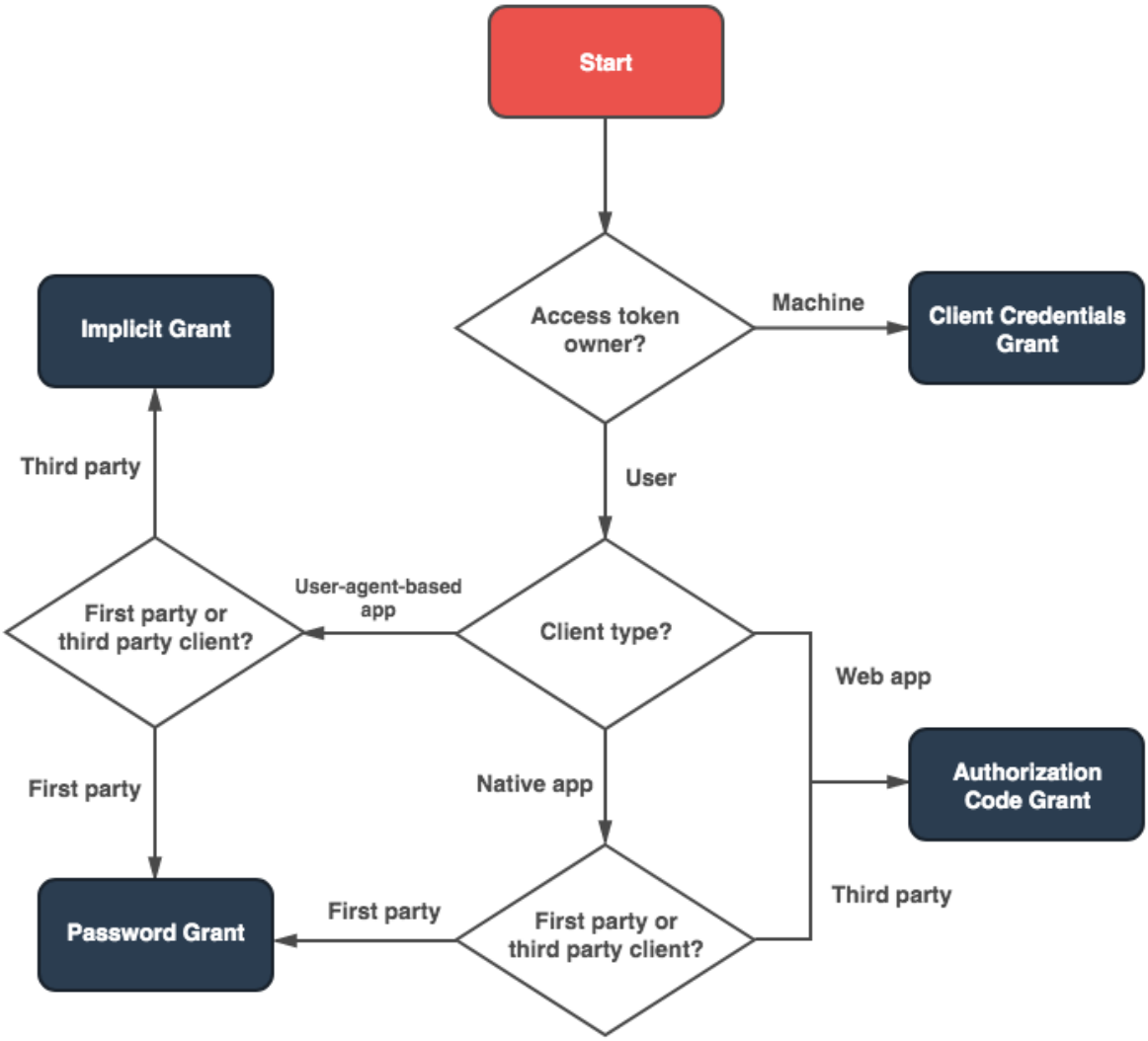
První krokem u tohoto způsobu získání tokenu je získání implicitního autorizačního odkazu. Téměř se neliší od odkazu použitého v prvním způsobu získání tokenu, liší se pouze v parametru **response_type** – nedostáváme typ *code*, ale přímo token.

```
/authorize?response_type=to-
ken&client_id=CLIENT_ID&redirect_uri=CALLBACK_URL&scope=read
```

Po prokliknutí se musí uživatel přihlásit (pokud již přihlášen není) a povolit nebo zakázat aplikaci žádající o přístup k jeho účtu. Pokud uživatel povolí aplikaci přístup, služba ho přesměruje na URI aplikace, která také bude obsahovat přístupový token:

```
yyy.com/callback#token=ACCESS_TOKEN
```

User-agent aplikaci přesměruje na výše zmíněnou URI, ale token si uchová. Aplikace poté vrátí webovou stránku obsahující script, který umí získat přístupový token, jenž si user-agent uchoval. User-agent poté spustí script a pošle přístupový token aplikaci. Aplikace je poté autorizovaná a je jí umožněno pracovat se stanovenými právy po dobu platnosti tokenu. [29, 32, 37, 38, 39, 40, 41]



Obrázek č. 14 – OAuth 2 Grants [44]

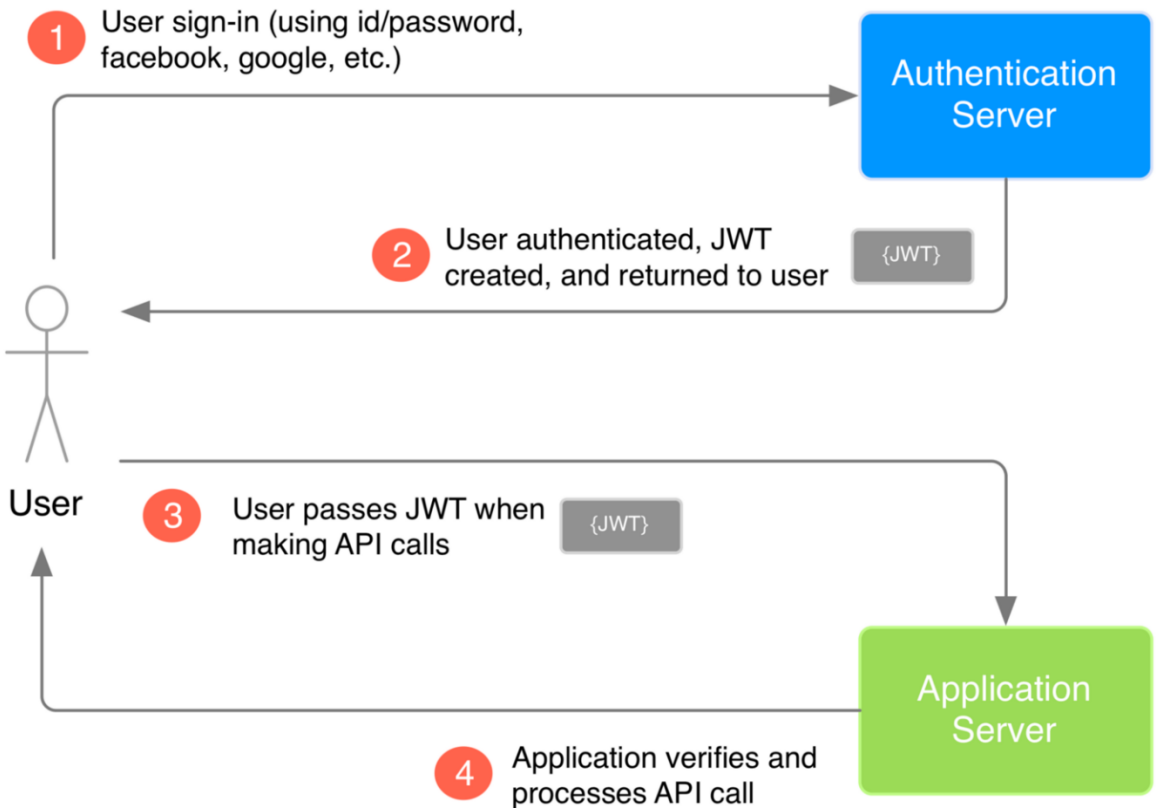
3.2.3 Typy tokenů protokolu OAuth2

Protokol OAuth2 využívá pro autentizaci a autorizaci 2 typy tokenů – Access tokeny – jedná se o tokeny, které nesou informace vedoucí k autentizaci na cílovém serveru, většinou mívají expirační dobu a jsou platné jen krátce - JWT (JSON Web Token) a nebo Refresh tokeny – ty v sobě nesou informace potřebné k vygenerování nového Access tokenu. Také

mají expirační dobu, ale většinou delší než Access tokeny. Bývají uloženy na serveru, který je vydává. Říká se jim Opaque tokeny. [51]

3.2.3.1 JWT

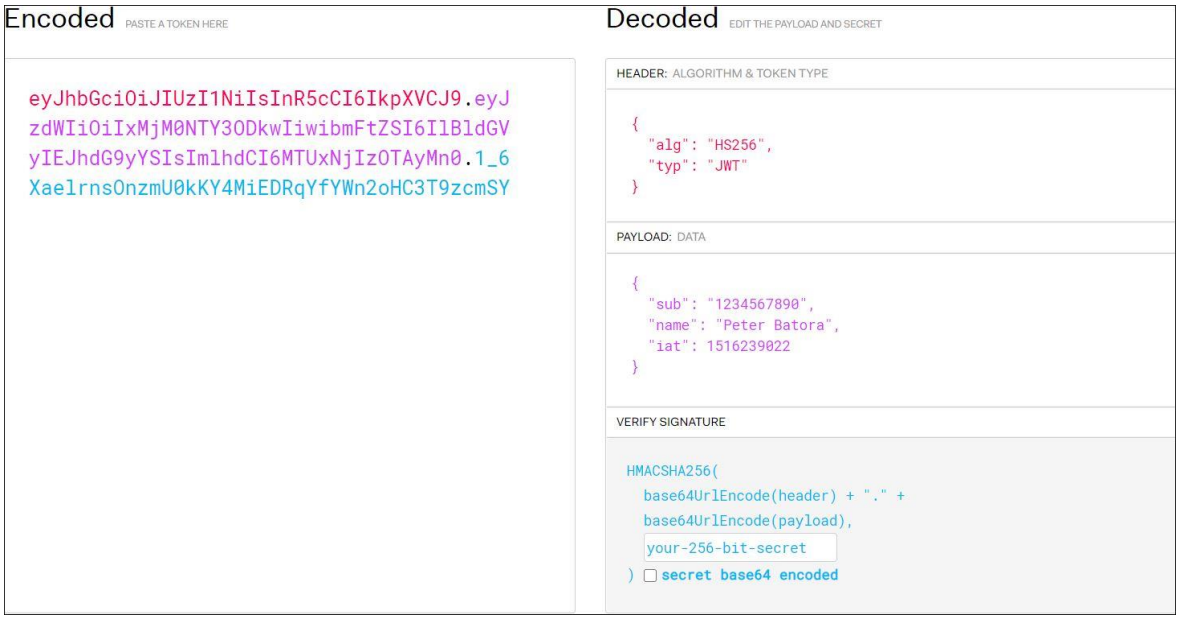
JSON Web Token (JWT) představuje způsob pro **bezpečnou výměnu informací** mezi dvěma stranami. Cílem JWT je možnost **ověření autenticity dat** – skutečnosti, že data nebyla cestou změněna. Nikoli však skrýt obsah dat. Účelem zakódování dat je převod struktury dat do lépe přenositelné podoby.



Obrázek č. 15 – JWT Flow [50]

V tomto případě se uživatel přihlásí do ověřovacího serveru svými přihlašovacími údaji nebo údaji z Facebooku, Googlu atd. Po ověření server vytvoří JWT a odešle ho zpět uživateli. Uživatel odešle požadavek API obsahující JWT. V tomto konkrétním grafu bude aplikační server nakonfigurován, aby si ověřil, že JWT byl vytvořen autorizačním serverem. A po úspěšném ověření pustí uživatele dál.

JWT je **JSON objekt**, který se skládá z **hlavičky** (*header*), **dat** (*payload*) a **podpisu** (*signature*). Podle specifikace RFC 7519. V JSON formátu jsou tyto části odděleny tečkami, ty bývají zakódovány algoritmem Base64 (pojmenováno podle 64 znaků, do kterých šifruje) – který umožní zašifrovat jakýkoliv znak do sady znaků obsahujících klasická velká a malá písmena, čísla, plus a lomítko. [43;45;46;47;48;49]

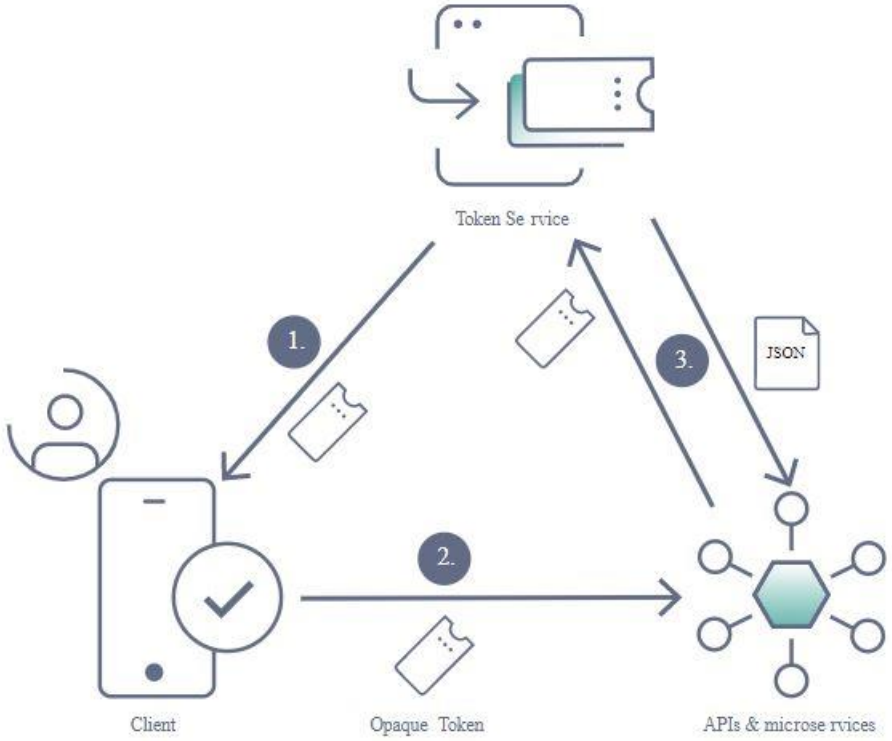


Obrázek č. 16 – JWT Base 64 encoding – Vlastní zpracování dle [50]

Nevýhodou tohoto tokenu je, že nemůže být **revokován** (obnoven) a také, že při odchytnutí tohoto tokenu může útočník dekódovat přenášené informace (např. na veřejné stránce: https://jwt.io/).[48]

3.2.3.2 Opaque token

Opaque (přeloženo jako *neprůhledný*) token se od JWT liší hlavně v tom, že v sobě nenese žádná data a je posílán v takovém formátu, který je pro příjemce nečitelný a tudíž, aby příjemce mohl tento token ověřit, musí zavolat server, který tento token vydal. Nicméně v sobě uchovává identifikátor, odkaz k informacím, uloženým na straně vydávajícího serveru. Další velkou výhodou neprůhlednosti je to, že se tyto tokeny nemusí podepisovat. [43;45;46;52]



Obrázek č. 17 – Opaque token flow [53]

II. PRAKTICKÁ ČÁST

4 TVORBA WEBOVÉ APLIKACE

V této kapitole stručně popíšu pojem webová aplikace a následně také hlavní nástroje a programovací jazyk, který jsem použil při tvorbě svého návrhu. Při návrhu jsem zvolil programovací jazyk Python, protože s ním mám největší zkušenosti ze svého zaměstnání a také má nejjednodušší syntaxi. Pro vývoj webové aplikace jsem ze stejného důvodu použil framework Flask (je to aplikační rámec, který slouží jako podpora při programování – obsahuje potřebné knihovny, utility, programy a také postupy při vývoji).

4.1 Webová aplikace

Pod pojmem webová aplikace (WA) si můžeme představit aplikaci, která je nainstalovaná a poskytovaná z webového serveru přes internet, nebo podnikovou síť(intranet). Hlavní výhodou WA je, že si ji uživatelé nemusí stahovat a instalovat – stačí, když přejdou na konkrétní webovou stránku a hned ji mohou používat. Mezi další výhody patří:

- Vysoká kompatibilita – protože je WA instalovaná na serveru a uživatelé k ní přistupují přes webový prohlížeč, tak nás nezajímá jejich OS,
- jednoduchá realizace aktualizací – stačí aktualizovat WA na serveru a rázem ji mají aktualizovanou všichni uživatelé,
- bezpečnost – stránky i jejich databáze jsou uloženy na serveru, a tak není jednoduché WA ukradnout.

WA se skládá z kombinace statických a dynamických webových stránek. [56;58]

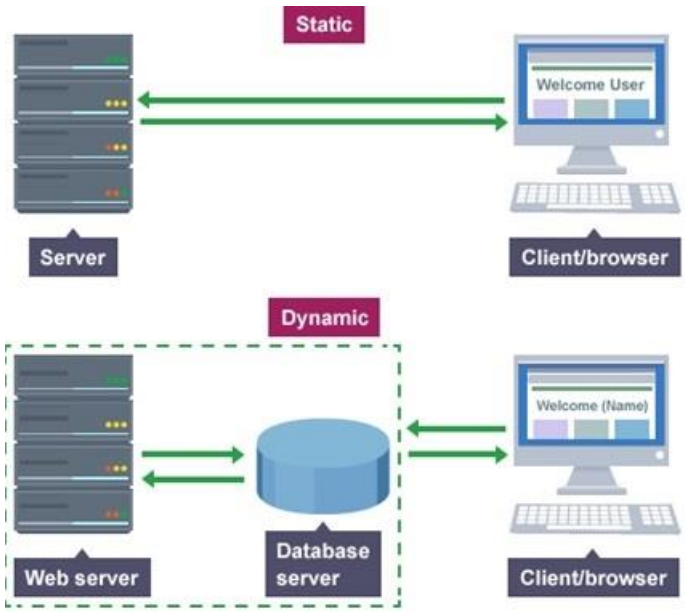
4.1.1 Statické webové stránky

Tento typ webových stránek je obvykle rychlejší, než tomu je u dynamický stránek. Z důvodu, že tyto stránky jsou jednoduše uloženy na serveru jako html soubory. Po dotazu webového prohlížeče server tuto stránku vyhledá a pošle ji zpět webovému prohlížeči. Tyto stránky, jak již lze poznat z jejich názvu, jsou na serverové straně neměnné – nemají žádný dynamický (měnící se) obsah. Ale i na těchto stránkách může být například zavěšený JavaScript, který dokáže u klienta provádět nějaké změny. Pokud bychom obsah těchto stránek chtěli měnit, musíme na server nahrát novou verzi html souboru.

4.1.2 Dynamické webové stránky

Jak jsme si popsali výše, tak při dotazu na statickou stránku server pouze vrátí uloženou webovou stránku, kterou nijak nemění. V případě dynamických stránek je tomu jinak – po dotazu na webový server je požadavek nejdříve předán aplikačnímu/databázovému serveru (v našem případě se jedná o Flask), ten se podle instrukcí v dotazu (př. klient klikl na odkaz týkající se konkrétního modelu auta) připojí do databáze serveru, načte potřebná data a na jejich základě vygeneruje HTML stránku, kterou poté pošle webovému serveru, aby ji vrátil klientovi. Ten sice vidí klasickou statickou stránku, která však byla dynamicky vygenerovaná podle jeho konkrétního požadavku. [54;55]

Pro grafickou představu se můžeme podívat na obrázek níže:



Obrázek č. 18 – Statické a dynamické stránky [54]

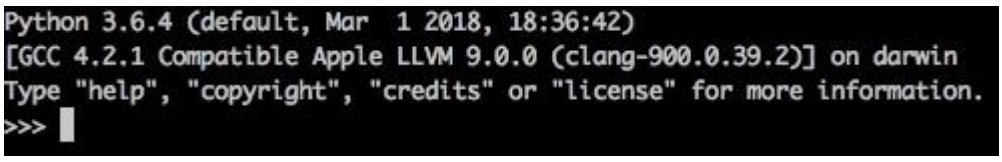
4.2 Python 3

Python je dynamický interpretovaný open-source (jeho zdrojový kód je veřejný) programovací jazyk, který byl vyvinut v roce 1991 Guido Van Rossumem. Obsahuje moduly, výjimky, dynamické psaní, vysokou úroveň dynamických datových typů a tříd (classes). Podporuje také vícero programovacích paradigmat nad rámec OOP (objektově orientovaného programování), jako například procedurální a funkční programování. Jelikož je dynamicky interpretovaný – jeho kód se překládá až za běhu, čímž se případné chyby v kódu objeví až při spuštění programu. Byl navržen tak, aby umožňoval tvorbu od jednoduchých až po plnohodnotné, rozsáhlé aplikace. Obsahuje také velké množství systémových knihoven a modulů. Jednou z hlavních výhod tohoto jazyku je jeho jednoduchost z hlediska učení. Používá jednoduchou a čistou syntaxi – k odsazování se používají mezery nebo tabulátor. Bloky kódu nejsou uzavřeny složenými závorkami, na rozdíl od ostatních programovacích jazyků, a jsou odděleny novým řádkem při stejné úrovni odsazení. Python spolupracuje s dalšími programovacími jazyky, a tudíž má mnoho interpretací:

- CPython – implementace interpretru v jazyce C
- IronPython – implementace interpretru v jazyce .NET/Mono
- Brython – implementace interpretru v jazyce JavaScript
- a další.

Jakožto open-source projekt je také zdarma stažitelný na mnoha platformách, jako například: Windows, Mac, Unix. Python má také vlastní repozitář knihoven **PyPi** – ty se do Pythonu dají nainstalovat package managerem **pip** – příklady použití pipu budou ukázány v návrhu aplikace.

Pro vývoj webových aplikací v jazyce Python se používají frameworky. Mezi nejznámější patří Django nebo Flask. [56;57]



Obrázek č. 19 - Aktuálně nainstalovaná verze Python (Vlastní zpracování)

4.3 Flask

Nabízí se otázka, proč jsem si zvolil Flask místo více propagovanějšího Djanga. Největším rozdílem je, že Django je full-stack framework – zjednodušeně má v sobě mnohem více nástrojů, než je pro naši WA potřeba -> Flask je vlastně microframework, tzn. že se snaží udržet si jednoduché, ale rozšiřitelné jádro. Je teoreticky možné napsat celou aplikaci v jednom souboru. Flask nám nevnucuje žádné své nástroje– např. výběr databáze (ve výchozím stavu ani žádnou databázovou vrstvu neobsahuje), ale naopak nechá na nás, jaké rozšíření přidáme, které se poté chovají tak, jako kdyby již součástí Flasku byly. [58;59]

4.4 SQLAlchemy

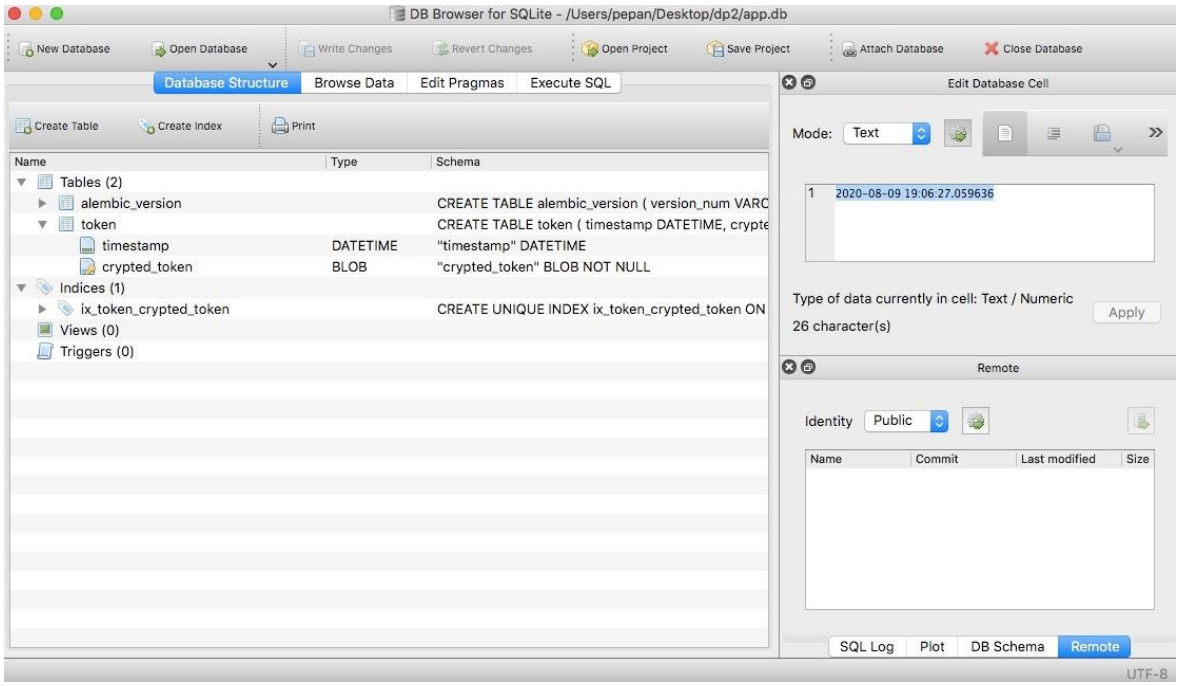
Nicméně pro naše účely jsem potřeboval nějakou databázi pro uložení zašifrovaných tokenů a časových otisků. Databáze se rozdělují do 2 velkých skupin: první skupina respektuje relační modely. Druhá, která je nerespektuje, např. NoSQL – z názvu je patrné, že nemá implementovaný dotazovací jazyk SQL. Já jsem zvolil relační databázový model a rozšíření do Flasku, které se jmenuje Flask-SQLAlchemy – jedná se o Flask-friendly použití knihovny SQLAlchemy. Tato knihovna přidává možnost ORM (Object Relational Mapper). Díky ORM je možné spravovat databázi vysoko-úrovňovými entitami jako např. třídami, objekty a metodami místo tabulek a SQL. Jeho práce je překládat tyto operace na SQL dotazy. Tabulky jsou vytvářeny na základě tříd a jednotlivé proměnné jsou zase sloupce.

```
class Token(db.Model):
    timestamp = db.Column(db.DateTime)
    crypted_token = db.Column(db.LargeBinary, index=True, unique=True, primary_key=True)
```

Obrázek č. 20 – Tabulka Token v interpretaci SQLAlchemy v souboru models.py (Vlastní zpracování)

Ze screenshotu je vidět, že každá proměnná obsahuje třídu Column, která definuje, že se jedná o sloupec. Dále můžeme nastavit o jaký typ sloupce jde, jak bude záznam dlouhý a také lze jednoduše nastavit, která proměnná má být primární klíč atd.

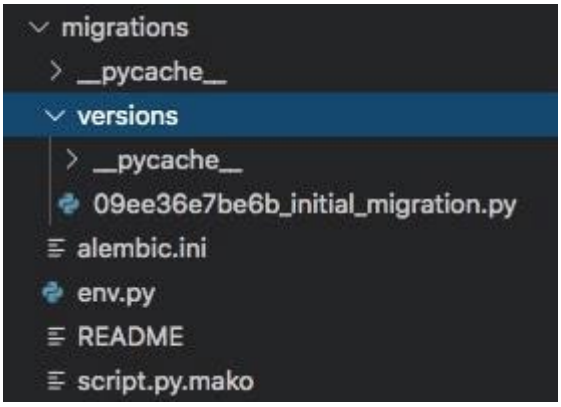
SQLAlchemy podporuje různé typy databází (MySQL, PostgreSQL a SQlite). Já jsem použil poslední jmenovaný typ databáze SQlite, z důvodu, že není nutné instalovat databázový server – databáze je v jednom souboru *app.db*. Pokud člověk chce s DB pracovat v grafickém režimu, tak se dá nainstalovat aplikace DB Browser for SQLite. [60;61]



Obrázek č. 21 – Grafické rozhraní SQLite DB Browseru (Vlastní zpracování)

4.5 Alembic

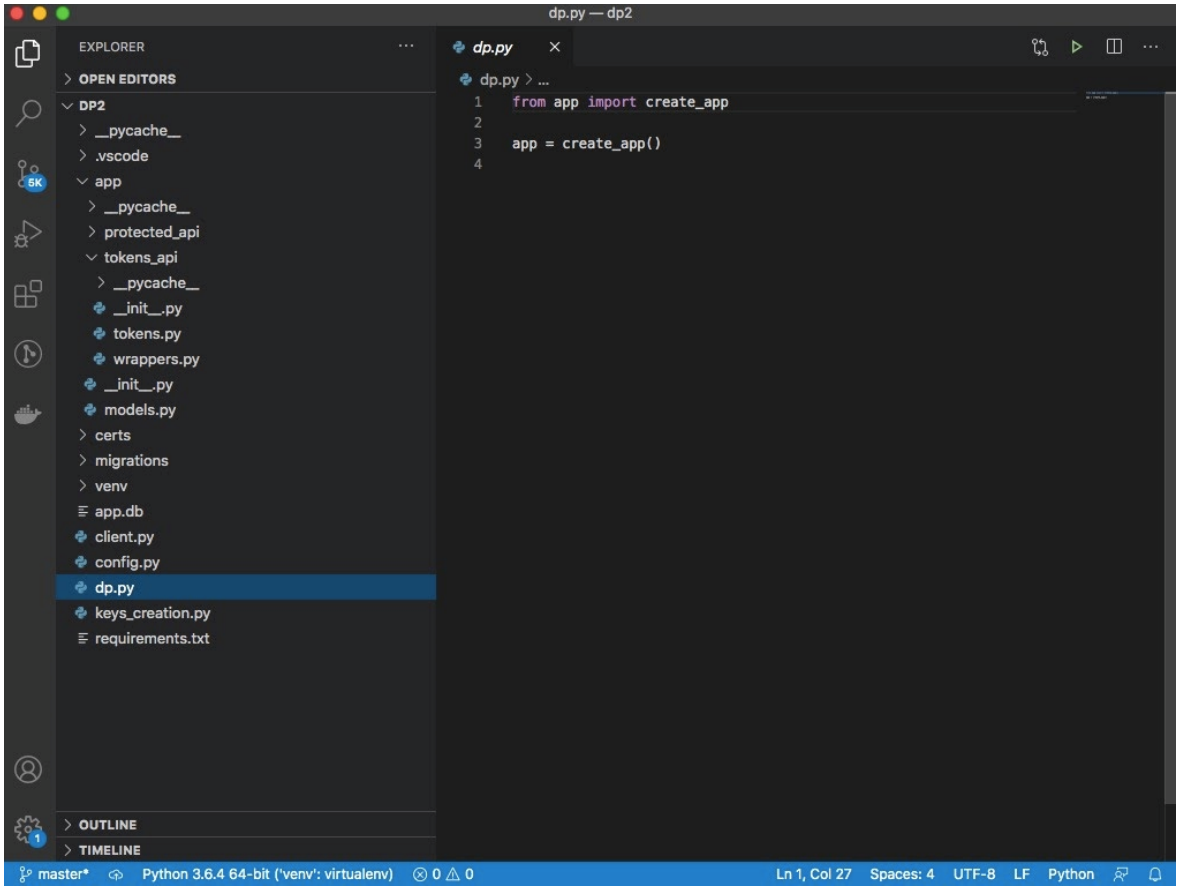
Rozhodl jsem se také použít databázový migrační framework SQLAlchemy, který se jmenuje Alembic. Rozšíření pro Flask se nazývá *flask-migrate*. V podstatě se jedná o velmi silný nástroj, který řeší problémy s aktualizováním relační databáze. Často se stává, že při vytváření nové tabulky v DB nebo při změně již vytvořené, se může přihodit chyba a my přijdeme o již vytvořená data. Alembic toto řeší tak, že konkrétními příkazy umí zjisti změny v modelech. Prvním příkazem, který mi vytvoří novou složku **migrations** a **migrations/versions** a samozřejmě další soubory nutné k fungování tohoto rozšíření, je: *flask db init*. Po spuštění dalšího důležitého příkazu *flask db migrate -m „<text změny>“*se ve složce **migrations/versions** vytvoří nová verze migrace, která obsahuje nalezené změny. Nasazení této změny do naší databáze se provede příkazem: *flask db upgrade*. Naopak, pokud si budeme přát tuto změnu vrátit, stačí použít příkaz: *flask db downgrade*. [62;63]



Obrázek č. 22 – Stromová struktura migrací (Vlastní zpracování)]

4.6 Visual Studio Code

K tvorbě WA jsem také použil editor kódu Visual Studio Code. Tento bezplatný editor byl vyvinutý společností Microsoft a je podporován na všech OS (macOS, Linux a samozřejmě Windows). Důvodem, proč jsem si tento konkrétní editor vybral byla možnost využití různých vývojářský funkcí jako např. debuggování (zachycení a oprava chyb). Velkou výhodou je také možnost nainstalování si libovolných rozšíření, v mém případě interpreteru Pythonu. [64]



Obrázek č. 23 – Grafické rozhraní Visual Studia (Vlastní zpracování)

4.7 Tvorba webové aplikace

V této kapitole stručně popíšu, jak jsem vytvářel svoji WA. Ukážu některé základní příkazy Pythonu a Flasku, které jsem při tvorbě použil a také hlavní soubory, které jsou nutné k fungování flaskové WA.

4.7.1 Nastavení virtuálního prostředí

I když všechny instalace macOSu přicházejí s již předinstalovaným Pythonem, tak je doporučeno na lokální instalaci nechytat a raději si vytvořit virtuální prostředí (VP). Začíná se jednoduše, a to vytvořením si kořenové složky, ve které chceme VP vytvořit. V mém případě jde o složku **dp2**, druhý zobrazený příkaz slouží ke přepnutí se do této složky:

```
Btch:Desktop pepan$ mkdir dp2
Btch:Desktop pepan$ cd dp2/
```

Obrázek č. 24 – Příkazy k vytvoření a přepnutí se do složky (Vlastní zpracování)

Následuje vlastní vytvoření virtuálního prostředí. Z obrázku je patrné, že se nám do vybrané složky nainstaluje spustitelný Python ve verzi 3.6.

```
Btch:dp2 pepan$ virtualenv venv
Using base prefix '/usr/local/Cellar/python/3.6.4_3/Frameworks/Python.framework/Versions/3.6'
New python executable in /Users/pepan/Desktop/dp2/venv/bin/python3.6
Also creating executable in /Users/pepan/Desktop/dp2/venv/bin/python
Installing setuptools, pip, wheel...
done.
```

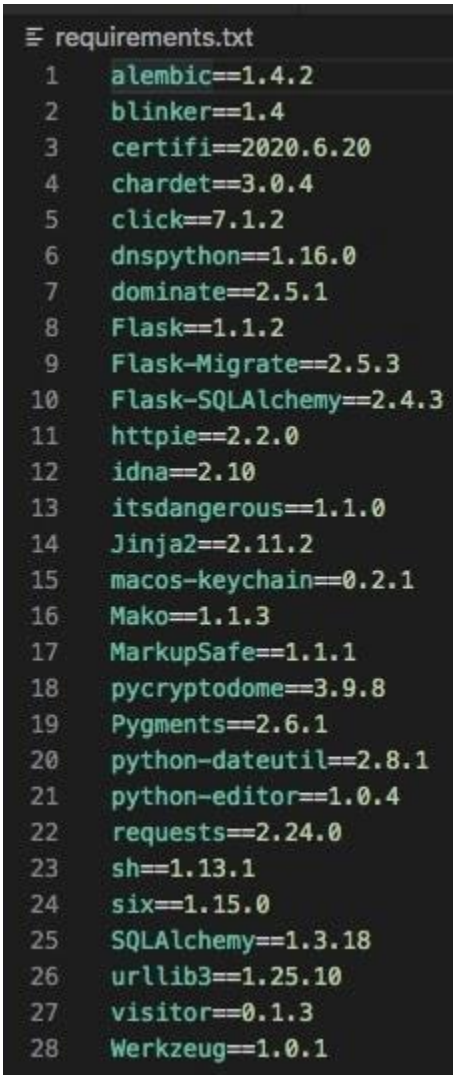
Obrázek č. 25 – Instalace VP (Vlastní zpracování)

Dále je potřeba nainstalovat si potřebné balíčky (knihovny, packages), které jsou nezbytné pro zajištění správné funkčnosti WA. K tomu nám slouží příkaz:

```
Btch:dp2 pepan$ ./venv/bin/pip install -r requirements.txt
```

Obrázek č. 26 – Instalace nezbytných balíčků (Vlastní zpracování)

Při pohledu na výše zmíněný příkaz si všimneme, že se balíčky instalují ze souboru **requirements.txt**. Do tohoto souboru se zapisují všechny nainstalované balíčky v konkrétní flaskové WA. A slouží k tomu, aby kdokoliv, kdo bude mít naklonovaný repozitář (SW uložiště pro SW balíčky) měl stejnou verzi všech knihoven jako vývojář. V mém případě soubor vypadá takto:



Obrázek č. 27 – Obsah requirements.txt (Vlastní zpracování)

Dalšími neméně důležitým příkazy a soubory jsou: *config.py* – soubor, ve kterém jsou uloženy parametry a nastavení nutné pro inicializování WA. Na to přímo navazuje *__init__.py* – tento soubor má za funkci vytvořit naši WA, inicializuje databázi a registruje blueprinty (soustava operací, která může být použitá vícekrát a která se může registrovat na konkrétní URL). Soubor *models.py*, ve kterém máme třídy SQLAlchemy. A v neposlední řadě také soubor *dp.py*, který spouští celou aplikaci. [65]

dp.py > ...

```python
from app import create_app

app = create_app()
```

Obrázek č. 28 - Obsah souboru dp.py (Vlastní zpracování)

config.py > ...

```python
import os
basedir = os.path.abspath(os.path.dirname(__file__))

class Config(object):
    SECRET_KEY = os.environ.get('SECRET_KEY') or 'you-will-always-guess'
    SQLALCHEMY_DATABASE_URI = os.environ.get('DATABASE_URL') or \
        'sqlite:///' + os.path.join(basedir, 'app.db')
    SQLALCHEMY_TRACK_MODIFICATIONS = False
    DEBUG = True
```

Obrázek č. 29 – Obsah souboru config.py (Vlastní zpracování)

app > __init__.py > {} models

```python
import os
from flask import Flask, current_app
from config import Config
from flask_sqlalchemy import SQLAlchemy
from flask_migrate import Migrate

db = SQLAlchemy()
migrate = Migrate()


def create_app(config_class=Config):
    app = Flask(__name__)
    app.config.from_object(config_class)

    db.init_app(app)
    migrate.init_app(app, db)

    from app.tokens_api import bp as tokens_api_bp
    app.register_blueprint(tokens_api_bp, url_prefix='/tokens')

    from app.protected_api import bp as protected_api_bp
    app.register_blueprint(protected_api_bp, url_prefix='/api')

    return app

from app import models
```

Obrázek č. 30 – Obsah souboru __init__.py (Vlastní zpracování)

5 VLASTNÍ TVORBA A POUŽITÍ BEZPEČNOSTNÍHO TOKENU

V této kapitole praktické části se podíváme na mé řešení tvorby bezpečnostního tokenu a jeho následné použití v nově vytvořené WA. Vývoj a testování proběhlo na operačním systému **macOS High Sierra 10.13.6** a pro lokální ukládání tokenu na klientské straně využívám macovské klíčenky – **Keychain Access**. Dále je také důležité říct, že pro produkční verzi není možné použít flaskový server, ale například Nginx. Samozřejmě se počítá s tím, že komunikace probíhá jenom po SSL kanále (proti odchycení headeru se zašifrovaným tokenem) - API, které by toto používalo musí být provozováno pouze na HTTPS (port 443), pokud by přišel request (dotaz) na HTTP (port 80), tak musí být vynucen redirect (přesměrování) na HTTPS.

5.1 Tvorba tokenu

Na základě analýzy aktuálně využívaných tokenů a jejich šifrování jsem přišel s nápadem k přidání časového otisku v době tvorby tokenu (modul **datetime**) a následné připojení k náhodně vygenerovanému 32místnému stringu (řetězci), obsahující kombinaci velkých písmen a čísel. Pro generování těchto náhodných znaků jsem nepoužil funkci **Random**, která je součástí Pythonu, ale importoval jsem si knihovnu **Secrets**. **Random** je vlastně pseudonáhodný generátor – točí se kolem tzv. seed čísla, které je většinou založeno na systémových hodinách a tím pádem se dá vygenerovat stejná sekvence čísel. Naopak modul **Secrets** používá jako seed nějaké číslo, které je nemožné algoritmicky reprodukovat – jedná se například o klávesové vstupy, pohyb myši, počet spuštěných programů, vytížení procesoru atd. Takto spojený string jsem pojmenoval ***unhashed_token***. Hashovnání probíhá funkcí **sha3_512** (naprogramovaná v knihovně **hashlib**), kterou zašifruji proměnnou ***unhashed_token***, který je ale navíc obrácený zprava doleva. **Sha3_512** byla zvolena z důvodu, že se jedná o nejnovější hashovací funkci s collision resistancí o 256 bitech a délkou 512 bitů. Těmito parametry by měla být v současné době neprolomitelná. Nově zahashovaný token je poté zakličován veřejným klíčem serveru a uložen v databázi. Na screenshotu níže se můžeme podívat, jak vypadá postup tvorby zahashovaného tokenu před zašifrováním.

```
>>> import string, hashlib, datetime, secrets
>>> now = datetime.datetime.now()
>>> print(now)
2020-08-11 16:22:38.489782
>>> secret_string = ''.join(secrets.choice(string.ascii_uppercase + string.digits) for x in range (32))
>>> print(secret_string)
Y0FRYPQITEHX2H2BZM2WCMUX2EOG1J7F
>>> unhashed_token = secret_string + str(now.timestamp())
>>> print(unhashed_token)
Y0FRYPQITEHX2H2BZM2WCMUX2EOG1J7F1597155758.489782
>>> hashed_token = hashlib.sha3_512(unhashed_token[::-1].encode()).digest()
>>> print(hashed_token)
b'l\xdb,]\x87(R5\xc5\xe9\xb8t\xd3\x0788\x88\x0c\xfb@)\x939\xfah\xf6\xa9\xce\xa2A\xab\x03\x12\xbeI\x02\xaa\xb7\xfcg\xa9\x86\x89\x85\xccqkN\xde6_6\x05a\xab\xca\xb8\xb9\x8c\xb8HK\xa7y'
>>>
```

Obrázek č. 32 – Tvorba zahashovaného tokenu (Vlastní zpracování)

Samotná tvorba probíhá na serverové straně v souboru *models.py*. Po vytvoření onoho zahashovaného tokenu dojde k načtení veřejného klíče serveru **pu_key**, který byl již předem vygenerován scriptem *keys_creation.py* – je nutné tento script jednorázově spustit před testování funkčnosti tokenu. Z veřejného klíče se poté vytvoří šifra **cipher**, kterou následné zašifrujeme **hashed_token** a ten poté uložíme do databáze i s časovým razítkem **now**.

app > models.py > ...

```python
from flask import current_app
import hashlib, string, secrets, datetime, base64
from app import db
from Crypto.Cipher import PKCS1_OAEP
from Crypto.PublicKey import RSA

### Přiřazení cesty k veřejnému klíči serveru do constanty
KEY_URL = 'certs/server/public_server.pem'


class Token(db.Model):
    timestamp = db.Column(db.DateTime)
    crypted_token = db.Column(db.LargeBinary, index=True, unique=True, primary_key=True)

    def __init__(self):
        ### Tvorba zahashovaného tokenu
        now = datetime.datetime.now()
        secret_string = ''.join(secrets.choice(string.ascii_uppercase + string.digits) for x in range (32))
        unhashed_token = secret_string + str(now.timestamp())
        hashed_token = hashlib.sha3_512(unhashed_token[::-1].encode()).digest()

        ### Načtení vygenerovaného veřejného klíče serveru
        pu_key = RSA.import_key(open(KEY_URL, 'r').read())

        ### Vytvoření šifry z veřejného klíče
        cipher = PKCS1_OAEP.new(key=pu_key)

        ### Zašifrování hashed_tokenu vytvořenou šifrou a následné uložení do DB i s časovým otiskem
        self.crypted_token = cipher.encrypt(hashed_token)
        self.timestamp = now
        db.session.add(self)
```

Obrázek č. 31 - Obsah souboru models.py (Vlastní zpracování)

Na obrázku č. 33 vidíme obsah souboru *keys_creation.py*, ve kterém je napsána logika tvorby veřejným a soukromých klíčů jako pro server, tak pro klienta.

keys_creation.py > ...

```python
#Importování nezbytných modulů
from Crypto.PublicKey import RSA

#Vygenerování soukromého klíče pro server (klíč je RsaKey objekt) o délce 2048 bitů
private_key_server = RSA.generate(2048)

#Vygenerování soukromého klíče pro klienta (klíč je RsaKey objekt) o délce 4096 bitů
private_key_client = RSA.generate(4096)

#Vygenerování veřejného klíče pro server (klíč je RsaKey objekt) ze soukromého klíče serveru
public_key_server = private_key_server.publickey()

#Vygenerování veřejného klíče pro klienta (klíč je RsaKey objekt) ze soukromého klíče klienta
public_key_client = private_key_client.publickey()

#Konvertování RsaKey objektů na string
private_server_pem = private_key_server.export_key().decode()
public_server_pem = public_key_server.export_key().decode()
private_client_pem = private_key_client.export_key().decode()
public_client_pem = public_key_client.export_key().decode()

#Zapsání veřejného a soukromého servrového klíče do  'pem' souborů
with open('certs/server/private_server.pem', 'w') as private_server:
    private_server.write(private_server_pem)
with open('certs/server/public_server.pem', 'w') as public_server:
    public_server.write(public_server_pem)

#Zapsání veřejného a soukromého klientského klíče do  'pem' souborů
with open('certs/client/private_client.pem', 'w') as private_client:
    private_client.write(private_client_pem)
with open('certs/client/public_client.pem', 'w') as public_client:
    public_client.write(public_client_pem)
```

Obrázek č. 33 – Tvorba klíčového páru ze souboru keys_creation.py (Vlastní zpracování)

Obrázek č. 34 – Nově vytvořené klíče (Vlastní zpracování)

5.2 Server side

Ve své WA jsem také vytvořil několik API endpointů (koncových bodů) v souborech *tokens.py* a v *protected_api.py*. Jedná se o API pro získání, revokování (zrušení), kontrolu a obnovu aktivního tokenu, které se volají HTTP dotazy.

app > tokens_api > tokens.py > ...

```python
from app import db
from app.tokens_api import bp
from app.tokens_api.wrappers import token_auth
from app.models import Token
import base64

### Endpoint pro získání tokenu
@bp.route('/', methods=['GET'])
def get_token():
    crypted_token = Token()
    db.session.commit()
    return base64.b64encode(crypted_token.crypted_token)

### Endpoint pro revokování tokenu
@bp.route('/', methods=['DELETE'])
@token_auth
def revoke(token):
    db.session.delete(token)
    db.session.commit()
    return "Revoked"

### Endpoint pro obnovu tokenu
@bp.route('/renew', methods=['POST'])
@token_auth
def renew(token):
    db.session.delete(token)
    crypted_token = Token()
    db.session.commit()
    return base64.standard_b64encode(crypted_token.crypted_token)
```

Obrázek č. 35 – Obsah souboru tokens.py (Vlastní zpracování)

První z endpointů – endpoint pro získání tokenu, na obrázku č. 35 se nachází na routě ´/**tokens**/´ a reaguje na metodu **GET**. Do proměnné **crypthed_token** načteme třídu **Token()** importovanou z *models.py*. Ta má v sobě definovanou speciální metodu **__init__** (jak můžeme vidět na obrázku č. 32). Ta je speciální v tom, že se vždycky spustí při nově vytvořené instanci třídy. Tím pádem se tedy vytvoří token, který se nakonec vrátí zakódovaný algoritmem Base64. Hlavním důvodem tohoto kódování je výsledný formát tokenu, a tedy možnost jednoduchého posílání. Dalším API je revoke (smazání) platného tokenu. Také se nachází na routě ´/**tokens**/´, ale liší metodou – **DELETE**. U tohoto endpointu si můžeme všimnout rozdílu proti tomu minulému – jedná se o dekorátor (funkce, která vrací jinou funkci) **@token_auth**, který byl importován ze souboru *wrappers.py*:

app > tokens_api > wrappers.py > ...

```python
from flask import request, abort
from app.models import Token
import datetime, base64
from functools import wraps

EXPIRATION = 600

def token_auth(f):
    @wraps(f)
    def wrapper(*args, **kwargs):
        pu_crypted_token = base64.b64decode(request.headers.get('Authorization'))
        token = Token.query.filter_by(crypted_token=pu_crypted_token).one_or_none()

        if token and (token.timestamp + datetime.timedelta(seconds=EXPIRATION) > datetime.datetime.now()) :
            kwargs['token'] = token
            return f(*args, **kwargs)
        else:
            return abort(403)
    return wrapper
```

Obrázek č. 36 – Obsah souboru wrappers.py (Vlastní zpracování)

V tomto souboru mám definovanou konstantu **EXPIRATION** s číselnou hodnotou 600 – ta je později přiřazena parametru **seconds** ve funkci **timedelta.** Hlavní funkcí tohoto dekorátoru je ověření, že přijímaný token, který posílá klient http dotazem v headeru, se shoduje s tokenem, který je uložený v DB serveru a že je pořád platný. Prvním krokem je získání dekódovaného tokenu funkcí **request**. Tato funkce z headeru dotazu vytáhne přidělenou hodnotu klíči **Authorization**, která se rovná našemu tokenu (konkrétní použití bude ukázáno v dalších kapitolách DP). Dalším krokem je vyhledání obdrženého tokenu pomocí dotazů do DB. Všimněme si metody **one_or_none()**. Ta by měla vrátit právě jeden nebo žádný token uložený v DB serveru. Pokud takový token v DB není, server vrátí chybu **403** a funkce končí. Pokud ale takový token existuje, musí se ještě ověřit, zda je pořád aktivní. Toto řeší další podmínka, která sčítá časový otisk **token.timestamp** a funkci **timedelta** (životnost tokenu – 10minut). Toto číslo poté musí být větší než aktuální čas. Pokud není, tak víme, že životnost tokenu vypršela, a tudíž je neplatný. V tomto případě také obdržíme chybu **403**. Naopak, pokud je pořád aktivní předáme token parametru **kwargs**, abychom s ním mohli pracovat na našich API.

Pokud nám tedy dekorátor **@token_auth** předal platný token a je možné ho revokovat (smazat). To zařídí příkaz **db.session.delete(token),** ten se následně musí potvrdit **db.session.commit()** a poté server odpoví slovem „Revoked“ a náš token je smazaný z DB.

Posledním endpointem ze souboru tokens.py je endpoint na obnovu platného token (renew). Nachází se na routě ´**/tokens/renew**´ a pracuje s metodou **POST**. Hlavní rozdílem proti

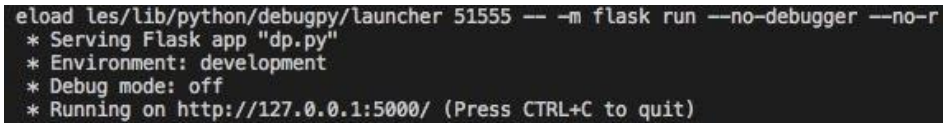
předchozí funkci revoke je to, že pokud nám dekorátor vrátí, že je token pořád platný, tak ho renew smaže a vytvoří nový.

Vytvořil jsem ještě poslední endpoint pro kontrolu/ověření platnosti odesílaného tokenu. Logiku jsem zapsal do jiného souboru a to *protected_api.py* z toho důvodu, že tento endpoint už by měl sloužit pro autentizované klienty. Také zde využívám výše zmíněného dekorátoru @token_auth. API se nachází na routě ´**/api/check**´. V současné době se autentizovaným klientům pouze zobrazí hláška: „Token je aktivni“.

Obrázek č. 37 – Obsah souboru protected_api.py (Vlastní zpracování)

5.3 Client side

Pro praktickou demonstraci jsem vytvořil script se jménem *client.py*, kde jsou shrnuty všechny výše popsané funkčnosti. Takto funguje nově vytvořený protokol na klientské straně. Je samozřejmě nutné, aby byl zapnutý server, na kterém nová flasková aplikace běží:



Obrázek č. 38 – Spuštěná WA (Vlastní zpracování)

client.py > ...

```python
import base64, macos_keychain, requests
from Crypto.Cipher import PKCS1_OAEP
from Crypto.PublicKey import RSA

SERVER_URL = 'http://127.0.0.1:5000'
PUBLIC_KEY_URL = 'certs/client/public_client.pem'
PRIVATE_KEY_URL = 'certs/client/private_client.pem'

### Generování tokenu ze serveru a následné uložení zaklíčovaného tokenu veřejným klíčem klienta do klíčenky

### Získání zakódovaného tokenu ze serveru
dotaz_na_token = requests.get('{}/tokens/'.format(SERVER_URL))
print('----------------')
print('Token ze serveru')
print('----------------')
print(dotaz_na_token.content)

### Dekodování tokenu
decoded_token = base64.b64decode(dotaz_na_token.content)
print('----------------')
print('Dekodovany token ze serveru')
print(decoded_token)
print('----------------')

### Načtení veřejného klíče klienta, vytvoření šifrovacího algoritmu a následné šifrování tokenu
pu_client_key = RSA.import_key(open(PUBLIC_KEY_URL, 'r').read())
cipher = PKCS1_OAEP.new(key=pu_client_key)
crypthed_token = cipher.encrypt(decoded_token)
print('Token ze serveru zasifrovany verejnym klicem klienta')
print('----------------')
print(crypthed_token)
print('----------------')

### Přidání zaklíčovaného tokenu do lokálního uložiště(klíčenky) v zakódovaném formátu
macos_keychain.add(name='token', value=base64.b64encode(crypthed_token))
print('Token ze serveru zasifrovany verejnym klicem klienta a  zakodovany base64 a ulozeny v klicence klienta')
print('----------------')
print(macos_keychain.get(name='token'))
print('----------------')

### Rozšifrování uloženého tokenu privátním klíčem klienta
```

Obrázek č. 39 - Obsah souboru client.py 1. část (Vlastní zpracování)

V první části souboru se importují potřebné knihovny a nastavují se konstanty (URL serveru a cesty ke klíčům). Dále odesílám dotaz na token na API **/tokens/**, jehož výsledkem je nově vygenerovaný token ze serveru. Tento token je potřeba dekódovat, protože byl odeslán v zakódovaném formátu. Následně načítám veřejný klíč klienta **pu_client_key**, vytvářím šifrovací algoritmus **cipher** a klíčuji. Takto zašifrovaný token lokálně ukládám do macOSové klíčenky. Musím ho ale uložit v jiném formátu než **bytes** a proto volím opět kódování Base64.

```
### Načtení soukromého klíče klienta, vytvoření dešifrovacího algoritmu a načtení tokenu z klíčenky
pr_client_key = RSA.import_key(open(PRIVATE_KEY_URL, 'r').read())
decrypt = PKCS1_OAEP.new(key=pr_client_key)
crypthed_token = base64.b64decode(macos_keychain.get(name='token'))

### Vymazání tokenu z klíčenky
macos_keychain.rm(name='token')
print('Decodovany token z klicenky klienta zasifrovany verejnym klicem klienta')
print(crypthed_token)
print('----------------')

### Dešifrování tokenu
decrypted_message = decrypt.decrypt(crypthed_token)
print('Desifrovany token klientskym klicem')
print('----------------')
print(decrypted_message)

### Zakódování dešifrovaného tokenu
encoded_decrypted_message = base64.b64encode(decrypted_message)
print('----------------')
print('Encodovany decryptovany token klientskym klicem')
print('----------------')
print(encoded_decrypted_message)

### Porovnání, zda výchozí vygenerovaný token ze serveru se shoduje s dešifrovaným tokenem klienta
if dotaz_na_token.content == encoded_decrypted_message:
    print('----------------')
    print('sedi')
else:
    print('----------------')
    print('nesedi')

### CHECK - dotaz na ověření, zda je token aktivní
headers = {'Authorization': encoded_decrypted_message}
check_req = requests.get('{}/api/check'.format(SERVER_URL), headers=headers)
print('----------------')
print(check_req.content)

### RENEW - dotaz na obnovení tokenu (pokud je token aktivní, tak se smaže z DB a vygeneruje se nový)
headers = {'Authorization': encoded_decrypted_message}
renew_req = requests.post('{}/tokens/renew'.format(SERVER_URL), headers=headers)
print('----------------')
print(renew_req.content)
```

Obrázek č. 41 – Obsah souboru clients.py 3. část (Vlastní zpracování)

```
### REVOKE - dotaz na zrušení aktivního tokenu
headers = {'Authorization': renew_req.content}
revoke_req = requests.delete('{}/tokens/'.format(SERVER_URL), headers=headers)
print('----------------')
print(revoke_req.content)
```

Obrázek č. 40 – Obsah souboru clients.py 2. část (Vlastní zpracování)

Druhá část souboru řeší načtení soukromého klíče klienta **pr_client_key** vytvoření dešifrovacího algoritmu **decrypt** a následně načítá dekódovaný token z klíčenky. V dalším kroku ho mažu z klíčenky a dešifruji ho. Nakonec takto zašifroavný token musím zakódovat, aby byl ve stejném formátu jako v okamžiku, kdy jsem ho na začátku obdržel od serveru.

Následuje jednoduchá porovnávací podmínka, která porovná původní obdržený token s tokenem, který prošel procesem zašifrování. Pokud se shodují, tak se jednoduše vypíše „sedi“, pokud ne, tak „nesedi“.

Poslední částí se testují všechny vytvořené API (check, renew a revoke). Jediná změna proti serverové straně je ta, že bylo nutné definovat header a přidat do něj testovací token. Výsledky spuštěného souboru si můžeme prohlédnout v Příloze P II.

6 VYHODNOCENÍ TOKENŮ

V této kapitole porovnáme námi vytvořený token s již existujícími tokeny JWT a SAML Assertion

6.1 Porovnání tokenů

Navržený token vlastně bere výhody od JWT a SAML Assertion a snaží se vyhnout jejich nevýhodám. Nový token se dá na rozdíl od JWT jednoduše revokovat – nemusí se generovat další token. Je mnohem menší než SAML – token je v bitové podobě a pro přenos se kóduje base64 – odpadá použití striktního XML formátu. Dále také již v základu podporuje šifrování klíčovými páry (asymetrické šifrování). Využívá nejnovějších hashovací algoritmů – SHA3-512. Na rozdíl od JWT nenese žádná data. Hlavním polem působnosti SAML Assertion a postupně i JWT je podnikové prostředí (Enterprise SSO). Ano, JWT vznikl hlavně pro sociální sítě, ale postupně se snaží dostat i do podnikového prostředí. Nový token je podobně jako JWT o dost jednodušší než SAML Assertion, u kterého nebyla jednoduchost cílem – počítalo se s tím, že se budou vytvářet nové knihovny za pochodu. Výchozím zabezpečením SAML Assertion je XML encryption a u JWT je to HS256 (HMAC, typ autentizačního kódu zprávy, který používá hashovací funkci v kombinaci s tajným šifrovacím klíčem + hashovací algoritmus SHA256). V mém případě je bezpečnost řešena hashovací funkcí SHA3_512, která hashuje náhodně generovanou kombinaci 32 velkých písmen a čísel, ke kterým je ještě přidán časový otisk v době tvorby tohoto tokenu. Celý tento nezahashovaný token je ještě zašifrovaný veřejným klíčem a poté je uložen v DB.

6.2 Možnosti rozšíření protokolu v budoucnu

Při dalším rozvíjení tohoto autentizačního protokolu v budoucnu by se dal vylepšit implementací tzv. nepřenositelnosti tokenu. Jednalo by se třeba o vazbu na hardware, software nebo geolokaci uživatele, pro kterého je bezpečnostní token generován.

ZÁVĚR

Cílem této diplomové práce bylo rozebrání a nastudování problematiky autentizace pomocí tokenů, popsání aktuálně využívaných principů. Ale hlavně navržení autentizačního protokolu s využitím bezpečnostních tokenů. Nové řešení mělo být implementované ve vhodném programovacím prostředí. A také výsledky řešení a implementace měly být vhodně reprezentovány.

Práce byla rozdělena do dvou hlavních částí – na část teoretickou a část praktickou. V první kapitole teoretické části diplomové práce jsem čtenáře seznámil s kryptologií, jejími základními pojmy a cíli. Zmínil jsem také problematiku šifrování a její dělení na šifrování symetrické a asymetrické. Neopomněl jsem také vyjmenovat a popsat nejznámější typy obou šifrování. Další kapitolou byla autentizace. Tu dělíme na autentizaci znalostí, vlastnictvím a autentizaci vlastností. Každý typ jsem se snažil objasnit. Hlavní kapitolou teoretické části byla kapitola zabývající se autentizačními protokoly. Popsal jsem dva nejčastější a nejpoužívanější protokoly a to SAML 2 a OAuth 2. U obou protokolů jsem popsal proces získání bezpečnostních tokenů. Také jsem popsal vlastnosti těchto tokenů.

Následuje praktická část, kde první kapitola řeší tvorbu webové aplikace, ve které bylo mnou navržené řešení implementováno. Na začátku jsem vysvětlil, co vůbec webová aplikace je a jaké má výhody proti statickým webovým stránkám. Dále jsem vyjmenoval nástroje, které jsem pro tvorbu využil a také jsem stručně popsal, proč jsem si je vybral. Hlavní důvodem výběru těchto konkrétních nástrojů byla zkušenost z mého zaměstnání. Další částí byla již konkrétní popsání tvorby webové aplikace. Popisoval jsem zde základní příkazy a soubory, které jsou nezbytné pro fungování. Nyní se dostáváme k nejdůležitější kapitole této diplomové práce, a to k tvorbě a použití bezpečnostního tokenu. Na začátku této kapitol jsem uvedl důležité informace, že tento protokol byl vyvíjen na počítači s operačním systémem macOS a využívám jeho nativní aplikace klíčenky Keychain Access. Z analýzy aktuálně používaných autentizačních protokolů a jejich bezpečnostních tokenů jsem zjistil, že jejich hlavním rozdílem je samotný token, jeho tvorba, formát, zabezpečení a jeho vlastnosti. S těmito poznatky jsem přišel s nápadem vytvářet token jako kombinaci náhodně generovaných velkých písmen a čísel v celkovém počtu 32 znaků, a navíc k tomu připojit časový otisk okamžiku tvorby tokenu ve formátu uplynutých sekund od tzv. epochy neboli od 1.1.1970. Takto vygenerovaný token ještě obracím zprava do leva a hashuji nejnovějším hashovacím algoritmem SHA3_512. Před uložením do databáze se takto

zahashovaný token ještě zašifruje veřejným klíčem. Dále v této kapitole ještě popisuji logiku na serverové a klientské straně. Na straně serveru mám vytvořených pár API endopointů, které generují tento token, ověřují platnost dotazovaného tokenu, obnovují ho anebo ho umí revokovat (zrušit). Následuje podkapitola týkající se klientské strany. Zde jsem vytvořil skript, který veškerou funkcionalitu testuje a do terminálu zapisuje výsledky (výsledky tohoto skriptu jsou uvedeny v Příloze P II.).

V závěrečné kapitole porovnávám nově vytvořený token s tokeny SAMLu 2 a OAuthu 2. Docházím k závěru, že mnou navržený token se snaží vyhnout největším nevýhodám těchto tokenů, a naopak se snaží uplatnit jejich výhody. Konkrétně jde o možnost revokování platného tokenu, menší velikost. Můj token také nenese žádná data na rozdíl od tokenu protokolu OAuth 2. A také je o dost jednodušší. Nezapomínám také na zabezpečení nově vytvořeného tokenu, které využívá nejnovějších hashovacích algoritmů. V poslední části potom přemýšlím nad možností vylepšení protokolu v budoucnu, a to minimálně tím, že by bylo dobré naprogramovat tzv. nepřenositelnost tokenu – vazbu na software, hardware nebo geolokaci žadatele tokenu.

Na závěr lze říct, že nově vytvořený protokol a jeho bezpečnostní token má potenciál pro další rozšiřování a je vhodný pro budoucí vývoj.

SEZNAM POUŽITÉ LITERATURY

[1] JANÁČ, Richard. *BEZPEČNOSTNÉ TOKENY* [online]. Brno, 2014 [cit. 2020-03-15]. Dostupné z: https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=83893. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií. Vedoucí práce Ing. Lešek FRANEK.

[2] *Kryptografie* [online]. [cit. 2020-03-15]. Dostupné z: https://cs.wikipedia.org/wiki/Kryptografie

[3] *Úvod do kryptologie* [online]. [cit. 2020-03-16]. Dostupné z: https://is.mendelu.cz/eknihovna/opory/zobraz_cast.pl?cast=7021

[4] KLIMEŠ, Cyril. *Kryptografie* [online]. [cit. 2020-03-16]. Dostupné z: https://prfczv.osu.cz/nabidka/seminar/data/Kryptografie.pdf

[5] *Šifrování* [online]. [cit. 2020-03-20]. Dostupné z: https://kore.fi.muni.cz/wiki/index.php/%C5%A0ifrov%C3%A1n%C3%AD

[6] *Symetrické a asymetrické šifrování* [online]. [cit. 2020-03-20]. Dostupné z: https://www.napocitaci.cz/33/symetricke-a-asymetricke-sifrovani-uniqueidgOkE4NvrWuNY54vrLeM677jX7sp3Lu-ZpLpGVMy1prA/

[7] *Symetricke-sifrovani* [online]. [cit. 2020-03-20]. Dostupné z: http://www.512.cz/index.php?title=Soubor:Symetricke-sifrovani.jpg

[8] *Data Encryption Standard* [online]. [cit. 2020-03-20]. Dostupné z: https://en.wikipedia.org/wiki/Data_Encryption_Standard

[9] *Frequently Asked Questions (FAQ) About the Electronic Frontier Foundation's "DES Cracker" Machine* [online]. [cit. 2020-03-20]. Dostupné z: https://web.archive.org/web/20170507231657/https://w2.eff.org/Privacy/Crypto/Crypto_misc/DESCracker/HTML/19980716_eff_des_faq.html

[10] *DATA ENCRYPTION STANDARD* [online]. [cit. 2020-03-20]. Dostupné z: https://csrc.nist.gov/csrc/media/publications/fips/46/3/archive/1999-10-25/documents/fips46-3.pdf

[11] *DES Key Schedule* [online]. [cit. 2020-03-23]. Dostupné z: https://upload.wikimedia.org/wikipedia/commons/0/06/DES-key-schedule.png

[12] *3DES is Officially Being Retired* [online]. [cit. 2020-03-23]. Dostupné z: https://www.cryptomathic.com/news-events/blog/3des-is-officially-being-retired

[13] *Trojnásobná aplikace šifry Data Encryption Standard (DES)* [online]. [cit. 2020-03-23]. Dostupné z: https://cs.wikipedia.org/wiki/Triple_DES#/media/Soubor:3des-overall-view.png

[14] *The Advanced Encryption Standard (AES) Algorithm* [online]. [cit. 2020-03-23]. Dostupné z: https://www.commonlounge.com/discussion/e32fdd267aaa4240a4464723bc74d0a5

[15] *Asymetrické šifrování* [online]. [cit. 2020-03-20]. Dostupné z: https://wikisofia.cz/wiki/Asymetrick%C3%A9_%C5%A1ifrov%C3%A1n%C3%AD

[16] *Asymetricke-sifrovani* [online]. [cit. 2020-03-20]. Dostupné z: http://www.512.cz/index.php?title=Soubor:Asymetricke-sifrovani.jpg

[17] *RSA* [online]. [cit. 2020-04-12]. Dostupné z: http://www.kryptografie.wz.cz/data/RSA.htm

[18] VLČEK, Martin. *ElGamal, Diffie-Hellman – Asymetrické šifrování* [online]. [cit. 2020-04-12]. Dostupné z: https://kmlinux.fjfi.cvut.cz/~balkolub/Vyuka/Vlcek.pdf

[19] *ElGamal* [online]. [cit. 2020-04-12]. Dostupné z: https://cs.wikipedia.org/wiki/ElGamal

[20] *Autentizace* [online]. [cit. 2020-04-20]. Dostupné z: https://cs.wikipedia.org/wiki/Autentizace

[21] *Autentizace a Autorizace* [online]. [cit. 2020-04-20]. Dostupné z: https://wikisofia.cz/wiki/Autentizace_a_Autorizace

[22] *Od anonymity k autentizácii* [online]. [cit. 2020-04-20]. Dostupné z: http://prevent-tista.sk/info/od-anonymity-k-autentizacii/

[23] *Autentizace a identifikace uživatelů* [online]. [cit. 2020-04-20]. Dostupné z: http://webserver.ics.muni.cz/bulletin/clanky_tisk/560.pdf

[24] How to animate an *UML Sequence Diagram?* [online]. [cit. 2020-04-20]. Dostupné z: https://www.visual-paradigm.com/tutorials/sequence-diagram-animation.jsp

[25] *Brute Force Attacks* [online]. [cit. 2020-04-23]. Dostupné z: https://www.thewindowsclub.com/wp-content/uploads/2015/02/Brute-Force-Attacks.jpg

[26] *Bezpečnost dat v informačních systémech* [online]. [cit. 2020-04-23]. Dostupné z: http://ikaros.cz/bezpecnost-dat-v-informacnich-systemech

[27] *Čipová karta* [online]. [cit. 2020-04-25]. Dostupné z: https://cardhouse.cz/sites/default/files/styles/produkt-full-560px/public/produkty/smart-karta_sle5542_0.jpg?itok=aNM8p4BM

[28] *Vstupní kódová klávesnice se čtečkou prstů a RFID* [online]. [cit. 2020-04-25]. Dostupné z: https://www.dstechnik.cz/vstupni-kodova-klavesnice-ctecka-rfid-125khz-ctecka-otisku-prstu-az-pro-500-uzivatelu/dst-ds-ac-f6-8177.html

[29] *Choosing an SSO Strategy: SAML vs OAuth2* [online]. [cit. 2020-05-19]. Dostupné z: https://www.mutuallyhuman.com/blog/choosing-an-sso-strategy-saml-vs-oauth2/

[30] *What is SAML and How Does it Work?* [online]. [cit. 2020-05-19]. Dostupné z: https://www.varonis.com/blog/what-is-saml

[31] *SAML vs. OAuth: Which One Should I Use?* [online]. [cit. 2020-05-19]. Dostupné z: https://dzone.com/articles/saml-versus-oauth-which-one

[32] VELÍŠEK, Ondřej. *Nasazení OAuth 2.0 pro systém Perun* [online]. Brno, 2016 [cit. 2020-05-19]. Dostupné z: https://is.muni.cz/th/wl7pd/thesis05.pdf. Bakalářská práce. Masarykova univerzita, Fakulta informatiky. Vedoucí práce RNDr. Michal PROCHÁZKA, Ph.D.

[33] *Assertions* [online]. [cit. 2020-06-03]. Dostupné z http://saml.xml.org/assertions

[34] *SAML2 vs JWT: A Comparison* [online]. [cit. 2020-06-03]. Dostupné z https://medium.com/@robert.broeckelmann/saml2-vs-jwt-a-comparison-254bafd98e6

[35] *Authorization Code Flow* [online]. [cit. 2020-05-19]. Dostupné z: https://assets.digitalocean.com/articles/oauth/auth_code_flow.png

[36] *Implicit Flow* [online]. [cit. 2020-05-19]. Dostupné z: https://assets.digitalocean.com/articles/oauth/implicit_flow.png

[37] *OAuth 2 Simplified* [online]. [cit. 2020-06-03]. Dostupné z: https://aaronparecki.com/oauth-2-simplified

[38] *An Introduction to OAuth 2* [online]. [cit. 2020-06-03]. Dostupné z: https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2

[39] *A Guide To OAuth 2.0 Grants* [online]. [cit. 2020-06-03]. Dostupné z: https://alexbilbie.com/guide-to-oauth-2-grants/

[40] *What is OAuth? Definition and How it Works* [online]. [cit. 2020-06-15]. Dostupné z: https://www.varonis.com/blog/what-is-oauth/

[41] *Why is there an "Authorization Code" flow in OAuth2 when "Implicit" flow works so well?* [online]. [cit. 2020-06-15]. Dostupné z: https://stackoverflow.com/questions/13387698/why-is-there-an-authorization-code-flow-in-oauth2-when-implicit-flow-works-s

[42] *OAuth 2.0 Grants* [online]. [cit. 2020-06-03]. Dostupné z: https://alexbilbie.com/images/oauth-grants.svg

[43] *Access Tokens* [online]. [cit. 2020-06-15]. Dostupné z: https://auth0.com/docs/tokens/access-tokens

[44] *JWT Flow* [online]. [cit. 2020-06-15]. Dostupné z: https://cdn-images-1.medium.com/max/800/1*SSXUQJ1dWjiUrDoKaaiGLA.png

[45] *Secure APIs using OAuth2 Opaque(Reference) Access Tokens [*online]. [cit. 2020-06-15]. Dostupné z: https://apim.docs.wso2.com/en/latest/learn/api-security/oauth2/access-token-types/opaque-tokens /

[46] *Understanding the different token formats* [online]. [cit. 2020-06-23]. Dostupné z: https://openiddict.github.io/openiddict-documentation/guide/token-formats.html

[47] *JWT Fundamentals for Beginners* [online]. [cit. 2020-06-23]. Dostupné z: https://morioh.com/p/79f6f8b073f8

[48] *Pros and Cons of JWTs* [online]. [cit. 2020-06-23]. Dostupné z: https://fusionauth.io/learn/expert-advice/tokens/pros-and-cons-of-jwts

[49] *JWT (Json Web Token) In Django REST API* [online]. [cit. 2020-06-23]. Dostupné z: https://medium.com/@ajitdhulam/jwt-json-web-token-in-django-rest-api-3056df2a4dfa

[50] *JWT* [online]. [cit. 2020-06-25]. Dostupné z: https://jwt.io/

[51] *JWT vs OAuth* [online]. [cit. 2020-06-25]. Dostupné z: https://community.apigee.com/questions/21139/jwt-vs-oauth.html

[52] *The Phantom Token Approach* [online]. [cit. 2020-06-25]. Dostupné z: https://curity.io/resources/architect/api-security/phantom-token-pattern/

[53] *Opaque token flow* [online]. [cit. 2020-06-25]. Dostupné z: https://curity.io/resources/architect/api-security/phantom-token-pattern

[54] *Web pages and web apps* [online]. [cit. 2020-07-15]. Dostupné z: https://www.bbc.co.uk/bitesize/guides/znkqn39/revision/1

[55] *Informace o webových aplikacích* [online]. [cit. 2020-07-15]. Dostupné z: https://helpx.adobe.com/cz/dreamweaver/using/web-applications.html#about_web_applications

[56] *General Python FAQ* [online]. [cit. 2020-07-17]. Dostupné z: https://docs.python.org/3/faq/general.html

[57] *Python (programming language)* [online]. [cit. 2020-07-17]. Dostupné z: https://en.wikipedia.org/wiki/Python_(programming_language)

[58] *Lekce 1 - Úvod do frameworku Flask a webových aplikací v Pythonu* [online]. [cit. 2020-07-17]. Dostupné z: https://www.itnetwork.cz/python/flask/uvod-do-frameworku-flask-a-webovych-aplikaci-v-pythonu

[59] *Foreword* [online]. [cit. 2020-07-17]. Dostupné z: https://flask.palletsprojects.com/en/1.1.x/foreword/#what-does-micro-mean

[60] *Key Features of SQLAlchemy* [online]. [cit. 2020-07-20]. Dostupné z: https://www.sqlalchemy.org/features.html

[61] HORKÝ, Michal. Informační systém florbalového klubu [online]. Brno, 2018 [cit. 2020-07-20]. Dostupné z: https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=181632. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních techonologií. Vedoucí práce Ing. Vladimír BARTÍK, Ph.D.

[62] *Alembic* [online]. [cit. 2020-07-20]. Dostupné z: https://alembic.sqlalchemy.org/en/latest/

[63] *Flask-Migrate* [online]. [cit. 2020-07-20]. Dostupné z: https://flask-migrate.readthedocs.io/en/latest/#why-use-flask-migrate-vs-alembic-directly

[64] *Visual Studio Code FAQ* [online]. [cit. 2020-07-20]. Dostupné z: https://code.visualstudio.com/docs/supporting/faq

[65] *Modular Applications with Blueprints* [online]. [cit. 2020-07-20]. Dostupné z: https://flask.palletsprojects.com/en/1.1.x/blueprints/#the-concept-of-blueprints

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

AES	Advanced Encryption Standard.
API	Application Programming Interface
DB	Databáze
DES	Data Encryption Standard.
DP	Diplomová práce
HTTP	Hypertext Transfer Protocol
IB	Internetové bankovnictví.
JSON	Javascript Object Notation.
JWT	JSON Web Token.
ORM	Object Relational Mapper
OTP	One-time passwords
PID	Poskytovatel identity.
PDF	Portable Document Format
PS	Poskytovatel služeb
RSA	Rivest-Shamir-Adleman.
SaaS	Software as a Service
SAML	Security Assertion Markup Language.
SQL	Structured Query Language
SSO	Single-Sign on
SW	Software
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
WA	Webová aplikace
XML	Extensible Markup Language

SEZNAM OBRÁZKŮ

SEZNAM PŘÍLOH

PŘÍLOHA P I: OBSAH CD

Struktura obsahu přiloženého CD:

- Adresář **Diplomová práce** – obsahuje text DP ve formátu PDF
- Adresář **Webová aplikace** – obsahuje všechny zdrojové kódy a soubory nutné pro fungování flaskové webové aplikace

PŘÍLOHA P II: VÝSLEDKY SPUŠTĚNÍ SOUBORU CLIENT.PY