Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

# Porovnanie API tokenov

Bakalárska práca

2023
Jitka Muravská

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

# Porovnanie API tokenov

Bakalárska práca

| | |
|---|---|
| Študijný program: | Informatika |
| Študijný odbor: | Informatika |
| Školiace pracovisko: | Katedra informatiky |
| Školiteľ: | RNDr. Richard Ostertág, PhD. |

Bratislava, 2023
Jitka Muravská

20174181

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

# ZADANIE ZÁVEREČNEJ PRÁCE

**Meno a priezvisko študenta:** Jitka Muravská
**Študijný program:** informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
**Študijný odbor:** informatika
**Typ záverečnej práce:** bakalárska
**Jazyk záverečnej práce:** slovenský
**Sekundárny jazyk:** anglický

**Názov:** Porovnanie API tokenov
*Comparison of API Tokens*

**Anotácia:** Aplikačné rozhrania (API) často majú aj neverejnú časť. V takom prípade treba pri ich implementácii riešiť aj kontrolu prístupu k chránenej časti API. Väčšina schém zabezpečenia API používa token, ktorý je súčasťou jednotlivých požiadaviek. Tieto tokeny sú nejakým spôsobom spojené s identitou a autorizáciou používateľa. Aplikačné rozhranie prevezme požiadavku, extrahuje token, a podľa pravidiel prístupu rozhodne ako pokračovať.

Cieľom práce je porovnanie rôznych API tokenov (napríklad: JWT, PASETO, Authenticated Requests, Macaroons, Biscuits, ...). Prvým krokom bude zozbieranie a popísanie v praxi používaných API tokenov. Následne sa vykonajú porovnania ich výhod a nevýhod (napríklad rýchlosť, jednoduchosť použitia) na jednoduchej základnej aplikácii.

**Vedúci:** RNDr. Richard Ostertág, PhD.
**Katedra:** FMFI.KI - Katedra informatiky
**Vedúci katedry:** prof. RNDr. Martin Škoviera, PhD.

**Spôsob sprístupnenia elektronickej verzie práce:**
bez obmedzenia

**Dátum zadania:** 12.10.2022

**Dátum schválenia:** 13.10.2022

doc. RNDr. Dana Pardubská, CSc.
garant študijného programu

.....
študent

.....
vedúci práce

**Poďakovanie: Chcem poďakovať všetkým, ktorí mi pomohli pri písaní tejto práce, ale aj počas celého bakalárskeho štúdia. Ďakujem môjmu školiteľovi doktorovi R. Ostertágovi za odborné vedenie, cenné rady a pripomienky. Ďakujem aj mojej rodine a kamarátom za podporu pri celom štúdiu. V neposlednom rade ďakujem môjmu priateľovi Emuškovi za neutíchajúcu trpezlivosť a pomoc s hľadaním motivácie.**

# Abstrakt

Veľa aplikácií má neverejné časti, prípadne poskytujú rôzne služby rôznym používateľom na základe ich prístupových práv. V takom prípade je nutné, aby mala aplikácia zavedenú schému zabezpečenia ošetrujúcu autentifikačné a autorizačné procesy. V prípade aplikačných rozhraní (API) je tento problém častý a preto existuje jeho viacero riešení. Táto práca popisuje základné schémy zabezpečenia a detailne sa venuje schéme zabezpečenia využívajúcej API tokeny. Konkrétne sa táto práca zaoberá porovnaním rôznych API tokenov na základe viacerých kritérií ako bezpečnosť, flexibilita, popularita a rýchlosť. Porovnanie je vykonané na základe štúdia špecifikácií jednotlivých API tokenov a experimentov na jednoduchom API, ktorého implementácia je v práci popísaná.

**Kľúčové slová: Aplikačné rozhranie (API), API token, autorizácia**

# Abstract

Many applications have non public parts, or provide different services to different users based on certain access rights. In this case, it is necessary for the application to have an established security scheme that takes care of authentication and authorization processes. In the case of application programming interfaces (APIs), this problem is common and therefore there are several solutions for it. This thesis describes basic security schemes and details a security scheme using API tokens. Specifically, this work deals with the comparison of different API tokens based on several criteria such as security, flexibility, popularity and speed. The comparison is made on the basis of a study of the specifications of individual API tokens and experiments on a simple API, the implementation of which is described in the work.

**Keywords: Application programming interface (API), API token, authorization**

# Obsah

**Úvod** **1**

**1 Schémy zabezpečenia API a využitie tokenov** **3**

1.1 Zabezpečenie bez autentifikácie . . . . . 3
1.2 Zabezpečenie bez schémy . . . . . 3
1.3 Schéma zabezpečenia s využitím identifikátora sedenia . . . . . 4
1.4 Schéma zabezpečenia využívajúca API kľuče . . . . . 5
1.5 Schéma zabezpečenia využívajúca API tokeny . . . . . 6
1.6 Typy tokenov . . . . . 7
1.6.1 Prístupový token . . . . . 7
1.6.2 Nositeľský token . . . . . 8
1.6.3 Obnovovací token . . . . . 8
1.6.4 Identifikačný token . . . . . 8
1.7 Formáty tokenov . . . . . 9
1.7.1 Nepriehľadný token . . . . . 9
1.7.2 Štruktúrovaný token . . . . . 9
1.7.3 Fantómový token . . . . . 10
1.7.4 Rozdelený token . . . . . 10

**2 Špecifikácia konkrétnych API tokenov** **13**

2.1 JSON Web Token . . . . . 13
2.1.1 Štruktúra JWT . . . . . 13
2.1.2 Generovanie a validácia JWT . . . . . 14
2.2 Platform Agnostic Security Token . . . . . 15
2.2.1 Verzie PASETO . . . . . 15
2.2.2 Štruktúra PASETO . . . . . 15
2.2.3 Generovanie a validácia PASETO . . . . . 16
2.3 Fernet . . . . . 16
2.3.1 Štruktúra Fernet . . . . . 17
2.3.2 Generovanie a validácia Fernet . . . . . 18
2.4 Branca . . . . . 18

2.4.1 Štruktúra Branca . . . . . 18
2.4.2 Generovanie a validácia Branca . . . . . 19
2.5 Macaroons . . . . . 20
2.5.1 Štruktúra Macaroons . . . . . 20
2.5.2 Generovanie a delegácia Macaroons . . . . . 21
2.5.3 Vytvorenie požiadavky s Macaroons tokenom . . . . . 22
2.5.4 Spracovanie požiadavky cieľovou službou . . . . . 24
2.6 Biscuits . . . . . 24
2.6.1 Štruktúra Biscuits . . . . . 25
2.6.2 Generovanie a delegácia autorizácie . . . . . 25
2.6.3 Validácia Biscuits . . . . . 26
2.6.4 Delegácia časti autorizácie na tretiu stranu . . . . . 27

**3 Teoretické porovnanie API tokenov 29**
3.1 Bezpečnosť . . . . . 30
3.1.1 Kryptografické primitíva . . . . . 30
3.1.2 Útok pomýlením algoritmu . . . . . 32
3.1.3 Útok opakovaním a problém odvolania . . . . . 34
3.2 Flexibilita . . . . . 36
3.2.1 Bezstavovosť autorizačnej služby . . . . . 36
3.2.2 Delegácia autorizácie . . . . . 36
3.2.3 Autorizačná schéma v tokene . . . . . 37
3.2.4 Štandardná validácia . . . . . 38
3.3 Popularita a využiteľnosť . . . . . 38

**4 Testovacie API 41**
4.1 Použité technológie . . . . . 41
4.2 Popis rozhrania . . . . . 42
4.3 Obsah a generovanie tokenu . . . . . 42
4.4 Práca s knižnicami . . . . . 43
4.5 Meranie rýchlosti . . . . . 44
4.6 Analýza nameraných výsledkov . . . . . 45

**Záver 53**

**Príloha A 61**

# Zoznam obrázkov


1.1 Schéma s použitím identifikátora sedenia . . . . . 5
1.2 Schéma s použitím API kľúča . . . . . 6
1.3 Schéma s použitím tokenu . . . . . 7

2.1 JWT token . . . . . 14
2.2 PASETO token . . . . . 17
2.3 Fernet token . . . . . 18
2.4 Branca token . . . . . 19
2.5 Macaroons token . . . . . 21
2.6 Pridanie pravidla tretej strany . . . . . 23
2.7 Získanie vybíjacieho tokenu používateľom . . . . . 24
2.8 Biscuits token . . . . . 26

3.1 Útok pomýlením algortimu . . . . . 33

4.1 Krabicový graf – generovanie, všetky hodnoty . . . . . 45
4.2 Krabicový graf – validácia, všetky hodnoty . . . . . 46
4.3 Krabicový graf – generovanie, hodnoty bez nepriehľadného tokenu . . . 46
4.4 Krabicový graf – validácia, hodnoty bez nepriehľadného tokenu . . . . 47
4.5 Čiarový graf – generovanie, hodnoty bez nepriehľadného tokenu . . . . 47
4.6 Čiarový graf – validácia, hodnoty bez nepriehľadného tokenu . . . . . 48
4.7 Histogram – generovanie, všetky hodnoty . . . . . 49
4.8 Histogram – validácia, všetky hodnoty . . . . . 49
4.9 Čiarový graf – generovanie, odpočítaný NONE token . . . . . 50
4.10 Čiarový graf – validácia, odpočítaný NONE token . . . . . 50
4.11 Čiarový graf – generovanie, konštantný priemer . . . . . 50
4.12 Čiarový graf – validácia, konštantný priemer . . . . . 51

# Zoznam tabuliek

3.1 Teoretické porovnanie tokenov . . . . . 29
3.2 Bezpečnosť tokenov . . . . . 30
3.3 Flexibilita tokenov . . . . . 36
3.4 Vybrané implementácie tokenov . . . . . 39
3.5 Popularita tokenov . . . . . 40

4.1 Kryptografické funkcie na podpisovanie a šifrovanie tokenov . . . . . 43
4.2 Jednoduchosť práce s knižnicami . . . . . 44
4.3 Porovnanie tokenov . . . . . 54

# Úvod

Pri aplikačných rozhraniach (API), ktoré aspoň z časti nie sú verejné, je potrebné navrhnúť spôsob autentifikácie a autorizácie. Tento spôsob a jeho riadenie sa nazýva schéma zabezpečenia API. Úlohou API tokenu v schéme zabezpečenia je prenášať informácie nutné pre autorizáciu požiadavky klienta na API. Tieto informácie môžu byť napríklad identifikátor používateľa, jeho prístupové práva, čas platnosti tokenu a podobne. API token väčšinou získa používateľ úspešnou autentifikáciou voči API.

Cieľom tejto práce je porovnanie konkrétnych API tokenov. Naša práca môže slúžiť aj ako základ pri vytváraní schémy zabezpečenia v rámci vývoja API, konkrétne pri výbere adekvátneho API tokenu ku štruktúre a funkcionalite API. V práci sa zameriame na vybrané API tokeny, konkrétne: Nepriehľadný token, JWT, PASETO, Fernet, Braca, Macaroons a Biscuits.

API tokeny sú úzko spojené s problematikou autentifikácie a autorizácie vo všeobecnosti. Preto sa zo začiatku práce venujeme aj tejto problematike, kde predstavujeme základné schémy zabezpečenia softvérových systémov.

Pre komplexné porovnanie vybraných API tokenov je nutné pochopenie ich fungovania. Teda pochopenie štruktúry API tokenu, spôsobu jeho vytvorenia a overenia. Preto v práci detailne popisujeme špecifikácie jednotlivých API tokenov.

API tokeny nejakým spôsobom autorizujú požiadavky na API. Taktiež môžu prenášať aj citlivé informácie o používateľoch. Preto je zaujímavé sa venovať ich bezpečnosti. Moderné systémy často pozostávajú z veľa komponentov, ktoré medzi sebou komunikujú. Okrem iného môžu medzi sebou komunikovať aj pre účely autorizácie požiadaviek. Preto ak používajú v svojej architektúre API tokeny, je dôležité aby boli schémy týchto API tokenov schopné pracovať nad viacerými službami. Každý používaný softvér sa vyvíja, preto je pre dobré fungovanie akéhokoľvek systému dôležité, aby bol aktívne udržiavaný a schopný reagovať na vývoj ostatných služieb. Aby sa autorovi softvéru oplatilo ho udržiavať, musí byť tento softvér používaný a teda populárny.

Všetky tieto pozorovania sa dajú zhrnúť do troch vlastností – bezpečnosť, flexibilita a popularita. Tieto vlastnosti tvoria dôležité aspekty, ktoré je potrebné zohľadniť pri výbere API tokenu. Preto sa im v práci pri porovnaní API tokenov venujeme a porovnávame ich medzi rôznymi API tokenmi.

Významným parametrom API je jeho výkonnosť. Teda koľko požiadaviek vie spra-

covať za určitý časový interval. Generovanie a validácia API tokenu sa spája aj s výpočtom rôznych kryptografických funkcií, ktoré sú často náročné na výpočet. Preto je pri porovnaní API tokenu zaujímavé zohľadniť aj rýchlosť jeho generovania a validácie. Z tohto dôvodu v práci navrhneme a implementujeme jednoduché API, na ktorom porovnáme rýchlosť generovania a validácie vybraných API tokenov.

# Kapitola 1

# Schémy zabezpečenia API a využitie tokenov

V tejto kapitole uvedieme viaceré známe prístupy k riešeniu autentifikácie a autorizácie. Stručne objasníme ako fungujú a aké sú ich výhody a nevýhody. V našej práci sa detailnejšie pozrieme na prístup využívajúci API tokeny a na samotné tokeny a ich rozdelenie.

## 1.1 Zabezpečenie bez autentifikácie

V prípade, že je aplikačné rozhranie plne verejné, nepotrebuje žiadnu schému zabezpečenia. Ľubovoľný používateľ môže volať rozhranie bez predchádzajúcej autentifikácie a neexistuje spôsob ako obmedziť prístup k volaniam rozhrania.

Jediným identifikátorom autora požiadavky je jeho IP adresa. To je však veľmi slabý identifikátor a nedáva nám veľké možnosti v obmedzení prístupu k rozhraniu.

Hlavnou výhodou tohto riešenia je jednoduchosť implementácie, rozhranie nepotrebuje uchovávať žiadne dáta o používateľoch a všetky požiadavky sú jednoduché, ich rýchlosť závisí len od rýchlosti samotného volania a rýchlosti siete.

Nevýhodou je samozrejme strata kontroly nad prístupom k rozhraniu, ktorá sa obmedzila buď na nejakú kontrolu na základe IP adresy alebo úplne vymizla. Môžeme regulovať počet požiadaviek na rozhranie v časovom intervale alebo rýchlosť komunikácie s rozhraním, či už pre celé rozhranie alebo na základe IP adresy.

## 1.2 Zabezpečenie bez schémy

Najjednoduchšie riešenie autentifikácie a autorizácie je posielanie prihlasovacích údajov v každej požiadavke na rozhranie. Klient jednoducho pripojí prihlasovacie údaje ku každej požiadavke a rozhranie si ich overí vo svojej databáze a v prípade úspechu vráti

požadované dáta.

Toto riešenie nie je vhodné ak sa niekde v rámci komunikácie nachádza nezabezpečené spojenie. Útočník, ktorý by takúto komunikáciu zachytil, by mal jednoduchý prístup k prihlasovacím údajom používateľa.

Používanie nezabezpečeného spojenia je všeobecne nebezpečné bez ohľadu na schému zabezpečenia a typ prenášaného údaju používaného na autorizáciu, preto ďalej v práci budeme predpokladať, že spojenia s každým koncovým bodom sú zabezpečené pomocou SSL/TLS.

Aj v prípade zabezpečeného spojenia však existujú zraniteľnosti [50]. Prihlasovacie údaje sú zriedkavo menený identifikátor a teda po ich odchytení sa dajú dlho zneužívať. Okrem samotného úniku citlivých údajov a z toho vyplývajúcich nepríjemností pre používateľa má tento prístup aj ďalšie nevýhody.

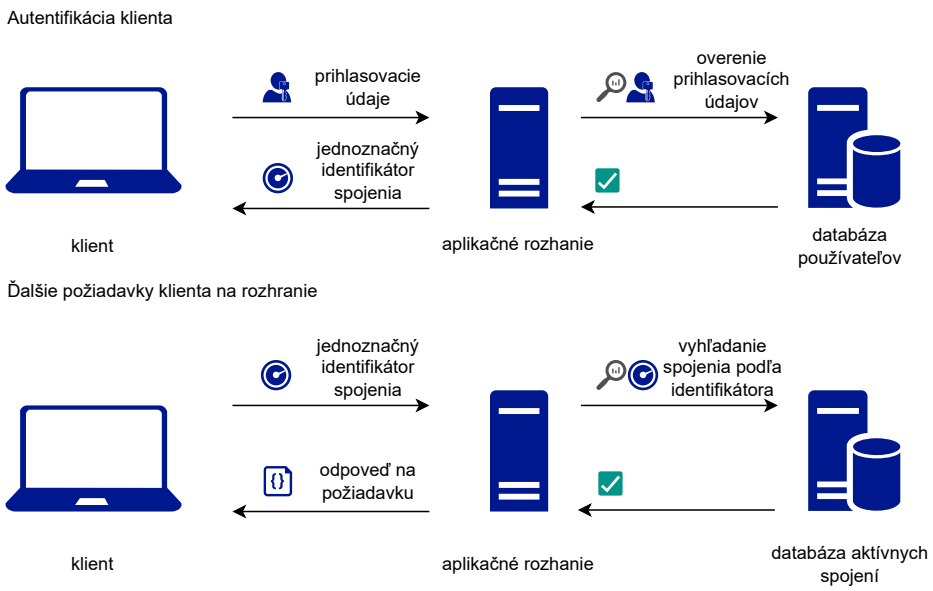
Okrem priameho zneužitia získaných hesiel útočníkom, môže útočník ukladať heslá do databázy odchytených hesiel, ktorá môže byť využitá aj inými útočníkmi pri ďalších útokoch. Tento prístup sa nazýva *slovníkový útok* a môže byť veľmi efektívny. [44]. Ľudia tiež často používajú podobné heslá na všetky svoje účty. Potom sa podľa istých vzorov odhalených v sade hesiel jedného používateľa dajú ľahšie uhádnuť ďalšie heslá. [25].

## 1.3 Schéma zabezpečenia s využitím identifikátora sedenia

Prvé riešenie, kde môžeme hovoriť o nejakej schéme zabezpečenia, nie len o existencii prihlasovacích údajov a ich overení pri každej požiadavke, je sledovanie každého aktívneho sedenia s rozhraním (angl. session).

Pri prvej požiadavke sa používateľ autentifikuje voči serveru, ten si vytvorí a uloží záznam o sedení. Tento záznam môže obsahovať rôzne základné informácie o sedení (napríklad čas vytvorenia) a informácie o používateľovi, s ktorým je sedenie vytvorené. Dôležitá časť záznamu je jednoznačný identifikátor sedenia v podobe náhodného reťazca. Potom klientovi vráti identifikátor sedenia. Všetky nasledujúce požiadavky klienta budú obsahovať tento identifikátor, podľa ktorého vie server určiť, ktorému používateľovi patria, teda či je autentifikovaný, prípadne či má právo vykonať dané volanie (ak rozhranie implementuje nejakú autorizačnú schému). Popísaná schéma je znázornená na obrázku 1.1.

Oproti predošlému riešeniu pribudla potreba spravovať sedenia, no veľkou výhodou je, že sa už neposielajú citlivé prihlasovacie údaje v každej požiadavke. Rozhranie však stále potrebuje pri každej požiadavke preložiť identifikátor na záznam o používateľovi.




Obr. 1.1: Autentifikačná schéma s použitím identifikátora sedenia.

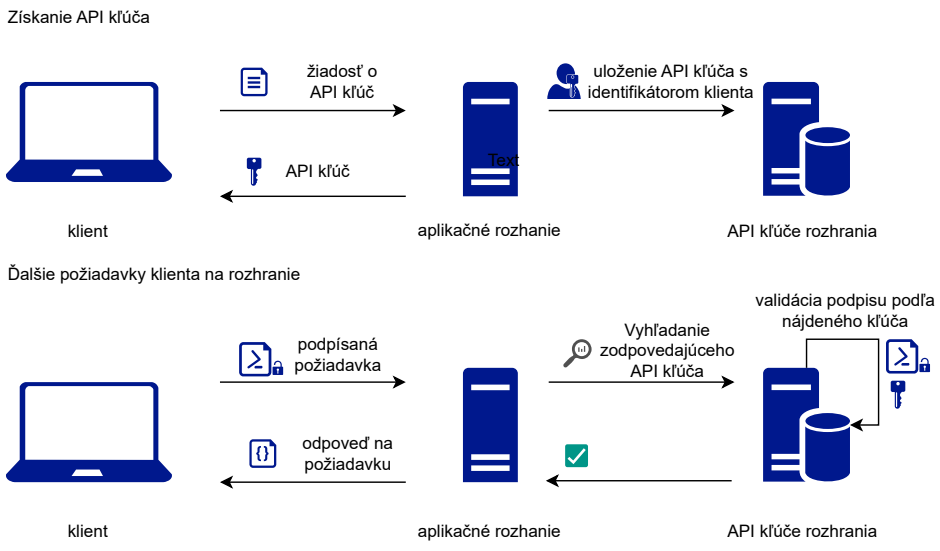
## 1.4 Schéma zabezpečenia využívajúca API kľuče

API kľúč je tajný kľúč slúžiaci na podpisovanie celej požiadavky na rozhranie. Podpísaním požiadavky vznikne autentifikovaná požiadavka (angl. authenticated request). Na podpisovanie sa väčšinou používa hešovanie s kľúčom a ako podpis potom slúži hešovaný autentifikačný kód. Tento kód potom klient posiela spolu s požiadavkou. API kľúče využíva napríklad populárna cloudová služba AWS [4].

Na vytvorenie požiadavky teda klient potrebuje najprv získať API kľúč. Ten klient získa tak, že si ho vyžiada od rozhrania, ktorého služby alebo zdroje chce využívať. Rozhranie vygeneruje API kľúč pre klienta (po prípadnom overení jeho identity, či zaplatení poplatku za využívanie služieb rozhrania), uloží si ho a vráti ho klientovi. Klient tento kľúč následne používa pri vytváraní požiadaviek na rozhranie tým, že ich podpíše. Rozhranie zvaliduje podpis pomocou kľúča, ktorý si uložilo a ak je platný vykoná požiadavku.

HTTP požiadavka nemá jednotnú formu a teda požiadavka reprezentujúca to isté volanie môže mať viac reprezentácií, ktoré sa líšia napríklad poradím parametrov alebo využitím rôznych hlavičiek pre uloženie rovnakej informácie. Podobne serializácia požiadavky do textovej podoby môže byť rôzna. Preto je potrebné definovať kanonickú schému, ako sa má požiadavka serializovať. Inak by rozhranie nedokázalo serializovať požiadavku s rovnakým výstupom ako klient, podpísať ju a tento podpis porovnať s podpisom priloženým v požiadavke. Schéma na obrázku 1.2 zobrazuje realizáciu schémy zabezpečenia pomocou API kľúčov.

Nevýhodou použitia API kľúčov je teda nutnosť udržiavania stavu v rozhraní. Rozhranie si musí pamätať kľúče, ktoré vydalo a ku každému kľúču nejaký identifikátor,




Obr. 1.2: Autentifikačná schéma s použitím API kľúča.

s ktorým klientom je spojený, prípadne aj jeho oprávnenia, napríklad maximálny počet volaní za deň.

Samotný API kľúč sa teda neposiela spolu s požiadavkou, ale iba raz po jeho vygenerovaní rozhraním. Tým pádom jeho odchytenie nie je jednoduché a preto máva API kľúč neobmedzenú platnosť. Avšak požiadavka, ktorá je už podpísaná pomocou API kľúča sa posiela po sieti a tým pádom sa dá odchytiť. Takáto požiadavka už je autentifikovaná. Preto po jej odchytení ju môže útočník použiť viac krát a tým používať nejaký cenný zdroj rozhrania alebo preťažiť rozhranie veľkým množstvom požiadaviek, ktoré rozhranie vykoná, lebo sú autentifikované.

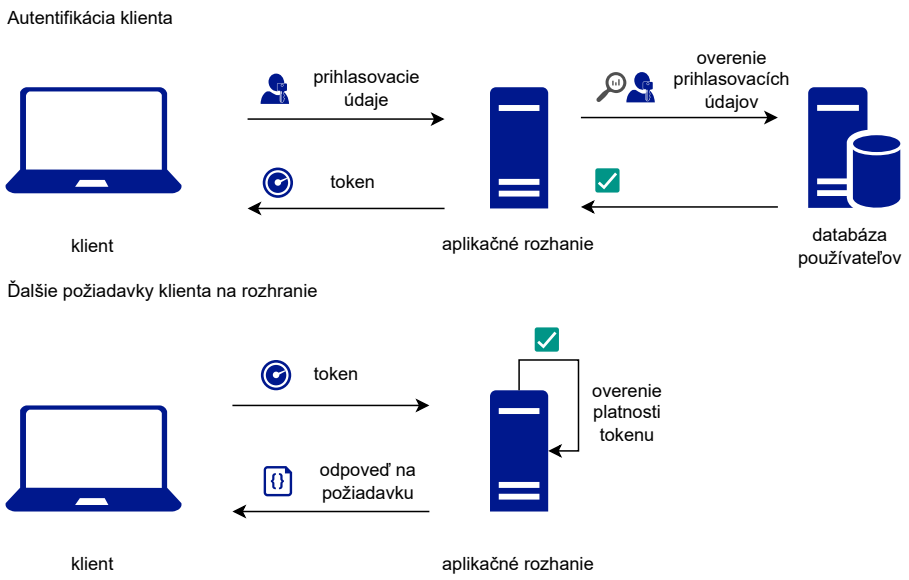
Ako obrana voči tejto zraniteľnosti sa využíva časová platnosť autentifikovanej požiadavky. Klient vloží do požiadavky, pri jej vytváraní, časovú pečiatku dokedy je platná, tento interval môže byť veľmi krátky, napríklad 5 minút.

Ďalšia zraniteľnosť API kľúčov často nastáva pri ich uložení na nesprávne miesto. Napríklad pri webovej aplikácii nie je bezpečné ukladať kľúč na klientskej strane, ale na serveri a odtiaľ vykonávať volania na rozhranie. Rovnako pri mobilnej aplikácii nie je bezpečné ukladať kľúč priamo v aplikácii, lebo sa dá získať reverzným inžinierstvom [18].

## 1.5 Schéma zabezpečenia využívajúca API tokeny

API token (ďalej token) je identifikátor, ktorý slúži na autorizáciu prípadne aj identifikáciu používateľa pri prístupe k rozhraniu. Token môže mať viaceré podoby, jednotlivým typom a formátom sa venujú podkapitoly 1.6 a 1.7.

Tok v rámci schémy zabezpečenia rozhrania funguje podobne ako pri využití iden-




Obr. 1.3: Autentifikačná schéma s použitím tokenu.

tifikátora sedenia, ktorý sme popísali v podkapitole 1.3. Líši sa najmä v tom, že token môže niesť pridanú informáciu a rozhranie vie jeho platnosť overiť bezstavovo. Popísaná schéma s využitím tokenu je zobrazená na obrázku 1.3. Na dosiahnutie tejto výhody je nutné použiť štruktúrovaný formát tokenu. Možné formáty tokenov detailnejšie popíšeme v podkapitole 1.7.

## 1.6 Typy tokenov

Tokeny využívané v schéme zabezpečenia rozhrania sa dajú rozdeliť podľa ich generického využitia na niekoľko typov. Nie v každom scenári máme explicitne dané aký token využiť, no niektoré sú často vhodnejšie ako iné. V tejto kapitole predstavíme viaceré typy tokenov a to prístupový (angl. access), nositeľský (angl. bearer), obnovovací (angl. refresh) a identifikačný token.

### 1.6.1 Prístupový token

Prístupový token [45] je najzákladnejší a najčastejšie používaný typ tokenu. Tento token vygeneruje autentifikačný server po autentifikácii používateľa a ďalej slúži na autorizáciu v rozsahu, ktorý sa určil v rámci autentifikácie konkrétneho používateľa. Prístupový token môže mať rôzne formáty podľa toho ako funguje schéma zabezpečenia rozhrania.

Token je spojený s istými údajmi o používateľovi, ako napríklad identifikátor a prístupové práva. Tieto údaje môžu byť uložené priamo v tokene alebo na serveri, v závislosti od toho aký formát tokenu je použitý. Zároveň je s tokenom spojený aj časový

limit platnosti, ktorý býva kvôli bezpečnosti krátky. Po odchytení tokenu totiž môže útočník vystupovať v mene používateľa, ktorého údaje sú uložené v tokene. Po uplynutí časového limitu platnosti tokenu sa používateľ musí opätovne autentifikovať voči autentifikačnému serveru a ten mu vydá nový token.

### 1.6.2 Nositeľský token

Nositeľský token reprezentuje najčastejší spôsob použitia prístupového tokenu. Pri jeho použití má nositeľ tokenu prístup k rozhraniu bez ohľadu nato, kto je. Stačí aby v požiadavke poslal platný nositeľský token.

Následne rozhranie nekontroluje identitu používateľa, ale iba platnosť tokenu. Preto pri jeho využití treba prikladať väčší dôraz na bezpečnosť tokenu napríklad tak, že jeho platnosť bude veľmi krátka, napríklad 5 minút.

### 1.6.3 Obnovovací token

Ako sme už spomínali vyššie, prístupový token má často relatívne krátky časový limit platnosti. Ak chce používateľ využívať rozhranie aj po jeho uplynutí, je potrebné aby získal nový prístupový token.

Na riešenie tohto problému existujú dva jednoduché spôsoby. Buď sa používateľ znova autentifikuje a server mu vygeneruje nový prístupový token alebo si klient, cez ktorého používateľ komunikuje s rozhraním bude pamätať prihlasovacie údaje používateľa a využije ich na opätovnú autentifikáciu používateľa. Oba prístupy nie sú ideálne, prvá možnosť je nekomfortná pre používateľa a nie je možná, ak je klientom nejaká služba alebo iná aplikácia. Druhá možnosť je nebezpečná, lebo klient musí mať niekde uložené prihlasovacie údaje používateľa a tým sa zvyšuje riziko ich získania útočníkom.

Obnovovací token adresuje problém s krátkou platnosťou prístupových tokenov. Pri prvotnej autentifikácii používateľa server okrem prístupového tokenu vygeneruje aj obnovovací token. Oba tokeny vráti klientovi. Následne keď uplynie platnosť prístupového tokenu, klient pošle požiadavku o nový prístupový token pomocou obnovovacieho tokenu.

Schému s obnovovacími tokenmi využíva napríklad populárny autentifikačný protokol OAuth 2.0 [26].

### 1.6.4 Identifikačný token

Identifikačný token je špeciálny typ prístupového tokenu. V tomto prípade musí ísť o štruktúrovaný formát tokenu. Token obsahuje identifikátor používateľa a autorizačné údaje. Navyše môže token obsahovať doplnkové údaje ako vydavateľa tokenu, čas platnosti tokenu, čas vydania tokenu a podobne.

Identifikačný token využíva napríklad protokol OpenID Connect (OIDC), čo je nadstavba protokolu OAuth 2.0 o identitu používateľa [54]. OIDC dokonca presne špecifikuje použitie JWT (JSON Web Token), ktorý viac predstavíme v kapitole 2.

## 1.7 Formáty tokenov

Už sme spomenuli, že existujú rôzne typy tokenov, no iba pri identifikačnom tokene je striktne daný formát tokenu. Pri ostatných typoch sa stretávame s rôznymi formátmi tokenu. Vo všeobecnosti rozoznávame dva formáty tokenov a to nepriehľadné (angl. opaque) a štruktúrované tokeny. Existujú aj hybridné formáty, ktoré kombinujú výhody obidvoch spomínaných typov, no pri nich sa dá hovoriť skôr o istých vzoroch ako o formátoch. My predstavíme dva hybridné formáty a to konkrétne fantómové a rozdelené (angl. split) tokeny.

Tokeny často zaručujú ochranu autenticity a integrity. Vybrané tokeny, ktoré porovnávame v tejto práci, používajú rôzne kryptografické algoritmy na zabezpečenie integrity a autenticity tokenu. Využíva sa buď digitálne podpisovanie (založené na asymetrickom šifrovaní) alebo hešovanie s kľúčom [37]. Výsledkom týchto algoritmov je buď digitálny podpis alebo hešovaný autentifikačný kód (HMAC). V celej práci budeme v oboch prípadoch hovoriť o podpise tokenu. A proces ich vytvárania budeme nazývať podpisovanie. V prípade, že budeme chcieť hovoriť iba o podpise, ktorý vznikol pomocou asymetrického šifrovania, budeme explicitne hovoriť o digitálnom podpise.

### 1.7.1 Nepriehľadný token

Ide o najjednoduchší formát tokenu. Nepriehľadný token je náhodný reťazec znakov. Jednoduchý je v tom, že nenesie žiadnu pridanú informáciu. Všetky metadáta o tokene si musí pamätať rozhranie.

Takýto prístup so sebou nesie významné výhody aj nevýhody. Výhodou je, že samotné vytvorenie tokenu je veľmi rýchle, rozhranie jednoducho vygeneruje náhodný reťazec. Vďaka tomu, že nenesie žiadne pridané informácie, nenesie ani citlivé informácie o používateľovi. Navyše je jeho platnosť obmedzená (narozdiel od platnosti prihlasovacích údajov), teda jeho zachytenie útočníkom nie je také nebezpečné. Nevýhodou však je, že rozhranie ho nevie bezstavovo overiť, teda ho musí napríklad vyhľadať v databáze platných tokenov a to je časovo náročné.

### 1.7.2 Štruktúrovaný token

Na rozdiel od nepriehľadného tokenu štruktúrovaný token obsahuje pridanú informáciu napríklad identifikáciu používateľa, jeho prístupové práva, čas platnosti tokenu, čas

vydania tokenu a podobne.

Aby sa predišlo úniku citlivých informácií z tokenu, je token šifrovaný. Využívajú sa rôzne symetrické aj asymetrické algoritmy. Všetky konkrétne protokoly, ktoré rozoberáme v tejto práci, používajú štruktúrované tokeny a v kapitole 2 sa venujeme ich vytváraniu a s ním spojenému šifrovaniu alebo hešovaniu tokenov.

Najväčšou výhodou štruktúrovaného tokenu je, že rozhranie ho môže bezstavovo overiť, lebo pozná kľúč, ktorým bol token zašifrovaný, teda ho vie dešifrovať a získať informácie, ktoré nesie. Zo získaných informácií vie rozhranie overiť platnosť tokenu a autorizovať používateľa.

### 1.7.3 Fantómový token

Ako sme naznačili v úvode podkapitoly, fantómový token [11] je hybridný formát tokenu. Podľa tohto vzoru autentifikačný server vygeneruje dva tokeny, nepriehľadný token pre klienta a štruktúrovaný token pre rozhranie. Ďalej sa využíva API brána alebo reverzný proxy server (RPS), v ktorom sa uloží dvojica tokenov ako kľúč a hodnota. Kľúčom je nepriehľadný token, hodnotou je štruktúrovaný token.

Následne, keď klient pošle požiadavku na rozhranie, tak pridá nepriehľadný token. API brána alebo RPS ho preloží na štruktúrovaný token a tento štruktúrovaný token poskytne rozhraniu. Rozhranie ho môže bezstavovo overiť a využiť pridané informácie, ktoré nesie.

Brána alebo RPS síce musí vyhľadať záznam s dvojicou tokenov, kde je kľúčom poslaný nepriehľadný token, ale môže si výsledok uložiť do medzipamäte (napríklad služba Redis [53]) pre ďalšie požiadavky. Tým sa zvýši priepustnosť brány alebo RPS a zároveň ubudnú nároky na výkonnosť rozhrania. Moderné systémy majú často architektúru mikroslužieb, kde jeden autentifikačný server môže vydávať tokeny autorizujúce požiadavky na viacero rozhraní. V takomto systéme môže byť jedna brána alebo jeden RPS zdieľaný medzi viacerými rozhraniami.

Výhodou oproti obyčajnému štruktúrovanému tokenu je, že klient nedrží v pamäti žiadne citlivé informácie.

### 1.7.4 Rozdelený token

Rozdelený token [12] má podobnú schému a výhodu ako fantómový token, no navyše obmedzuje štruktúrovaný token vydaný autentifikačným serverom tým, že musí obsahovať podpis chrániaci jeho autenticitu.

V schéme rozdeleného tokenu vydá autentifikačný server len štruktúrovaný token. Podpis z tohto tokenu pošle klientovi a do medzipamäte brány alebo RPS zapíše celý token s hešovaným podpisom ako kľúčom.

Požiadavky od klienta budú obsahovať ako nepriehľadný token získaný podpis a brána alebo RPS ho zahešuje a preloží na štruktúrovaný token pomocou medzipamäte. Token následne spolu s požiadavkou pošle rozhraniu.

# Kapitola 2

# Špecifikácia konkrétnych API tokenov

V tejto kapitole predstavíme v praxi využívané API tokeny, ktorými sa zaoberá naša práca. Uvedieme ich formát a základné postupy pre ich vytvorenie, či validáciu. Ich využitie, výhody či nevýhody rozoberieme v kapitole 3.

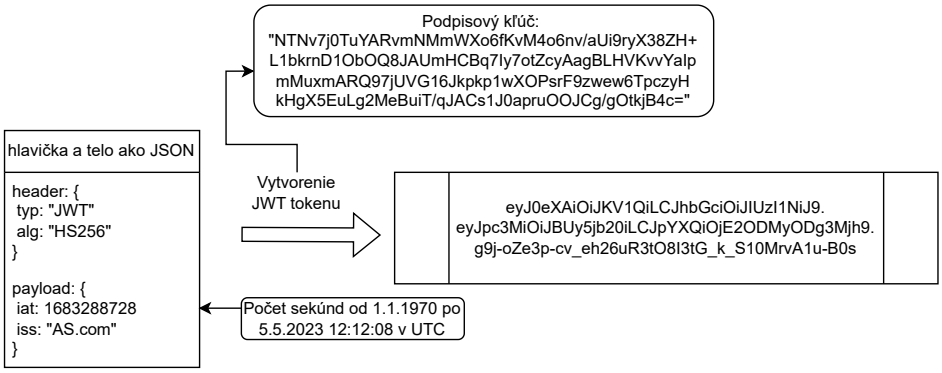
## 2.1 JSON Web Token

Prvý token, ktorým sa budeme zaoberať je JSON web token (JWT) [32]. JWT vznikol ako súčasť JOSE (JSON object signing and encryption) štandardov [6], čo je dokument vypracovaný pracovnou skupinou IETF (Internet Engineering Task Force) na základe aplikácií bezpečnostných mechanizmov v rámci vývoja softvéru.

Definujú štandard pre bezpečný prenos JSON objektov medzi službami, ktoré sú schopné ich overiť a dešifrovať. Zavádzajú tri základné formáty JSON objektov a to JWS (JSON Web Signature), JWE (JSON Web Encryption) a JWK (JSON Web Key), ktorým sa detailnejšie venujú ďalšie štandardy [31, 33, 30]. Prvé dva sú formáty zabezpečujúce bezpečnostné vlastnosti JSON objektov. Oba zabezpečujú autentickosť a integritu pomocou digitálnych podpisov alebo hešovaného autentifikačného kódu. JWE navyše zabezpečuje aj dôvernosť a to šifrovaním obsahu JSON objektu. Posledný formát JWK je formát pre reprezentáciu kľúčov, ktoré sú použité v kryptografických algoritmoch využitých v JWS a JWE. Kryptografické algoritmy a ich identifikátory sú definované JSON Web Algorithms (JWA) štandardom [29].

### 2.1.1 Štruktúra JWT

Samotný JWT je v podstate iba serializovaný JSON objekt chránený JWS alebo JWE. Podľa štandardu JWT obsahuje tri samostatné časti oddelené bodkami - hlavičku, telo a podpis. Hlavička a telo sú serializované JSON objekty obsahujúce oprávnenia (angl. claim) vo forme dvojíc kľúč, hodnota. Niektoré oprávnenia (konkrétne ich kľúče) sú definované v štandarde a teda by sa nemali používať pre žiadne iné hodnoty.




Obr. 2.1: Príklad jednoduchého JWT tokenu.

A to konkrétne v hlavičke sú najdôležitejšie *typ* a *alg*. Prvý určuje typ tokenu a druhý algoritmus, ktorý bol použitý na vytvorenie podpisu alebo v rámci šifrovania obsahu tokenu. Rôzne možnosti pre algoritmy sú definované v JWA štandardoch.

Telo tvorí logický obsah tokenu, napríklad môže obsahovať oprávnenia týkajúce sa konkrétnej autentifikácie a používateľa, pre ktorého bol token vydaný. Dôležité, štandardom popísané, kľúče sú napríklad *iss*, *sub*, *exp*, *nbf*, *iat*. Popisujú postupne vydavateľa tokenu, identifikátor používateľa, čas vypršania platnosti tokenu, čas, kedy sa token začne považovať za platný a čas vydania tokenu.

Do tela aj hlavičky sa môžu vkladať ľubovoľné iné oprávnenia, napríklad *admin*, *role*, *permissions*, určujúce oprávnenia používateľa.

### 2.1.2 Generovanie a validácia JWT

Hlavička a telo sa serializujú pomocou base64url kódovania [34]. V prípade JWE sa telo ešte pred serializáciou šifruje. Následne sa obe časti podpíšu algoritmom definovaným v hlavičke a podpis sa zreťazí s hlavičkou a telom. Výsledný reťazec sa používa ako token.

Na overenie platnosti tokenu treba overiť podpis. Podpis je vytvorený pomocou algoritmu definovanom v hlavičke. Teda pri validácii tokenu sa ako prvé dekóduje hlavička tokenu a z nej sa prečíta hodnota v kľúči *alg*.

Na základe hodnoty v kľúči *alg* sa určí algoritmus, ktorý bol použitý na podpis tokenu. Následne sa podpis zvaliduje.

Ak bol token vytvorený pomocou JWE, na prečítanie tela je potrebné ho najprv dešifrovať pomocou algoritmu, ktorý je zapísaný ako oprávnenie s kľúčom *enc* v hlavičke tokenu. V prípade využitia JWS je telo po dekódovaní priamo čitateľné. Následne môžeme overiť informácie o časovej platnosti tokenu, či právach používateľa a pod.

Príklad vytvorenia nešifrovaného JWT tokenu s podpisovou funckiou HMAC-SHA256 a telom obsahujúcim oprávnenie *iss* a *iat* je zorbazený na obrázku 2.1.

## 2.2 Platform Agnostic Security Token

Platform Agnostic Security Token (PASETO) je relatívne nový štandard tokenu navrhnutý v roku 2018 a je stále v štádiu RFC draftu [3]. Je inšpirovaný rodinou štandardov JOSE (JWT, JWS, JWE, JWK). Jednoducho povedané snaží sa zjednodušiť implementáciu a použitie kryptografických funkcií.

Rovnako ako JWT, PASETO serializuje JSON objekty a zaručuje rôzne bezpečnostné kvality pri ich prenose cez internet. Pôvodne bol PASETO navrhnutý s dvoma verziami *v1* a *v2* líšiacimi sa v použitých kryptografických algoritmoch. Dnes už existujú štyri verzie *v1*, *v2*, *v3* a *v4* popísané štandardom [58]. Každá verzia tokenu zaručuje autentickosť a integritu obsahu tokenu a to pomocou digitálneho podpisu alebo pomocou hešovaného autentifikačného kódu.

### 2.2.1 Verzie PASETO

Ako sme spomenuli PASETO má štyri verzie. Každá verzia sa delí na dve ďalšie podľa jej využitia na lokálnu a verejnú. Lokálne tokeny majú zašifrované telo a tým zabezpečujú dôvernosť dát uložených v tele tokenu. Na rozdiel od toho sú verejné tokeny nešifrované a dáta v ich tele sú čitateľné pre kohokoľvek s prístupom k danému tokenu.

Každá verzia PASETO používa iný algoritmus na podpisovanie a prípadne šifrovanie tokenu v prípade lokálnych tokenov. Jednotlivé algoritmy pre konkrétne verzie a ich použitie sú popísané v špecifikácii [58].

Novšie verzie *v3* a *v4* prinášajú modernejšie kryptografické algoritmy a pridávajú niektoré funkcionality. Napríklad verzia *v3* prináša nepopierateľnosť autorstva ako novú bezpečnostnú kvalitu. Dosahuje to dokonca bez predĺženia podpisu a to pomocou pridania verejného kľúča do tokenu pred vypočítaním podpisu [49]. Tiež zavádza podporu pre implicitné informácie, teda také informácie, ktoré nie sú uložené priamo v tokene, ale používajú sa pri výpočte podpisu. Teda sú to informácie potrebné pre validáciu tokenu, ale z nejakého dôvodu nie je vhodné ich vkladať priamo do tokenu. Napríklad môže ísť o citlivé interné dáta. Podrobná motivácia za zavedením nových verzií je popísaná v špecifikácii [58].

### 2.2.2 Štruktúra PASETO

PASETO sa skladá z troch alebo štyroch častí zreťazených bodkou. Časti postupne reprezentujú verziu, využitie, telo a pätu. Prvé tri časti sú povinné a päta je nepovinná, ale dovoľuje nám zapísať akékoľvek ďalšie informácie do tokenu.

- Verzia – reprezentuje verziu PASETO. Môže byť *v1*, *v2*, *v3* alebo *v4*.

- Využitie – určuje využitie tokenu ako lokálne alebo verejné. Možné hodnoty sú *local* alebo *public*.

- Telo – reprezentuje samotné dáta uložené v tokene. Podobne ako pri JWT ide o oprávnenia vo forme dvojíc kľúč hodnota a rovnako sú niektoré dôležité kľúče definované špecifikáciou. [58]

- Päta – môže obsahovať ľubovoľné ďalšie informácie.

Telo a päta sú vo forme JSON objektu, ktorý je serializovaný pomocou base64url [34].

### 2.2.3 Generovanie a validácia PASETO

Pri vytváraní tokenu sa musíme najprv rozhodnúť pre verziu a využitie podľa toho aké bezpečnostné požiadavky máme na token. Následne vytvoríme telo tokenu obsahujúce informácie, ktoré chceme pomocou tokenu prenášať, napríklad informácie o vzniku tokenu, jeho platnosti, jeho autorovi, či určenom subjekte. Ďalej môžeme pridať ďalšie informácie do päty tokenu ako napríklad identifikátor kľúča kryptografickej funkcie.

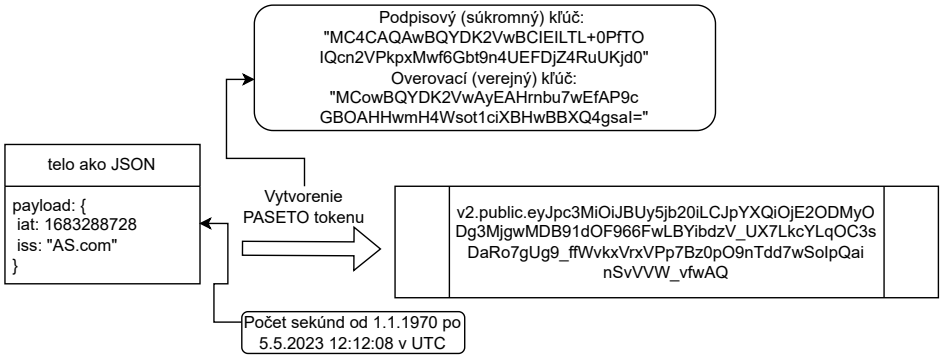
Ak sme zvolili lokálne využitie, tak telo tokenu zašifrujeme. Následne vypočítame podpis z tela a päty tokenu. Šifrovanie aj podpisovanie sa dosahuje pomocou kryptografickej funkcie vybratej podľa verzie a využitia. Nakoniec všetky časti spojíme do jedného reťazca a oddelíme bodkami.

Validácia tokenu je komplementárny proces ku generovaniu. Najprv rozdelíme reťazec na časti a zistíme verziu a využitie tokenu a podľa toho vyberieme použitú kryptografickú funkciu. Následne zistíme, či je podpis tokenu platný pomocou adekvátnej kryptografickej funkcie a kľúča. Ak mal token lokálne využitie dešifrujeme jeho telo a skontrolujeme časovú platnosť tokenu, ak to využívaná schéma podporuje a telo obsahuje informácie o platnosti tokenu.

Príklad vytvorenia PASETO tokenu verzie *v2* s verejným využitím a telom obsahujúcim oprávnenie *iss* a *iat* je zobrazený na obrázku 2.2.

## 2.3 Fernet

Pôvodne Fernet vznikol ako nástroj na zasielanie bezpečných správ v platforme cloudových služieb Heroku [21]. V súčasnosti už podľa špecifikácie [22] vzniklo veľa implementácií Fernetu v rôznych programovacích jazykoch [17, 42], v rámci Heroku bol implementovaný v Ruby. Fernet bol dokonca vybratý PYCA (Python Cryptographic Authority) [52] ako štandard pre implementáciu symetrického šifrovania v Pythone.




Obr. 2.2: Príklad jednoduchého PASETO tokenu.

Fernet je štruktúrovaný token, lebo v sebe nesie rôzne informácie, no nijak nešpecifikuje formát týchto informácií. Väčšina implementácií s nimi pracuje ako s obyčajným reťazcom znakov.

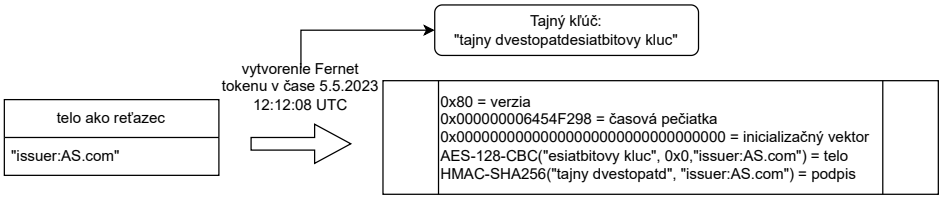
Token je navrhnutý s možnosťou pridania viacerých verzií, no v súčasnosti existuje len jediná verzia. V tejto verzii je obsah tokenu zašifrovaný pomocou AES-128-CBC [19] a celý token je podpísaný pomocou HMAC-SHA256 [20]. Z toho vyplýva, že Fernet zaručuje autentickosť, integritu a dôvernosť.

### 2.3.1 Štruktúra Fernet

Fernet sa skladá z piatich zreťazených častí. Každá časť reprezentuje jednu informáciu o tokene. Jednotlivé časti sú nasledovné:

- Verzia – reprezentuje verziu Fernet tokenu, aktuálne existuje len jedna verzia a je reprezentovaná číslom 0x80. Zaberá vždy 8 bitov.
- Časová pečiatka – reprezentuje čas vytvorenia tokenu. Čas je zachytený ako počet sekúnd od 1.1.1970 v UTC časovej zóne. Zaberá vždy 64 bitov.
- Inicializačný vektor (IV) – reprezentuje náhodný reťazec znakov, ktorý je použitý pri šifrovaní a dešifrovaní tokenu. IV musí byť unikátny a najmä nepredvídateľný pre každý token, preto sa generuje náhodnou funkciou. Zaberá vždy 128 bitov.
- Zašifrované telo – reprezentuje zašifrované dáta uložené v tokene. Môže mať premenlivú dĺžku, no vždy násobok 128 bitov.
- Podpis – reprezentuje výstup HMAC-SHA256 funkcie a zaberá vždy 256 bitov.

Ako vidíme, všetky časti tokenu okrem tela majú pevne danú dĺžku. Vďaka tejto vlastnosti nemusia byť zreťazené časti oddelené žiadnym špeciálnym symbolom, napríklad bodkou, lebo vieme jednoznačne oddeliť verziu, časovú pečiatku, IV aj podpis a tým pádom aj zašifrované telo.




Obr. 2.3: Príklad jednoduchého Fernet tokenu.

Pre jednoduché prenášanie je celý token zakódovaný pomocou base64url [34].

Príklad Fernet tokenu pred base64url zakódovaním s telom obsahujúcim reťazec *issuer:AS.com* je schematicky vyjadrený na obrázku 2.3.

### 2.3.2 Generovanie a validácia Fernet

Pri generovaní Fernet tokenu sa využívajú dva kryptografické algoritmy, ktoré vyžadujú kľúč. Fernet definuje 256 bitový kľúč, ktorý je rozdelený na dve 128 bitové časti. Prvá časť reprezentuje podpisový kľúč a druhá šifrovací kľúč.

Existuje iba jedna verzia tokenu s pevne daným algoritmom. Pre vygenerovanie tokenu, potrebujeme zaznamenať aktuálny čas do časovej pečiatky a vygenerovať náhodný reťazec, ktorý bude slúžiť ako IV. Následne zašifrujeme telo tokenu pomocou šifrovacieho kľúča a IV. Ďalej vypočítame podpis z predchádzajúcich častí tokenu pomocou podpisového kľúča. Nakoniec všetky časti spojíme do jedného reťazca a zakódujeme pomocou base64url [34].

Validácia tokenu spočíva v dekódovaní tokenu, rozdelení na časti a overení podpisu. Potom dešifrujeme telo tokenu pomocou šifrovacieho kľúča a IV. Následne prípadne overíme časovú platnosť tokenu, ak telo obsahuje potrebné informácie pre overenie časovej platnosti ako vznik a doba platnosti tokenu.

## 2.4 Branca

Motiváciou k vzniku Branca tokenu [61] bolo modernizovanie použitých kryptografických konštrukcií Fernet tokenu. Branca má podobnú štruktúru aj generovanie a validáciu ako Fernet. Branca sa líši najmä v tom, že využíva šifrovaciu a podpisovú funkciu XChaCha20-Poly1305 AEAD [2].

### 2.4.1 Štruktúra Branca

Branca sa podobne ako Fernet skladá z piatich zreťazených častí. Tieto časti sa však jemne líšia od častí Fernet tokenu a zakódované sú *base62* kódovaním [28].

- Verzia – reprezentuje verziu Branca tokenu, aktuálne existuje len jedna verzia a je reprezentovaná číslom 0xBA. Zaberá vždy 8 bitov.

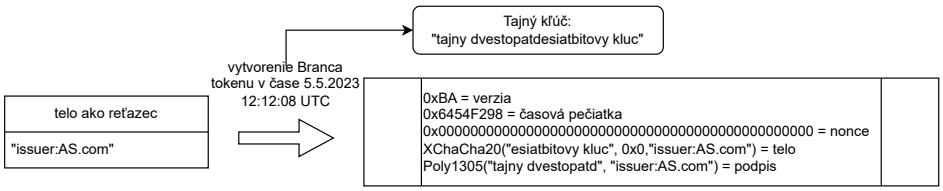
- Časová pečiatka – reprezentuje čas vytvorenia tokenu. Čas je zachytení ako počet sekúnd od 1.1.1970 v UTC časovej zóne. Zaberá vždy 32 bitov a je zapísaná ako číslo bez znamienka.

- Príležitostné slovo (angl. nonce) – reprezentuje náhodný reťazec znakov, ktorý využíva šifrovacia funkcia. Ide v podstate o IV, ale využíva sa naozaj len raz, zatiaľ čo IV sa v blokovej šifre využije viac krát. Zaberá vždy 192 bitov.

- Zašifrované telo – reprezentuje zašifrované dáta uložené v tokene. Môže mať ľubovoľnú dĺžku.

- Podpis – v podobe hešovaného autentifikačného kódu - reprezentuje výstup funkcie Poly1305 a zaberá vždy 128 bitov.

Príklad Branca tokenu pred base64url zakódovaním s telom obsahujúcim reťazec *issuer:AS.com* je schematicky vyjadrený na obrázku 2.4.




Obr. 2.4: Príklad jednoduchého Branca tokenu.

### 2.4.2 Generovanie a validácia Branca

Vygenerujeme 192 bitový reťazec znakov, ktorý bude slúžiť ako príležitostné slovo a zachytíme aktuálny čas do časovej pečiatky. Vyrobíme hlavičku zreťazením verzie, časovej pečiatky a príležitostného slova. Následne zašifrujeme telo tokenu pomocou šifrovacej funkcie, do ktorej vložíme tajný kľúč, príležitostné slovo a ako dodatočnú informáciu použijeme hlavičku. Funkcia vráti zašifrované telo a hešovaný autentifikačný kód vypočítaný aj z hlavičky. Nakoniec všetky časti spojíme do jedného reťazca a zakódujeme pomocou *base62* kódovania [28].

Pri validácii tokenu ho ako prvé dekódujeme. Následne overíme, že prvých 8 bitov je 0xBA a token rozdelíme na jednotlivé časti. Dešifrujeme telo a overíme podpis pomocou funkcie XChaCha20-Poly1305 AEAD. Následne môžeme overiť napríklad časovú platnosť tokenu pomocou dodatočných informácií v tele tokenu a časovej pečiatky.

## 2.5 Macaroons

Macaroons sú tokeny s kontextovými pravidlami. Vznikli v rámci výskumného projektu Belay v spoločnosti Google [23]. Google predstavil Macaroons v práci z roku 2014 [7]. Macaroons sú autorizačné poverenia (angl. credentials) pre cloudové služby s podporou decentralizovanej delegácie medzi službami v rámci cloudu. Ľubovoľná entita vlastniaca token autorizujúci určitý prístup môže tento token *zoslabiť* alebo aj *kontextovo obmedziť* a posunúť ďalšej entite. Zoslabenie aj kontextové obmedzenie sa realizuje pomocou pravidiel. Entitu generujúcu nový Macaroons token budeme označovať ako *cieľová služba*.

Pravidlá sa delia podľa strany, ktorá potvrdí alebo zabezpečí ich naplnenie na pravidlá prvej a tretej strany. Pravidlá prvej strany sú jednoduché predikáty. Na autorizáciu požiadavky sprevádzanej Macaroons tokenom sa musia všetky tieto predikáty vyhodnotiť pravdivo v rámci kontextu danej požiadavky. Kontextom požiadavky môže byť napríklad aktuálny čas alebo IP adresa odosielateľa. Pravidlom prvej strany môže byť napríklad obmedzenie typu požiadaviek iba na čítacie. Pravidlá tretích strán vyžadujú dôkaz o nejakej skutočnosti od tretej strany. Pri dôkaze od tretích strán sa využíva princíp dôkazu držiteľa kľúča, kde tretia strana dokáže, že pozná nejaký tajný kľúč napríklad tak, že podpíše zadaný reťazec znakov, ktorý vie cieľová služba overiť. Pravidlom tretej strany môže byť požiadavka na doloženie dôkazu od nejakej autentifikačnej služby (napríklad autentifikačnej služby Google), že používateľ je autentifikovaný. Tento dôkaz musí získať používateľ pred odoslaním požiadavky na rozhranie a pri jej odoslaní ho priloží k požiadavke spolu s originálnym tokenom. Dôkaz má formu špeciálneho tokenu, ktorý predstavíme v podkapitole 2.5.3. Pravidlá tretích strán sa používajú na delegáciu autorizácie medzi službami.

Macaroons zabezpečuje ochranu integrity a autenticity pomocou hešovaného autentifikačného kódu. Pôvodná práca [7] nevyžaduje použitie konkrétnej hešovacej funkcie, no napríklad implementácia [15] využíva funkciu HMAC-SHA256 [20].

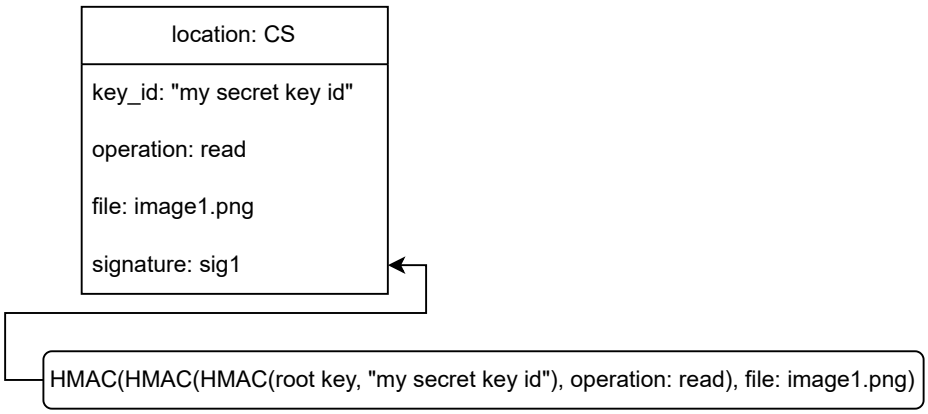
### 2.5.1 Štruktúra Macaroons

Macaroons token sa skladá z lokalizácie, identifikátora, pravidiel a podpisu.

- Lokalizácia – reprezentuje nápovedu na lokalitu cieľovej služby. Často reprezentovaná ako URL adresa.
- Identifikátor – slúži na odvodenie koreňového kľúča využitého pri tvorbe tokenu.
- Pravidlá – reprezentujú predikáty, ktoré musia byť splnené pre autorizáciu požiadavky.

- Podpis – reprezentuje postupne generovaný podpis identifikátora a pravidiel tokenu.

Príklad Macaroons tokenu je zobrazený v obrázku 2.5.




Obr. 2.5: Príklad jednoduchého Macaroons tokenu.

### 2.5.2 Generovanie a delegácia Macaroons

Každá cieľová služba disponuje tajným kľúčom, prípadne spôsobom ako ho vygenerovať. Ku každému tajnému kľúču musí vedieť odvodiť verejný nepriehľadný identifikátor, ktorý dokáže spätne previesť na tajný kľúč. Takého identifikátory môžu byť implementované napríklad ako náhodné príležitostné slová reprezentujúce kľúč v databáze alebo pomocou šifrovania s verejným alebo súkromným kľúčom [38].

Cieľová služba vytvorí nový token z danej lokácie, identifikátora a tajného kľúča $sk$ tak, že podpíše identifikátor pomocou $sk$ a spojí lokáciu, identifikátor a podpis. Podpis vypočíta ako *HMAC(sk, identifikátor)*.

Takto vytvorený token si môžeme predstaviť ako univerzálny kľúč, ktorý autorizuje ľubovoľnú požiadavku na cieľovú službu. Cieľová služba môže ďalej token poslať inej službe. Ako sme uviedli v úvode podkapitoly, každá služba môže token zoslabiť alebo kontextovo obmedziť pridaním pravidiel.

Pravidlá prvej strany pridá služba do tokenu $M$ tak, že pridá reťazec popisujúci pravidlo (tento reťazec označme ako $rule$) do tokenu a vypočíta nový podpis tokenu pomocou doterajšieho podpisu $M.signature$ ako $HMAC(M.signature, rule)$. Takto vytvoreným podpisom nahradí predchádzajúci podpis. Tento proces môže zopakovať viac krát a tým pridať ľubovoľný počet pravidiel.

Na pridanie pravidla tretej strany musí mať služba vzťah s danou službou tretej strany a dôverovať jej. Služba $A$ pridávajúca pravidlo vygeneruje koreňový kľúč pravidla $RK$ a potrebuje zabezpečiť, aby ho vedela zderivovať daná služba tretej strany $B$ ako aj cieľová služba $CS$.

Prvý prípad môže služba $A$ zabezpečiť viacerými spôsobmi napríklad tak, že pošle kľúč a pravidlo službe $B$ cez zabezpečený kanál a tá jej vráti jeho identifikátor $cId$. Alebo ak zverejňuje služba $B$ verejný kľúč, prípadne služby zdieľajú tajný kľúč, môže služba $A$ vytvoriť $cId$ zašifrovaním $RK$ a pravidla pomocou šifrovacieho kľúča $cRK$ (verejný kľúč služby $B$ alebo zdieľaný tajný kľúč). Druhý prípad zabezpečí služba $A$ symetrickým zašifrovaním $RK$ pomocou podpisu tokenu. Takto vzniknutý reťazec označme $vId$.

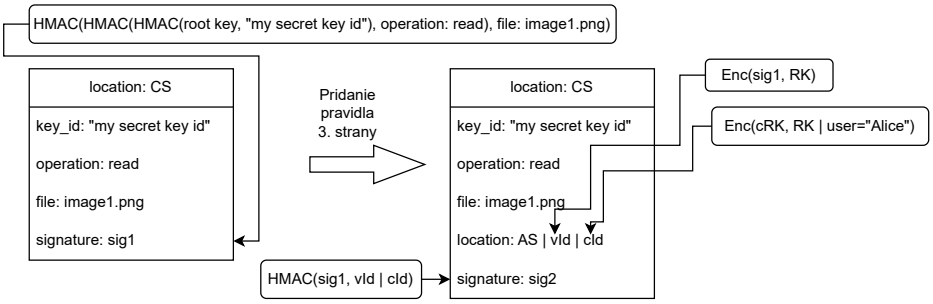
Reťazec reprezentujúci pravidlo tretej stany je zreťazenie lokácie služby tretej strany, $cId$ a $vId$, označme tento reťazec $rule_B$. Služba $A$ vloží $rule_B$ do zoznamu pravidiel tokenu $M$ a vytvorí nový podpis tokenu pomocou aktuálneho podpisu tokenu $M.signature$ ako $HMAC(M.signature, rule_B)$. Príklad pridania pravidla tretej strany je na obrázku 2.6. Služba tretej strany vie z $cId$ odvodiť koreňový kľúč pravidla, lebo $cId$ buď sama vytvorila alebo pozná kľúč na dešifrovanie $cId$. Cieľová služba pozná koreňový kľúč tokenu, teda si vie pomocou postupného podpisovania pravidiel odvodiť kľúč pre dešifrovanie $vId$ a dešifrovaním $vId$ získa koreňový kľúč pravidla.

Ako príklad využitia pravidla tretej strany si môžeme službu $A$ predstaviť ako službu spravujúcu nejaké fórum. Cieľová služba je v tomto prípade iná služba, ktorá autorizuje požiadavky na fórum. Fórum vlastní token $M$, oprávňujúci pridať príspevok na fórum. Používateľka Alica chce pridať príspevok na fórum. Fórum pridá do tokenu $M$ pravidlo tretej strany, vyžadujúce potvrdenie od autentifikačného serveru, že používateľ je naozaj Alica a pošle upravený $M$ Alici. Alica preskenuje $M$ a nájde pravidlo tretej strany s autentifikačným serverom ako lokáciou. Pošle toto pravidlo autentifikačnému serveru, ktorý po prečítaní pravidla vyzve Alicu aby sa autentifikovala prihlasovacím menom a heslom. Ak sa Alica úspešne autentifikuje, autentifikačný server vráti Alici dôkaz, že toto pravidlo je splnené a Alica teraz môže poslať požiadavku na fórum spolu s $M$ a priloženým dôkazom od autentifikačného serveru. Fórum prenechá validáciu $M$ a priloženého dôkazu na cieľovú službu. Ak je validácia úspešná, pridá služba spravujúca fórum príspevok od Alice na fórum. Presný tvar dôkazu od autentifikačného serveru popíšeme v nasledujúcej podkapitole.

Tvorba podpisu tokenu teda zodpovedá reťazovej aplikácii funkcie HMAC na identifikátor a pravidlá tokenu. Hešovanie je ireverzibilná operácia, preto nie je možné pravidlá odstraňovať, lebo nato by bolo potrebné vypočítať predchádzajúci podpis tokenu. To je možné iba podpísaním identifikátora koreňovým kľúčom a následným podpisovaním pravidiel vždy pomocou posledného podpisu ako kľúča.

### 2.5.3 Vytvorenie požiadavky s Macaroons tokenom

Služba komunikujúca s klientom pošle token klientovi. Na vytvorenie požiadavky autorizovanej týmto tokenom, musia byť splnené všetky pravidlá tokenu. Splnenie pravidiel




Obr. 2.6: Príklad pridania pravidla tretej strany, konkrétne pravidla *user="Alice"* na autentifikačný server (AS). $RK$ je koreňový kľúč pravidla a symbol | reprezentuje zreťazenie.

prvej strany závisí od samotného kontextu požiadavky, no pre splnenie pravidiel tretej strany musí klient poskytnúť cieľovej službe dôkazy o ich splnení od daných služieb tretích strán.

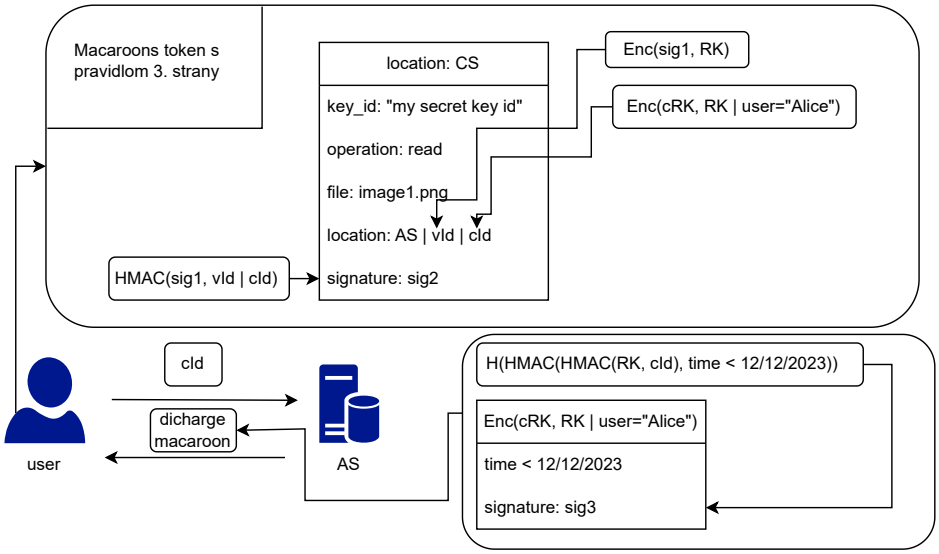
Tieto dôkazy sú reprezentované *vybíjacími tokenmi* (angl. discharge Macaroons). Vybíjacie tokeny majú rovnakú štruktúru aj postup generácie ako obyčajné Macaroons tokeny. Potom, čo klient obdrží Macaroons token, ho prehľadá pre všetky pravidlá tretích strán. Pre každé pravidlo tretej strany, pošle požiadavku na službu tretej strany s $cId$ daného pravidla. Služba tretej strany zderivuje koreňový kľúč pravidla a samotné pravidlo z $cId$. Následne môže vykonať akékoľvek opatrenia na overenie splnenia pravidla klientom, napríklad vyzvať klienta, aby zadal heslo. Ak je pravidlo splnené, vytvorí služba tretej strany vybíjací token z $cId$ pomocou koreňového kľúča pravidla. Následne môže pridať do vybíjacieho tokenu ľubovoľné ďalšie pravidlá vrátane pravidiel tretích strán. Potom podpis tokenu zahešuje (hešovaním bez kľúča), aby sa nedali do vybíjacieho tokenu pridávať ďalšie pravidlá. Nakoniec pošle vybíjací token klientovi. Príklad získania vybíjacieho tokenu je na obrázku 2.7.

Keď klient získa vybíjací token pre každé pravidlo tretej strany, vytvorí požiadavku na cieľovú službu, ku ktorej priloží Macaroons token a všetky vybíjacie tokeny.

V príklade z predchádzajúcej podkapitoly je práve vybíjací token dôkazom od autentifikačného serveru, ktorý server zašle Alici po úspešnej autentifikácii.

Vybíjacie tokeny sú silnými dôkazmi od tretích strán, dokazujú splnenie pravidla s rovnakým $cId$ a koreňovým kľúčom pravidla v každom Macaroons tokene. Ak by klient poslal požiadavku s vybíjacími tokenmi inej ako cieľovej službe (napríklad ako výsledok phishing útoku), táto služba môže jednoducho zneužiť vybíjacie tokeny klienta na dôkaz splnenia pravidiel vo vlastných tokenoch. Preto je potrebné aby klient vybíjacie tokeny zviazal s jeho Macaroons tokenom ešte predtým ako vytvorí požiadavku.

Zviazanie pôvodného Macaroons tokenu $M$ s vybíjacím tokenom $M'$ prebieha úpravou podpisu $M'$. Špecifikácia necháva otvorené, ako presne má táto úprava vyzerať.




Obr. 2.7: Príklad získania vybíjacieho tokenu používateľom. $RK$ je koreňový kľúč pravidla a symbol | reprezentuje zreťazenie.

Jedna z možných implementácií by bola vytvoriť nový podpis $M'$ zahešovaním podpisov $M$ a $M'$ pomocou hešovacej funkcie bez kľúča (H), napríklad SHA-256. Nový podpis $M'.signature$ by sa potom vypočítal ako $M'.signature = H(M.signature, M'.signature)$.

### 2.5.4 Spracovanie požiadavky cieľovou službou

Predtým ako cieľová služba autorizuje požiadavku od klienta zvaliduje priložený Macaroons token. Pre úspešnú validáciu tokenu musia byť všetky pravidlá splnené a podpis tokenu korektný.

Splnenie pravidiel prvej strany overí cieľová služba overením splnenia predikátu každého pravidla v rámci kontextu požiadavky. Pravidlá tretej strany overí služba rekurzívne nasledovným postupom. Pre každé pravidlo nájde vybíjací token a z $vId$ pravidla zderivuje koreňový kľúč pravidla. Rekurzívne overí všetky pravidlá vo vybíjacom tokene a pomocou koreňového kľúča pravidla overí korektnosť podpisu vybíjacieho tokenu. Ak sú všetky pravidlá splnené a podpisy všetkých tokenov korektné, autorizuje služba požiadavku klienta.

## 2.6 Biscuits

Ako posledný predstavíme najmladší token rozoberaný v tejto práci. Biscuits token bol predstavený v roku 2021 v blogu od spoluautora z firmy Clever Cloud [9]. Vo voľne dostupnom repozitári [8] nájdeme detailne popísanú motiváciu a vývoj tokenu. Pôvodne bol Biscuits implementovaný v jazyku Rust, no v súčasnosti sú k dispozícii aj implementácie v ďalších jazykoch, všetky nájdeme v repozitári [8].

Biscuits bol inšpirovaný Macaroons, implementuje podobnú schému zabezpečenia aj funkcionality. Rovnako dovoľuje delegovať autorizáciu medzi službami a ľubovoľná entita vlastniaca token ho môže *zoslabiť* alebo aj *kontextovo obmedziť*. Hlavným rozdielom oproti Macaroons je, že Biscuits tokeny používajú na postupné vytváranie podpisu digitálne podpisy, konkrétne podpisovú funkciu Ed25519 [35]. Ďalším veľkým rozdielom je, že Biscuits používa na modelovanie práv, kontrol a dát v rámci tokenu špeciálny variant Datalogu bez negácie a na konkrétnych dátových typoch [40].

Zoslabenie a kontextové obmedzenie tokenu sa realizuje pomocou pridania nového bloku. Blok môže pridať aj služba tretej strany, v takom prípadne budeme hovoriť o externom bloku.

### 2.6.1 Štruktúra Biscuits

Samotný token sa skladá z blokov a niektorých globálnych informácií pre celý token. Globálne informácie sú identifikátor koreňového verejného kľúča a dôkaz, ktorý slúži na pridávanie ďalšieho bloku. Každý token obsahuje autoritatívny blok, ktorý pridala služba vytvárajúca token.

Každý blok obsahuje serializovaný datalogový program, podpis bloku a verejný kľúč. V prípade, že ide o externý blok, aj podpis bloku službou tretej strany a príslušný verejný kľúč služby tretej strany. Príklad základného Biscuits tokenu je na obrázku 2.8.

Datalogový program sa skladá z faktov, pravidiel, kontrol a pomocných informácií, detailnú schému formátu nájdeme v súbore schema.proto repozitára [8]. Fakty a pravidlá sú bežné datalogové fakty a pravidlá. Kontroly sú množiny datalogových dotazov. Dotaz je splnený práve vtedy, keď je jeho výsledok aspoň jeden fakt. Kontrola je splnená práve vtedy, keď je splnený aspoň jeden dotaz z množiny dotazov danej kontroly. Okrem toho definuje Biscuits aj politiky, ktoré vytvára entita validujúca token. Viac sa politikám budeme venovať v nasledujúcej podkapitole. Bloky a politiky môžu mať anotáciu definujúcu, ktorým blokom dôverujú a teda s faktami a pravidlami, ktorých blokov pracujú. Vždy platí, že blok verí autoritatívnemu bloku, sebe a informáciám v službe validujúcej token. Anotáciou môže blok definovať, že dôveruje aj všetkým predchádzajúcim blokom alebo všetkým blokom podpísaným konkrétnym verejným kľúčom. Posledná možnosť sa využíva pre integráciu služieb tretích strán do autorizačnej schémy. Datalogový program je serializovaný ako Protocol Buffer [24] podľa konkrétnej schémy definovanej v súbore schema.proto alebo v novšej verzii pomocou base64url kódovania [34].

### 2.6.2 Generovanie a delegácia autorizácie

Na vygenerovanie nového Biscuits tokenu potrebuje služba dvojicu súkromného a verejného kľúča. Do tokenu vloží verejný kľúč alebo jeho identifikátor ako koreňový ve-

```
Biscuits token

root_key_id: "my pk id"

authority: Block 0 {
 helper information
 facts: [user("Alice")]
 rules: []
 checks: []
 pk_1
 sig_0: sign("my secret key", Block 0 data)
}

proof: Proof {
 nextSecret: sk_1
}
```

Obr. 2.8: Príklad Biscuits tokena s jediným blokom.

rejný kľúč daného tokenu. Vytvorí autoritatívny blok, do ktorého vloží základné fakty a pravidlá platné pre tento token. Následne vygeneruje novú dvojicu kľúčov $pk_1$ a $sk_1$. Kľúč $pk_1$ vloží do autoritatívneho bloku a $sk_1$ do dôkazu tokenu. Kľúč $sk_1$ bude slúžiť na podpísanie ďalšieho bloku tokenu a $pk_1$ na overenie tohto podpisu. Nakoniec celý autoritatívny blok podpíše súkromným koreňovým kľúčom a podpis vloží do bloku.

Takto vytvorený token môže služba poslať iným službám a tieto môžu pridávať ďalšie bloky a tým obmedzovať autorizačné práva tokenu. Na vytvorenie $i$-teho bloku potrebuje služba vygenerovať dvojicu kľúčov $pk_{i+1}$, $sk_{i+1}$, kde $pk_{i+1}$ vloží do bloku a $sk_{i+1}$ do dôkazu tokenu. Blok môže pridať aj služba tretej strany, v takom prípade musí vložiť aj podpis bloku služby tretej strany a verejný kľúč služby tretej strany. Celý blok následne podpíše kľúčom $sk_i$, ktorý vybrala z dôkazu tokenu pred jeho nahradením.

Ľubovoľná služba môže token zapečatiť a znemožniť tak pridávanie nových blokov. Zapečatenie tokenu pozostáva z podpísania posledného bloku kľúčom z dôkazu tokenu. Tento podpis sa vloží do dôkazu tokenu.

### 2.6.3 Validácia Biscuits

Služba validujúca Biscuits token musí vedieť odvodiť koreňový verejný kľúč tokenu z jeho identifikátora. Služba token deserializuje a postupne zvaliduje všetky podpisy tokenov. Podpis $i+1$-vého bloku validuje pomocou kľúča $pk_{i+1}$ vloženého vnútri $i$-teho bloku. Podpis autoritatívneho bloku validuje pomocou koreňového verejného kľúča a podpis služby tretej strany pomocou verejného kľúča danej služby uloženého vnútri daného bloku. Ak je token zapečatený validuje podpis v dôkaze tokenu pomocou verejného kľúča v poslednom bloku, ak token nie je zapečatený skontroluje, či verejný kľúč v poslednom bloku tvorí dvojicu so súkromným v dôkaze tokenu.

Ak je token validný, prebehnú postupne všetky kontroly v blokoch a to tak, že

sa spustí daný datalogový program nad faktami a pravidlami podľa anotácie bloku a následne sa vykonajú dotazy v kontrolách. Fakty definuje aj samotná služba, napríklad vytvorí fakty na základe kontextu požiadavky. Príkladom takýchto faktov je typ operácie a IP adresa volajúceho. Token je validný iba ak sú splnené všetky kontroly.

Okrem kontrol v rámci blokov tokenu môže validujúca služba definovať ďalšie kontroly a politiky a pravidlá. Pravidlá odvádzajú nové fakty len z faktov odvodených autoritatívnym blokom prípadne samotnou službou. Kontroly a politiky pracujú len nad faktami odvodenými autoritatívnym blokom a službou samotnou. Tieto kontroly musia byť tiež všetky splnené, aby bol token validný.

Politiky definujú väčšie kontroly, taktiež pozostávajú zo zoznamu dotazov. Delia sa na dva typy - povoľovacie a zamietacie politiky. Politika je splnená ak je splnený aspoň jeden dotaz danej politiky. Pri validácii tokenu sa vyhodnocujú politiky postupne po jednej. Ak je splnená povoľovacia politika, token je validný. Ak je splnená zamietacia politika alebo nie je splnená žiadna politika, token je nevalidný. Vyhodnocovanie končí s prvou splnenou politikou.

### 2.6.4 Delegácia časti autorizácie na tretiu stranu

Každá služba môže využiť inú službu na nejakú časť autorizácie. Na tento účel slúžia externé bloky. Ak chce služba $A$ delegovať autorizáciu na službu $B$, vytvorí blok s anotáciou obsahujúcou verejný kľúč služby $B$ a vytvorí kontrolu, ktorá používa fakty, ktoré vie zabezpečiť len služba $B$. Následne pošle službe $B$ informácie potrebné pre autorizáciu danej požiadavky službou $B$, ktorá vykoná ľubovoľné operácie nutné pre autorizáciu danej požiadavky a ak je úspešná vráti službe $A$ nový externý blok obsahujúci potrebné fakty a kontroly.

# Kapitola 3

# Teoretické porovnanie API tokenov

V tejto kapitole porovnáme rôzne vlastnosti a parametre konkrétnych tokenov podľa informácií získaných z ich dokumentácií a iných zdrojov. Niektoré dáta sme už zhrnuli v kapitole 2 a preto ich tu nespomíname alebo nevysvetľujeme. Pri jednotlivých parametroch vysvetlíme ich význam a teda aj dôležitosť pri porovnávaní tokenov. Porovnávať budeme všetky štruktúrované tokeny popísané v kapitole 2 a nepriehľadný formát tokenu popísaný v podkapitole 1.7.1. Pre jednoduchosť budeme pod pojmom nepriehľadný token myslieť náhodný reťazec s podpisom vo forme hešovaného autentifikačného kódu.

Tabuľka 3.1: Teoretické porovnanie tokenov

| Vlastnosť | Nepriehľadný | JWT | PASETO | Fernet | Branca | Macaroons | Biscuits |
|---|---|---|---|---|---|---|---|
| Počet kryptografických funkcií | 1 | 30 | 6 | 1 | 1 | 1 | 1 |
| Určenie podpisového algoritmu z tokenu | ⊘ | ● | ● | ○ | ○ | ○ | ○ |
| Náchylnosť na útok pomýlením algortimu | ⊘ | ● | ◐ | ○ | ○ | ○ | ○ |
| Riešenie problému odvolania | ● | ○ | ○ | ○ | ○ | ◐ | ◐ |
| Náchylnosť na útok opakovaním | ⊘ | ● | ● | ● | ● | ◐ | ◐ |
| Ochrana dôvernosti | ⊘ | ◐ | ◐ | ● | ● | ○ | ○ |
| Overenie autentickosti a integrity hocikým | ○ | ◐ | ● | ● | ● | ● | ● |
| Zoslabenie tokenu hocikým | ⊘ | ⊘ | ⊘ | ⊘ | ⊘ | ● | ● |
| Autorizačná schéma v tokene | ⊘ | ◐ | ◐ | ◐ | ◐ | ● | ● |
| Bezstavová validácia | ⊘ | ● | ● | ● | ● | ● | ● |
| Štandardná validácia | ⊘ | ● | ● | ○ | ○ | ◐ | ● |
| Popularita | ⊘ | ● | ◐ | ◐ | ○ | ◐ | ○ |

Kapitola je štruktúrovaná podľa porovnávaných vlastností a jej výsledkom je tabuľka 3.1 zhrňujúca závery porovnania. Tabuľku uvádzame na začiatku kapitoly, aby si čitateľ mohol utvoriť istý prehľad o porovnávaných vlastnostiach. V tabuľke používame symboly ●, ◐ a ○ pre vyjadrenie stupňovitosti istej kvality tokenu. Symbol ● znamená, že vlastnosť je pre daný token typická, častá alebo dôležitá. Symbol ○ vyjadruje presný opak a symbol ◐ niečo medzi tým. Symbol ⊘ znamená, že vlastnosť

sa nedá pre daný token posúdiť alebo z nejakého dôvodu nemá zmysel o nej uvažovať a porovnávať ju. Na začiatku každej podkapitoly uvedieme časť tabuľky, ktorá sa venuje vlastnostiam porovnávaným v danej podkapitole.

## 3.1 Bezpečnosť

V rámci porovnávania bezpečnosti tokenov nebudeme detailne rozoberať bezpečnosť jednotlivých kryptografických funkcií. Detaily ohľadom týchto funkcií je možné nájsť v ich citovaných dokumentáciách. Všetky tokeny ponúkajú možnosť použiť kryptografické funkcie, ktoré sú všeobecne považované za bezpečné. Zhrnutie výsledkov porovnania bezpečnosti tokenov je možné nájsť v tabuľke 3.2.

Tabuľka 3.2: Bezpečnosť tokenov
(vybrané riadky tabuľky 3.1 relevantné pre bezpečnosť)

| Vlastnosť | Nepriehľadný | JWT | PASETO | Fernet | Branca | Macaroons | Biscuits |
|---|---|---|---|---|---|---|---|
| Počet kryptografických funkcií | 1 | 30 | 6 | 1 | 1 | 1 | 1 |
| Určenie podpisového algoritmu z tokenu | ⊘ | ● | ● | ○ | ○ | ○ | ○ |
| Náchylnosť na útok pomýlením algortimu | ⊘ | ● | ◐ | ○ | ○ | ○ | ○ |
| Riešenie problému odvolania | ● | ○ | ○ | ○ | ○ | ◐ | ◐ |
| Náchylnosť na útok opakovaním | ⊘ | ● | ● | ● | ● | ◐ | ◐ |
| Ochrana dôvernosti | ⊘ | ◐ | ◐ | ● | ● | ○ | ○ |
| Overenie autentickosti a integrity hocikým | ○ | ◐ | ● | ● | ● | ● | ● |

Zameriame sa na porovnanie kryptografických primitív a z nich vyplývajúcich bezpečnostných kvalít a na náchylnosť na zraniteľnosti vyplývajúce zo špecifikácie tokenov. Konkrétne rozoberieme tri bezpečnostné problémy:

- Útok pomýlením algoritmu (angl. algorithm confusion attack) – útočník donúti overovaciu službu použiť nesprávny algoritmus na overenie podpisu tokenu.
- Útok opakovaním (angl. replay attack) – útočník odchytí token a následne ho opakovane používa na autorizáciu vlastných požiadaviek. Tento útok je priamo spojený s hlavnou nevýhodou používania tokenov v autentifikačnej a autorizačnej schéme a to problémom odvolania (angl. revocation).
- Problém odvolania – problém odvolania spočíva v schopnosti služby zneplatniť vydané tokeny. Napríklad po odhlásení používateľa alebo po zistení, že token bol zneužitý.

### 3.1.1 Kryptografické primitíva

Pri tokenoch rozoznávame tri kryptografické primitíva a to digitálne podpisy využívajúce asymetrické šifrovanie, symetrické šifrovanie a hešovanie s kľúčom. Výstupom

hešovania s kľúčom je hešovaný autentifikačný kód.

Symetrické šifrovanie sa v rámci nami porovnávaných tokenov využíva na šifrovanie obsahu tokenu a teda na ochranu dôvernosti informácií uložených v tokene. Digitálny podpis a hešovanie s kľúčom zaručujú ochranu autenticity a integrity tokenu. Rozdiel v použití digitálneho podpisu a hešovania s kľúčom je v tom, že v prípade digitálneho podpisu ide o asymetrické šifrovanie, teda podpis vie overiť ľubovoľná entita, ktorá pozná verejný kľúč tvoriaci dvojicu so súkromným kľúčom, ktorým bol token podpísaný. Takýto verejný kľúč je zväčša verejne dostupný a vie ho získať ľubovoľná entita. V prípade hešovania s kľúčom ide o symetrickú kryptografiu, pravosť hešovaného autentifikačného tokenu vie overiť len entita, ktorá pozná tajný kľúč, ktorým bol token zahešovaný, čo je často len entita, ktorá token vytvorila.

Výhodou digitálneho podpisu teda je, že autenticitu a integritu tokenu môže overiť ľubovoľná entita. Výhodou hešovania s kľúčom je, že je rýchlejšie pri generovaní kľúča a generovaní aj overovaní podpisu ako algoritmy pre digitálne podpisy, aj keď v prípade niektorých algoritmov nad eliptickými krivkami je rýchlosť porovnateľná [41]. Pri porovnaní iba algoritmov definovaných v JWA [1] je hešovanie s kľúčom vždy rýchlejšie.

V prípade JWT si môžeme vybrať, či budeme používať digitálny podpis alebo hešovanie s kľúčom pomocou nastavenia oprávnenia *alg* v hlavičke na požadovanú hodnotu. Všetky možnosti hodnôt oprávnenia *alg* definuje JWA [29]. Štandard ponúka aj možnosť *alg=none*, v tomto prípade nezaručuje JWT žiadne bezpečnostné kvality a je to jedna zo známych zraniteľností [46] JWT. Ak služba akceptuje aj JWT s *alg=none* ako platné tokeny, útočník jednoducho zamení hodnotu *alg='čokoľvek'* na *alg=none*, odstráni podpis z tokenu a môže ľubovoľne zmeniť token, napríklad si zvýši autorizačné práva. Bezpečné implementácie JWT, by nikdy nemali tokeny s *alg=none* považovať za platné.

PASETO využíva v prípade lokálneho využitia hešovanie s kľúčom a v prípade verejného využitia digitálny podpis. Fernet, Branca a Macaroons využívajú hešovanie s kľúčom a Biscuits využíva digitálny podpis. Nepriehľadný token sme pre potreby tejto kapitoly definovali s použitím hešovania s kľúčom.

Symetrické šifrovanie a z neho vyplývajúcu ochranu dôvernosti umožňujú tokeny JWT, konkrétne vo forme JWE, PASETO s lokálnym využitím, Fernet a Branca. Biscuits a Macaroons neposkytujú žiadnu ochranu dôvernosti. Nepriehľadný token tiež neposkytuje ochranu dôvernosti v rámci tokenu, no z definície nenesie žiadnu informáciu a všetky dôverné údaje sú uložené na strane servera a asociované s tokenom. Teda na strane klienta nie je dôvernosť čoho chrániť.

### 3.1.2 Útok pomýlením algoritmu

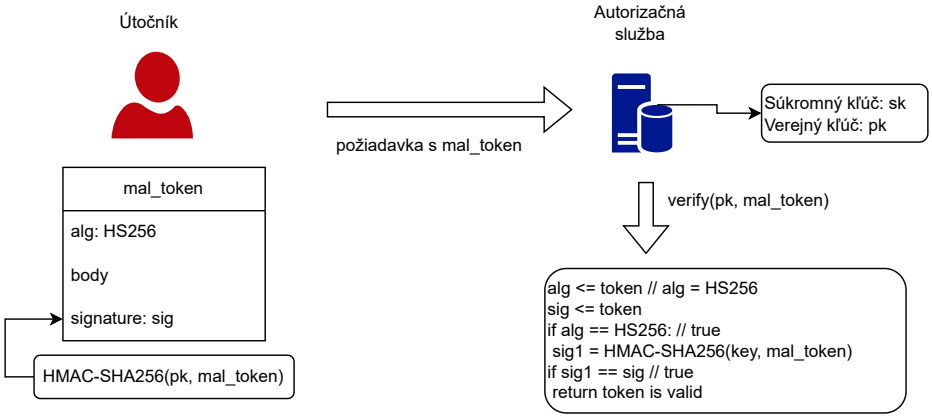
V prípade podpisov tokeny používajú digitálne podpisy na základe asymetrického šifrovania a teda dvojicu súkromného a verejného kľúča alebo hešovanie s kľúčom, ktorý je tajný. Jediný kľúč, ku ktorému má útočník ľahký prístup je verejný kľuč (označme ho $pk$). Útok pomýlením algoritmu potom prebehne tak, že útočník podpíše token hešovacou funkciou s kľúčom, kde ako kľúč použije získaný verejný kľúč $pk$. Následne oklame overovaciu službu, aby token overila pomocou hešovacej funkcie s kľúčom, kde ako kľúč použije $pk$. Ako útočník môže oklamať overovaciu službu popíšeme neskôr v podkapitole na príklade konkrétnej zraniteľnosti JWT. Takýmto spôsobom overovacia služba potvrdí platnosť ľubovoľného tokenu, ktorý jej útočník podvrhne. Aby bol tento útok úspešný, musí overovacia služba používať digitálne podpisy ako podpis tokenu a zároveň podporovať aj vytváranie podpisu tokenu pomocou hešovania s kľúčom.

Existuje viacero spôsobov ako predchádzať útokom pomýlením algoritmu. Najspoľahlivejším spôsobom je podpora jediného kryptografického primitíva na podpis tokenu, napríklad jedine digitálny podpis alebo jedine hešovanie s kľúčom. Takto útočník jednoducho nemá ako oklamať overovaciu službu, ktorý algoritmus má použiť pri overovaní, lebo pozná len jeden.

V prípade použitia viacerých kryptografických primitív sa dá predchádzať týmto útokom pomocou vloženia identifikátora kľúča, ktorým sa overí podpis, do tokenu. Následne pri overovaní služba zistí, ktorým algoritmom bol token podpísaný. Taktiež z pridaného identifikátora odvodí, ktorý kľúč má použiť na overenie, ak je to verejný kľúč z dvojice kľúčov pre digitálne podpisovanie, ale zistený podpisový algoritmus z tokenu je hešovanie s kľúčom, vyhodnotí token za neplatný. Útočník už nedokáže oklamať službu aby použila zlý algoritmus na overenie podpisu, lebo ak by sa o to pokúsil nebude sedieť identifikátor kľúča s podpisovým algoritmom. Úspešnosť tejto metódy ochrany nezávisí len od špecifikácie tokenu, ale najmä od jeho konkrétnej implementácie, pretože záleží na implementácii ako bude pracovať s identifikátorom kľúča a či vôbec vyžaduje jeho použitie.

Tokeny využívajúce jediné kryptografické primitívum sú Fernet, Branca, Biscuits a Macaroons. Sú teda bezpečné proti útokom pomýlením algoritmu, no všetky nejakým spôsobom podporujú budúce verzionovanie tokenu, ktoré môže teoreticky priniesť aj nové kryptografické primitíva. Preto do budúcnosti môžu byť zraniteľné útokom pomýlením algoritmu, ak nebudú implementovať inú ochranu voči tomuto útoku. V súčasnosti už Macaroons aj Biscuits vyžadujú vloženie identifikátora kľúča do tokenu v rámci ich formátov, teda aj v prípade podpory ďalších kryptografických primitív budú bezpečné voči útokom pomýlením algoritmu.

Jediné kryptografické primitívum využíva aj nepriehľadný token, ale v tomto prípade to nie je veľmi dôležité, lebo nenesie žiadnu informáciu a všetky autorizačné údaje




Obr. 3.1: Schéma útoku pomýlením algoritmu [46].

sú uložené v stave overovacej služby. Teda aj v prípade úspešného útoku pomýlením algoritmu, služba síce vyhodnotí podpis tokenu za platný, no ak nezodpovedá žiadnym dátam uloženým v stave služby, tak služba neautorizuje útočníkovu požiadavku, na ktorú nemá inak právo. Nepriehľadný token je teda absolútne bezpečný voči útokom pomýlením algoritmu.

Viac kryptografických primitív využívajú JWT a PASETO. Oba podporujú digitálne podpisy aj hešovanie s kľúčom na podpisovanie tokenu. V prípade JWT bol útok pomýlením algoritmu jednou zo známych zraniteľností v niektorých implementáciách [46]. Konkrétne útok prebiehal tak, že útočník si vybral službu používajúcu digitálny podpis. Získal jej verejný kľúč *pk*, vytvoril podvodný token *mal_token* a do jeho hlavičky zapísal *alg=HS256* (HS256 označuje funkciu HMAC-SHA256), následne funkciou HMAC-SHA256 podpísal token s využitím *pk* ako tajného kľúča pre funkciu. Služba, ktorá využívala zraniteľnú knižnicu a na podpisovanie iba digitálny podpis overila token zavolaním funkcie knižnice, napríklad *verify(pub_key, mal_token)*, lebo si myslela, že overuje token podpísaný digitálnym podpisom a na jeho overenie teda treba verejný kľúč *pk*. Knižnica následne prečítala z tokenu, že má overiť podpis pomocou HMAC-SHA256 a z prvého argumentu funkcie *verify*, že má pri tom využiť *pk* ako kľúč. Toto overenie bolo samozrejme úspešné, lebo token bol naozaj podpísaný funkciou HMAC-SHA256 s kľúčom *pk*. Popísaný útok je schematicky znázornený na obrázku 3.1. V súčasnosti už tieto konkrétne implementácie zaviedli ochranu voči útoku pomýlením algoritmu (pomocou identifikátora kľúča), no nič nezaručuje, že neexistujú iné implementácie s touto zraniteľnosťou. Popísaný útok dáva útočníkovi možnosť získať ľubovoľné práva, lebo celý *mal_token* môže vytvoriť presne tak ako potrebuje. Išlo teda o veľmi nebezpečný útok.

PASETO využíva verzionovanie tokenu ako prevenciu voči útoku pomýlením algoritmu. Ide o podobnú techniku ako pri vložení identifikátora kľúča do tokenu, no navyše vyžaduje kontrolu formátu kľúča. Špecifikácia PASETO [58] prikazuje každej

knižnici, ktorá chce implementovať PASETO, logicky rozlišovať medzi kľúčmi určenými pre rôzne podpisové funkcie. V rámci špecifikácie sa ochrane voči útoku pomýlením algoritmu venuje dedikovaný dokument [57]. Kľúč k ľubovoľnému algoritmu musí byť vždy uložený tak, aby sa dalo jasne určiť, pre ktorý algoritmus sa má použiť. Tento cieľ sa dá dosiahnuť napríklad tak, že sa kľúč uloží v nejakej štruktúre spolu s verziou a využitím tokenu. Následne pri validácii podpisu tokenu musí prebehnúť kontrola rovnosti verzie a využitia v kľúči s verziou a využitím v tokene. Podobne ako pri JWT popísaná ochrana bude úspešná len v prípade, že ju knižnice implementujúce PASETO budú využívať. Výhodou PASETO je, že sa ochrana vyžaduje v špecifikácii, teda každá knižnica, ktorá chce úspešne implementovať špecifikáciu PASETO ju musí implementovať. V prípade JWT, štandard [32] nevyžaduje využitie identifikátora kľúča.

S útokom pomýlením algoritmu súvisí aj celkový počet podporovaných podpisových a šifrovacích algoritmov. V tomto prípade ide skôr o problém pomýlenia algoritmu. Kryptografické algoritmy majú podobné názvy, ľahko sa teda môže stať, že sa programátor pri ich výbere pomýli a napríklad pri generovaní a validácii tokenu použije rôzny algoritmus. Toto nevedie priamo k zraniteľnostiam, no môže mať za následok nepredvídateľné správanie systému. Principiálne ide o útok pomýlením algoritmu pri každej validácii tokenu, lebo token sa nevaliduje očakávaným algoritmom. Pre útočníka je však ťažšie (potenciálne nemožné) vyrobiť falošný platný token.

Nepriehľadný token, Fernet, Branca, Macaroons aj Biscuits používajú len jednu kombináciu podpisového a šifrovacieho algoritmu. PASETO však vo všetkých potenciálnych kombináciách verzie a využitia používa dokopy 6 algoritmov. JWT dokonca až 30.

### 3.1.3 Útok opakovaním a problém odvolania

Útok opakovaním a problém odvolania sú úzko späté pojmy. Konkrétne, riešenie problému odvolania je čiastočnou ochranou voči útoku opakovaním. Ak by služba vedela okamžite zneplatniť ľubovoľný ňou vydaný token, útok opakovaním by mohla ihneď po odhalení zastaviť zneplatnením tokenov využitých pri útoku. Úplne predísť útoku opakovaním je pri využívaní nositeľských tokenov na autorizáciu nemožné. Z definície nositeľský token autorizuje požiadavku, ktorú sprevádza, ak je sám platný. Teda ak ho dokáže útočník získať a použiť v správnom kontexte, bude úspešný.

Hlavnou príčinou problému odvolania je udržiavanie stavu v tokene a nie v databáze autorizačnej služby. Ak by si služba udržiavala stav o vydaných tokenoch, jednoducho by token, ktorý chce odvolať, označila za neplatný. Tento prístup sa dá využiť s nepriehľadným tokenom, lebo v jeho prípade si už aj tak musí autorizačná služba udržiavať stav o vydaných tokenoch.

Pri validácii ostatných tokenov sa autorizačná služba spolieha len na informácie

v tokene a uložené kľúče na overovanie podpisov, prípadne dešifrovanie tela tokenu. Samozrejme môže štruktúrovaný token obsahovať identifikátor a podľa tohto identifikátora si o ňom môže autorizačná služba udržiavať stav, či je token platný. Napríklad JWT má pre tento účel štandardom dané oprávnenie *jti* (JWT ID). Takto by však použitie tokenov na autorizáciu stratilo signifikantnú výhodu oproti iným schémam zabezpečenia.

Autorizačná služba by si nemusela o každom tokene pamätať, či je platný alebo nie. Stačí si jej pamätať množinu platných (povoľovací zoznam – angl. allowlist) alebo množinu odvolaných, ale ešte nie exspirovaných (zamietací zoznam – angl. denylist) tokenov. Týmto sa zmenší veľkosť uloženého stavu.

Iným, často využívaným, riešením je vydávať tokeny s krátkou platnosťou. Platnosť tokenu môže byť uložená v samotnom tokene vo forme časovej pečiatky a času platnosti alebo času exspirácie, teda nekladie nároky na stav autorizačnej služby. Útočník má teda málo času na získanie a zneužitie konkrétneho tokenu s krátkou platnosťou, čo mu výrazne sťažuje realizáciu útoku. Ak má útočník možnosť dlhodobo zachytávať tokeny, vydané používateľovi počas jeho komunikácie so serverom. Tak pri odhalení útoku opakovaním sa mu klient môže brániť tým, že sa odhlási. Útok bude potom určite zastavený po krátkom čase, kedy vyprší platnosť tokenom, ktoré útočník získal a ďalšie nevie zachytiť, lebo používateľ sa odhlásil a už nekomunikuje so serverom.

Nejde teda o dokonalú ochranu pred útokom opakovaním, ale skomplikovanie jeho vykonania útočníkovi a zníženie jeho dopadov. Túto ochranu podľa špecifikácie podporujú tokeny Fernet a Branca. Formáty oboch musia obsahovať časovú pečiatku vo formáte počet sekúnd od 1.1.1970 v UTC časovej zóne. V prípade Branca tokenu ide o 32 bitové číslo, no vo formáte bez znamienka, čo posúva problém 2038 [62] na rok 2106. Stále však ide o potenciálny problém za cenu ušetrenia 4B dát v tokene. Všetky ostatné tokeny vedia túto metódu ľahko implementovať. Využíva ju aj populárny protokol OAuth 2.0 [26].

Macaroons a Biscuits poskytujú možnosť pridávať pravidlá tretích strán, takto môžu pridať pravidlo na službu udržujúcu povoľovací alebo zamietací zoznam tokenov. Týmto odľahčia autorizačnú službu od udržiavania stavu o vydaných tokenoch a zároveň budú poskytovať úplnú ochranu pred útokom opakovaním. Stav však z autorizačnej schémy nezmizol, iba sa presunul do inej služby a nutnosť získať dôkaz o tom, že token nebol odvolaný sa presunul z autorizačnej služby na klienta posielajúceho požiadavku.

Tabuľka 3.3: Flexibilita tokenov
(vybrané riadky tabuľky 3.1 relevantné pre flexibilitu)

| Vlastnosť | Nepriehľadný | JWT | PASETO | Fernet | Branca | Macaroons | Biscuits |
|---|---|---|---|---|---|---|---|
| Zoslabenie tokenu hocikým | ⊘ | ⊘ | ⊘ | ⊘ | ⊘ | ● | ● |
| Autorizačná schéma v tokene | ⊘ | ◐ | ◐ | ◐ | ◐ | ● | ● |
| Bezstavová validácia | ⊘ | ● | ● | ● | ● | ● | ● |
| Štandardná validácia | ⊘ | ● | ● | ○ | ○ | ◐ | ● |

## 3.2 Flexibilita

Pod flexibilitou tokenu rozumieme flexibilitu a škálovateľnosť schémy zabezpečenia využívajúcu daný token. To napríklad znamená, akú mohutnú a členitú schému zabezpečenia ňou vieme vyjadriť, aká veľká časť autorizačnej schémy môže byť dynamicky vyjadrená v tokene a aká staticky v stave autorizačnej služby. Ďalej môžeme flexibilitu merať pomocou možnosti nezávislého podieľania sa viacerých služieb na autorizácii a možnosti jednoducho a bezpečne delegovať autorizáciu na iné služby. Časť tabuľky 3.1 venujúca sa flexibilite je zobrazená v tabuľke 3.3.

### 3.2.1 Bezstavovosť autorizačnej služby

V predchádzajúcej podkapitole 3.1.3 sme naznačili, že autorizačná služba v schéme zabezpečenia využívajúcej tokeny, nemusí byť vždy bezstavová, no v tejto podkapitole budeme uvažovať schému bez udržiavania povoľovacích alebo zamietacích zoznamov vydaných tokenov. Bezstavovosť je jednoznačne významný prvok pri riešení škálovateľnosti ľubovoľnej schémy zloženej z viacerých služieb. Dovoľuje nám jednoducho pridávať do schémy nové služby schopné vykonávania autorizácie tokenov. Tieto nové služby potrebujú poznať teoreticky len príslušné kľúče pre overenie podpisu tokenu a prípadne dešifrovanie tela tokenu. Samozrejme v reálnych systémoch je autorizácia výrazne členená a pre úspešnú autorizáciu aj jednoduchej požiadavky treba overiť a porovnať viacero údajov z tokenu. Okrem nepriehľadného tokenu je možné všetky nami porovnávané tokeny validovať bezstavovo.

### 3.2.2 Delegácia autorizácie

Delegácia autorizácie je úzko spojená s možnosťou pridávania pravidiel tretích strán. V kapitole 2 sme o možnosti pridávania pravidiel alebo blokov tretích strán hovorili len pri Macaroons a Biscuits, no teoreticky je možné pridávať pravidlá tretích strán aj pri iných tokenoch.

V prípade nepriehľadného tokenu by to bolo veľmi nepraktické, lebo by museli byť zapísané v stave služby, ktorá token vydala a pri kontrole ich naplnenia, by museli

byť prečítané z tohto stavu. Následne by autorizačná služba sama získavala potvrdenie o splnení pravidla od príslušnej tretej strany, čo zbytočne zaťažuje autorizačnú službu.

Pri JWT, PASETO, Fernet a Branca tokenoch je možné pridávať do tela tokenu ľubovoľnú informáciu, teda aj pravidlo tretej strany. Ani jeden z týchto tokenov však podľa špecifikácie nie je prispôsobený na takéto pravidlá. Preto ich pravdepodobne žiadna knižnica nepodporuje a v prípade rozhodnutia používať pravidlá tretích strán s týmito tokenmi takým flexibilným spôsobom ako pri Macaroons alebo Biscuits, by bolo treba implementovať celú logiku využívania tokenov v schéme, vrátane ich generovania a validácie. To môže byť náročné a viesť ku chybám a z nich vyplývajúcich zraniteľnostiam.

### 3.2.3 Autorizačná schéma v tokene

V tejto časti predpokladáme, že podpis tokenu je už úspešne validovaný.

Nepriehľadný token z definície nemôže obsahovať žiadnu informáciu, teda ani autorizačnú logiku. Ostatné tokeny sú štruktúrované a nesú v sebe rôzne informácie. Väčšinou ide o statické informácie, napríklad časová pečiatka, administrátorské práva, vlastník tokenu, identifikátor služby, ktorá ho vydala a tak ďalej. Tieto informácie sa doplnia o kontext požiadavky, napríklad IP adresa klienta a aktuálny čas. Následne sa podľa logických pravidiel autorizácie naprogramovaných v autorizačnej službe vyhodnotí, či sa požiadavka autorizuje.

Dynamické informácie dovoľujú autorizačnej službe overiť logické pravidlá, o ktorých nemusela vedieť pri vydávaní tokenu a na ich vyhodnotenie nepotrebuje vlastný stav. Napríklad pravidlá tretích strán v Macaroons tokene dovoľujú preniesť požiadavku na autentifikáciu alebo autorizáciu inou službou na klienta a danú službu. Autorizačná služba potom len validuje dôkaz o autorizácii treťou stranou predložený klientom spolu s požiadavkou.

Biscuits prinášajú ešte väčšiu mieru prenesenia autorizačnej schémy do tokenu. Keďže ich autorizácia prebieha ako vyhodnotenie datalogového programu po blokoch tokenu, každý blok môže zaviesť vlastné symboly, fakty a kontroly v jednotnej syntaxi. Potom služba rozumejúca tejto syntaxi a schopná validovať autoritatívny blok tokenu, dokáže vyhodnotiť celý datalogový program a validovať token.

Aj Macaroons, aj Biscuits umožňujú ľubovoľnej službe pridávať obmedzenia do tokenu bez komunikácie s autorizačnou službou, čo prenáša časť autorizačnej schémy do tokenu. No v prípade Macaroons musí autorizačná služba vedieť pravidlá tvoriace dané obmedzenia vyhodnotiť podľa naprogramovanej logiky vyhodnocovania pravidiel. Špecifikácia Macaroons [7] nám v tomto prípade nijak nepomáha, lebo formát aj štruktúru pravidiel prenecháva na konkrétnu implementáciu. Naopak služba autorizujúca Biscuits, dokáže vyhodnotiť aj kontroly, ktoré nepozná, ak sú napísané v jednotnej

syntaxi.

### 3.2.4 Štandardná validácia

V predchádzajúcej sekcii sme naznačili problém Macaroons s validáciou pravidiel, ktoré sú v tokene, ale autorizačná služba ich nepozná. Rozšírme toto pozorovanie na všetky tokeny. Opäť budeme uvažovať token s už úspešne validovaným podpisom.

Štandardnosť je žiadaná kvalita pri každej technológii. Prináša jednoznačnosť použitia aj zápisu a programátorovi uľahčuje prácu, lebo nemusí vytvárať nové rozhrania s vlastnými názvami funkcií a ich parametrov. Taktiež zabezpečuje použiteľnosť toho istého kódu medzi viacerými aplikáciami. Ak aplikácia dodržuje štandard je zaručená jej interoperabilita s inými aplikáciami, ktoré ho tiež dodržujú. Pri validácii tokenov budeme uvažovať, že validácia je štandardná, ak je špecifikáciou dané, ktoré hodnoty majú byť v tokene uvedené, akou syntaxou sú zapísané a ako ich vyhodnotiť.

Nepriehľadný token sa validuje čisto podľa informácií v stave autorizačnej služby a kontextu požiadavky, teda jeho validácia nie je štandardná. Závisí od autorizačnej služby aké informácie si k tokenu pamätá a ako ich vyhodnotí pri validovaní požiadavky s daným tokenom. JWT a PASETO obsahujú vo svojom tele oprávnenia, z ktorých všetky bežné sú definované špecifikáciou. Stačí, aby sa ich implementácie riadili špecifikáciou a vyhodnocovanie základnej autorizačnej logiky bude všade rovnaké, teda štandardné.

Pri Fernet a Branca tokenoch sa nedá hovoriť o štandardnej validácii. Ide síce o štruktúrované tokeny, ale formát ich tela nie je definovaný špecifikáciou a je prenechaný na konkrétne implementácie.

Naopak špecifikácia Biscuits uvádza presný formát a syntax datalogového programu, ktorý treba používať v tokenoch.

## 3.3 Popularita a využiteľnosť

Popularitu tokenov posúdime podľa existencie používaných a udržiavaných knižníc v populárnych programovacích jazykoch pri programovaní serveru. My sme sa rozhodli vybrať jazyky JavaScript, Python a Java.

Tabuľka 3.4: Vybrané implementácie tokenov
(dáta pochádzajú z 19.4.2023)

| Token | Programovací jazyk | Implementácia | Počet hviezdičiek | Počet commitov | Posledný commit |
|---|---|---|---|---|---|
| JWT | JavaScript | panva/jose [55] | 3,3k | 1232 | niekoľko dní |
| | Python | jpadilla/pyjwt [47] | 4,6k | 782 | niekoľko dní |
| | Java | jwtk/jjwt [27] | 8,9k | 514 | niekoľko dní |
| PASETO | JavaScript | panva/paseto [56] | 287 | 113 | 3 mesiace |
| | Python | dajiaji/pyseto [13] | 34 | 601 | niekoľko dní |
| | Java | nbaars/paseto4j [5] | 28 | 196 | 3 týždne |
| Fernet | JavaScript | zoran-php/fernet-nodejs [14] | 0 | 14 | 2 týždne |
| | Python | pyca/cryptography [51] | 5,5k | 40 | 3 týždne |
| | Java | l0s/fernet-java8 [43] | 32 | 1335 | niekoľko dní |
| Branca | JavaScript | tuupola/branca-js [59] | 91 | 51 | rok |
| | Python | tuupola/pybranca [60] | 46 | 41 | 2 roky |
| | Java | bjoernw/jbranca [63] | 4 | 7 | 5 rokov |
| Macaroons | JavaScript | js-macaroon [48] | 39 | 100 | 3 roky |
| | Python | rescrv/libmacaroons [16] | 479 | 90 | 2 roky |
| | Java | nitram509/jmacaroons [36] | 122 | 316 | 2 týždne |
| Biscuits | JavaScript | neexistuje | | | |
| | Python | neexistuje | | | |
| | Java | biscuit-java [10] | 24 | 408 | 3 mesiace |

Popularitu nepriehľadného tokenu nedokážeme takto posúdiť, lebo nejde o konkrétny token a jeho implementácia sa veľmi líši od konkrétnej aplikácie. Neexistujú teda knižnice, ktoré ho implementujú a vedeli by sme ich použiť na porovnanie.

Používanosť a udržiavanosť knižníc budeme hodnotiť podľa informácií o knižniciach na platforme GitHub. Konkrétne podľa počtu hviezdičiek, dátumu posledného commitu a celkového počtu commitov. Jednotlivé knižnice a informácie o nich sú uvedené v tabuľke 3.4, uvedené dáta pochádzajú z 19.4.2023. V prípade Python knižnice pre prácu s Fernet tokenom, ide o knižnicu v rámci repozitára pyca/cryptography, teda oficiálnej knižnice pre kryptografiu v Pythone. Počet commitov a údaj o poslednom commite uvádzame iba pre súbor fernet.py implementujúci Fernet. Hviezdičky sa, ale udeľujú celému repozitáru, teda toto číslo je skresľujúce, keďže ide o veľký repozitár, ktorého len malá časť implementuje Fernet.

Používanosť sme vyhodnotili na základe počtu hviezdičiek a to konkrétne udelením bodu za každú splnenú úroveň, z úrovní: (1) $\geq$ 10, (2) $\geq$ 100 a (3) $\geq$ 1000. Podobne udržiavanosť ako súčet bodov z ostatných dvoch údajov. Konkrétne za splnenie úrovní: (1) $\geq$ 10, (2) $\geq$ 50, (3) $\geq$ 100, (4) $\geq$ 1000 v počte commitov a úrovní: (1)*niekoľko dní*, (2)*niekoľko týždňov*, (3)*niekoľko mesiacov* a (4)*niekoľko rokov* v časovom intervale od posledného commitu. Získané body tokenov v jednotlivých programovacích jazykov sme následne sčítali. Výsledky porovnania sú uvedené v tabuľke 3.5.

Do výslednej tabuľky porovnania tokenov 3.1 sme preniesli získané body do symbolov ○, ◐ a ● podľa toho, či token získal $\geq$ 5, $\geq$ 15 alebo $\geq$ 30 bodov.

Tabuľka 3.5: Popularita tokenov

| Token | Používanosť [max. 9] | Udržiavanosť [max. 24] |
|---|---|---|
| JWT | 9 | 22 |
| PASETO | 5 | 18 |
| Fernet | 4 | 16 |
| Branca | 2 | 6 |
| Macaroons | 5 | 13 |
| Biscuits | 1 | 5 |

# Kapitola 4

# Testovacie API

V tejto kapitole navrhneme, implementujeme a otestujeme jednoduché rozhranie so schémou zabezpečenia využívajúcou jednotlivé tokeny popísané v kapitole 2 a teoreticky porovnané v kapitole 3. Nepriehľadný token budeme, na rozdiel od prechádzajúcej kapitoly, chápať iba ako náhodný reťazec. Cieľom implementácie je porovnať rýchlosť spracovania požiadaviek využívajúcich rôzne tokeny na autorizáciu a jednoduchosť práce s knižnicami implementujúcimi jednotlivé tokeny vo vybranom programovacom jazyku. Implementácia môže zároveň slúžiť ako základ pre implementovanie jednoduchej schémy zabezpečenia a jej integrovanie s vybranými knižnicami. Zdrojový kód je dostupný na platforme GitHub[1] a v prílohe A.

## 4.1 Použité technológie

Testovacie API sme implementovali v jazyku JavaScript, konkrétne pomocou prostredia Node.js. Použili sme knižnicu Express.js, ktorá umožňuje ľahko implementovať jednoduchý server.

Na vytváranie a validáciu jednotlivých tokenov sme použili knižnice pre JavaScript z tabuľky 3.4, ktoré sme použili pri porovnaní popularity tokenov. Implementovali sme aj NONE token, reprezentujúci samotnú réžiu okolo vytvorenia a spracovania volania na API. Pre Biscuits neexistuje knižnica pre JavaScript, preto sme Biscuits v tejto kapitole neporovnávali. Prácu s nepriehľadným tokenom sme implementovali sami. Informácie spojené s nepriehľadným tokenom si ukladáme v databáze. Využili sme jednoduchú databázu sqlite. Klienta vykonávajúceho požiadavky na rozhranie sme tiež naprogramovali v Node.js. Pre interpretáciu nameraných výsledkov sme použili jazyk Python a knižnice: numpy pre prácu s vektormi čísel, matplotlib pre generovanie grafov a jinja2 pre prácu s HTML šablónou.

[1] Zdrojový kód je dostupný na adrese:
https://github.com/jitka1997/bachelor-thesis/tree/main/simpleAPI

## 4.2 Popis rozhrania

Vytvorili sme rozhranie s dvoma koncovými bodmi *signin* a *welcome*. Prvý z nich slúži na získanie tokenu klientom a druhý na vykonanie požiadavky, ktorá bude úspešná len v prípade, že bude obsahovať platný token, ktorý rozhranie úspešne validuje. Na získanie tokenu sa klient musí úspešne autentifikovať pomocou základnej HTTP autentifikácie, teda zaslaním base64url [34] zakódovaného mena a hesla v autentifikačnej hlavičke požiadavky.

Rozhranie overí korektnosť prihlasovacích údajov a ak sú správne, tak vygeneruje token. V reálnom systéme by boli prihlasovacie údaje uložené v databáze, kde by ich rozhranie overilo. My pre jednoduchosť databázu používateľov simulujeme pomocou načítania objektu s používateľmi zo súboru usersDB. V tomto objekte sú používateľské mená a heslá uložené ako dvojice kľúč a hodnota.

Po úspešnej autentifikácii vygeneruje rozhranie token a vráti ho klientovi. Klient následne môže token použiť na vykonanie požiadavky na koncový bod *welcome*. Rozhranie overí platnosť tokenu a ak je platný, vykoná požiadavku, teda vráti klientovi uvítaciu správu obsahujúcu jeho prihlasovacie meno získané z tokenu. Ak token nebol zaslaný alebo zlyhá jeho validácia, rozhranie vráti klientovi chybový kód.

## 4.3 Obsah a generovanie tokenu

Obsahovo vytvárame token nesúci štyri informácie overované pri autorizácii. Konkrétne čas vypršania platnosti tokenu, prihlasovacie meno používateľa, identifikátor vydavateľa tokenu a identifikátor prijímateľa tokenu. Čas vypršania platnosti tokenu je 5 minút od jeho vydania, identifikátory vydavateľa a prijímateľa sú pevne stanovené hodnoty a prihlasovacie meno je získané z prihlasovacích údajov klienta.

Tieto informácie sme do obsahu tokenu pridali, aby sme demonštrovali využitie údajov z tokenu na autorizáciu požiadavky, prípadne identifikáciu používateľa. Do konkrétneho tokenu sme vložili informácie spôsobom, ktorý nám umožnila špecifikácia daného tokenu a použitá knižnica implementujúca token. Konkrétne:

- Nepriehľadný token – do databázy sme vložili riadok obsahujúci samotný token ako identifikátor a ostatné informácie ako hodnoty jednotlivých stĺpcov.
- JWT a PASETO – meno používateľa sme vložili do tela tokenu ako oprávnenie *username* a ostatné informácie ako štandardné oprávnenia.
- Fernet – všetky informácie sme vložili ako serializovaný JSON objekt do tela tokenu. Fernet síce obsahuje samostatnú časovú pečiatku, no vybraná knižnica nepodporuje jej použitie na validáciu časovej platnosti tokenu.

- Branca – čas vypršania platnosti tokenu sme zaznamenali ako časovú pečiatku vytvorenia tokenu a následne pri jeho validácii sme určili ako stará môže táto časová pečiatka byť. Ostatné informácie sme vložili do tela tokenu ako serializovaný JSON objekt.

- Macaroons – všetky informácie sme do tokenu vložili ako pravidlá prvej strany.

NONE token sme implementovali ako samotné prihlasovacie meno používateľa. Token sa nešifruje ani nepodpisuje a nemá ani žiadnu štruktúru. Ide len o reťazec reprezentujúci prihlasovacie meno používateľa.

JWT a PASETO, aj vybrané knižnice, ktoré ich implementujú, ponúkajú viacero funkcií na podpísanie a prípadné šifrovanie tokenu. Ostatné tokeny však používajú na podpisovanie jedinú funkciu, vždy variant hešovania s kľúčom. Fernet a Branca navyše šifrujú telo tokenu. Preto sme pre objektívne porovnanie tokenov zvolili pri JWT a PASETO, čo najpodobnejšie funkcie na podpisovanie a taktiež šifrujeme telo tokenu. Šifrovacie funkcie sme tiež vybrali tak, aby boli najpodobnejšie šifrovacím funkciám v ostatných tokenoch. Kryptografické funkcie použité na podpisovanie a šifrovanie jednotlivých tokenov sú uvedené v tabuľke 4.1.

Tabuľka 4.1: Kryptografické funkcie na podpisovanie a šifrovanie tokenov

| Proces | Nepriehľadný | JWT | PASETO | Fernet | Branca | Macaroons |
|---|---|---|---|---|---|---|
| Šifrovanie | ⊘ | AES-128-CBC | AES-256-CTR | AES-128-CBC | XChaCha20 | ⊘ |
| Podpisovanie | ⊘ | HMAC-SHA256 | HMAC-SHA384 | HMAC-SHA256 | Poly1305 | HMAC-SHA256 |

## 4.4 Práca s knižnicami

Pre každý token sme pomocou danej knižnice implementovali 2 funkcie – *createToken(username)* a *verifyToken(token)*. Funkcia *createToken* vráti nový token s obsahom a formátom popísaným v podkapitole 4.3. Funkcia *verifyToken* validuje podpis tokenu. Následne dešifruje telo tokenu (ak bolo zašifrované) a validuje jeho obsah. Ako kľúč pre kryptografické funkcie sme používali konštantný náhodne vygenerovaný reťazec s potrebným počtom bitov.

Jednoduchosť práce s knižnicami implementujúcimi tokeny porovnávame na základe podpory jednoduchého generovania tokenu a štandardných validácií tela tokenu. Všetky knižnice podporujú jednoduché podpísanie a prípadné šifrovanie tokenu. Takisto všetky knižnice podporujú validáciu podpisu tokenu.

Pre JWT a PASETO nám knižnice ponúkli viacero štandardných oprávnení, ktoré sme mohli použiť pri vytváraní obsahu tokenu. Potom pri validácii tokenu stačilo uviesť

Tabuľka 4.2: Jednoduchosť práce s knižnicami

| Vlastnosť | Nepriehľadný | JWT | PASETO | Fernet | Branca | Macaroons |
|---|---|---|---|---|---|---|
| Jednoduchosť práce | ⊘ | ● | ● | ○ | ○ | ◐ |

požadované hodnoty týchto oprávnení a knižničné volanie ich validovalo. Fernet nepodporuje nijaké štandardné validovanie obsahu tokenu ani na úrovni špecifikácie, teda ani knižnica, ktorá ho implementuje, ho neponúkala. Overovanie formátu a obsahu tela tokenu sme teda implementovali sami. Branca podobne ako Fernet na úrovni špecifikácie neponúka štandardnú validáciu tela tokenu. No vybraná knižnica umožňuje validovať aspoň časovú platnosť tokenu pomocou časovej pečiatky v tokene a časového limitu zadaného ako argument pri dešifrovaní tela. Ani špecifikácia Macaroons nepodporuje žiadnu štandardnú validáciu pravidiel. Vybraná knižnica dovoľuje jednoducho pridávať pravidlá prvej strany tým, že riadi réžiu okolo postupného generovania podpisu tokenu. Neexistujú však žiadne štandardné pravidlá a preto sme ich validáciu implementovali sami.

NONE token slúži len na odmeranie času potrebného na réžiu spojenú s vykonávaním požiadaviek rozhraním, preto ho v tejto podkapitole neporovnávame.

Na základe týchto pozorovaní sme porovnali jednoduchosť práce s rôznymi tokenmi, rovnakým spôsobom ako vlastnosti porovnané v tabuľke 3.1 pomocou symbolov ●, ◐, ○ a ⊘. Výsledky sú uvedené v tabuľke 4.2.

## 4.5 Meranie rýchlosti

Pre porovnanie rýchlosti spracovania požiadavky sme implementovali jednoduchého klienta vykonávajúceho požiadavky na rozhranie. Klient sa najprv úspešne autentifikuje voči rozhraniu, ktoré mu vráti platný prístupový token. Následne klient vytvorí požiadavku na rozhranie s týmto tokenom.

Klient teda vykonáva jednu požiadavku bez tokenu, na ktorej vykonanie musí rozhranie vygenerovať nový token a jednu požiadavku s platným tokenom, na ktorej vykonanie musí rozhranie validovať token. Teda v princípe meriame čas generovania a validácie tokenu. Na meranie času vytvorenia tokenu meria klient čas od odoslania požiadavky na autentifikáciu po vrátenie tokenu rozhraním. Na meranie času validácie tokenu meria klient čas od odoslania požiadavky po vrátenie odpovede rozhraním.

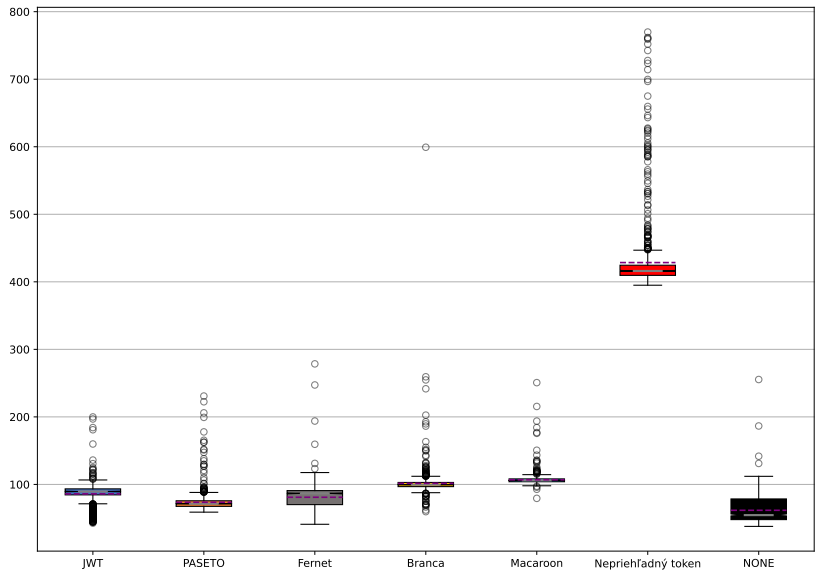
Pre presnejšie meranie meriame čas ako súčet 100 iterácií zaslania a spracovania požiadavky. Všetky namerané hodnoty sú teda súčtom 100 meraní a pre vyjadrenie času vykonania jednej požiadavky je nutné ich predeliť 100. Celý proces opakujeme 1000 krát. Klient teda vykoná spolu 100000 požiadaviek na autentifikáciu (generovanie to-

kenu) a 100000 požiadaviek s tokenom (validácia tokenu). Výsledky merania, ktoré používame na analýzu rýchlosti tokenov sú zaznamenané v súbore used_measurements[2] v elektronickej prílohe a v zdrojovom kóde na platforme GitHub.

## 4.6 Analýza nameraných výsledkov

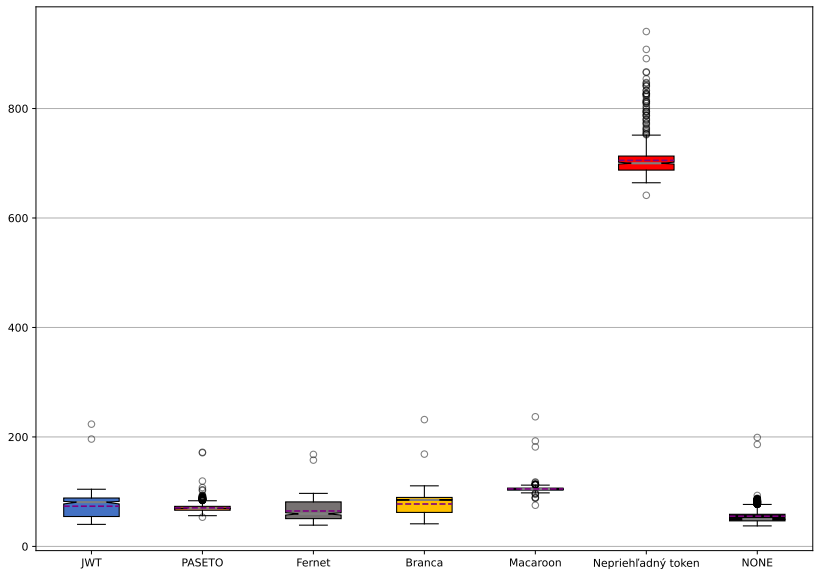
V tejto podkapitole uvedieme a popíšeme vybrané grafy, ktoré sme vygenerovali z nameraných hodnôt. Z každého grafu uvedieme dva varianty. Prvý vypočítaný z meraní generovania tokenu a druhý z meraní validácie tokenu. Keď budeme hovoriť o rýchlosti tokenu, myslíme tým rýchlosť generovania a validácie daného tokenu.

Vhodný graf pre vizualizáciu rozloženia dát je krabicový graf. Na grafe vidíme krabicu, ktorej hornú a dolnú hranu tvoria 1. a 3. kvartil. Vnútri krabice je plnou čiarou znázornený medián (2. kvartil) a čiarkovanou čiarou je znázornený aritmetický priemer. Z krabice vystupujú do dvoch strán takzvané fúzy. Hranice fúzov reprezentujú poslednú hodnotu v rámci 1,5 násobku medzikvartilového rozpätia. Medzikvartilovým rozpätím rozumieme rozdiel medzi 3. a 1. kvartilom. Všetky hodnoty, ktoré sú väčšie ako hranica horného fúzu alebo menšie ako hranica dolného fúzu sú považované za odľahlé hodnoty a sú znázornené malými kružnicami. Zárezy pri mediáne reprezentujú interval spoľahlivosti okolo mediánu. V našom prípade zobrazuje ľavá os čas v milisekundách.



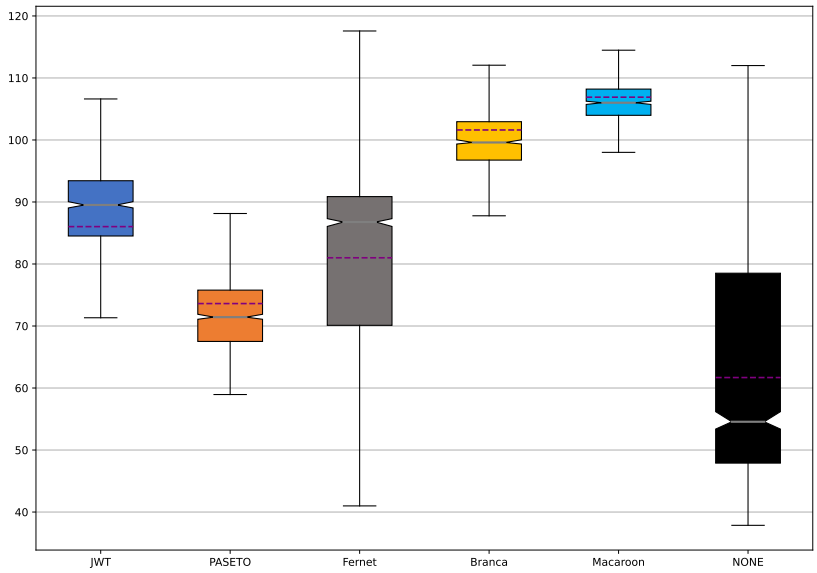

Obr. 4.1: Krabicový graf zobrazujúci všetky namerané hodnoty pri generovaní tokenu

[2]Meranie sme vykonali na počítači s procesorom Intel i7-1165G7 a s 16GB operačnej pamäte

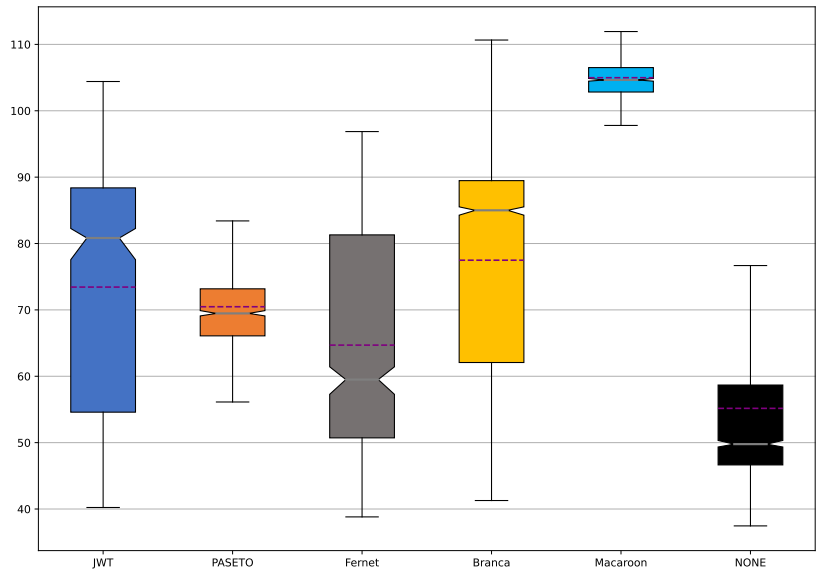



Obr. 4.2: Krabicový graf zobrazujúci všetky namerané hodnoty pri validovaní tokenu

Na grafoch 4.1 a 4.2 sú zobrazené všetky namerané hodnoty. Všimnime si množstvo odľahlých hodnôt, čo znamená, že namerané dáta obsahovali veľa extrémnych hodnôt. Výskyt extrémnych hodnôt viac popíšeme neskôr pomocou čiarových grafov a histogramov. Taktiež si všimnime, že nepriehľadný token je výrazne pomalší ako ostatné tokeny. Toto je spôsobené tým, že rozhranie si pri generovaní nového tokenu musí tento token uložiť do databázy spolu s autorizačnými údajmi a pri validácii musí token spolu s údajmi vyhľadať v databáze.



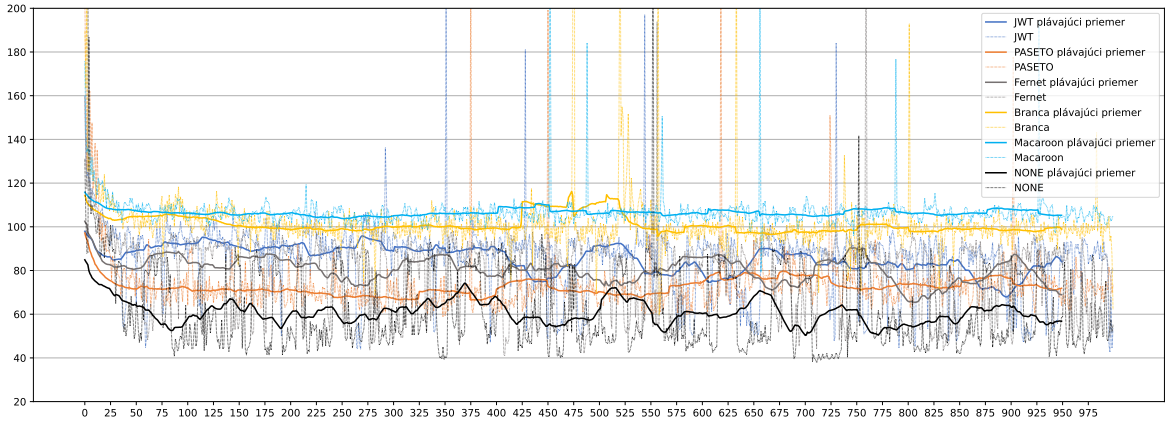

Obr. 4.3: Krabicový graf zobrazujúci namerané hodnoty bez nepriehľadného tokenu pri generovaní tokenu



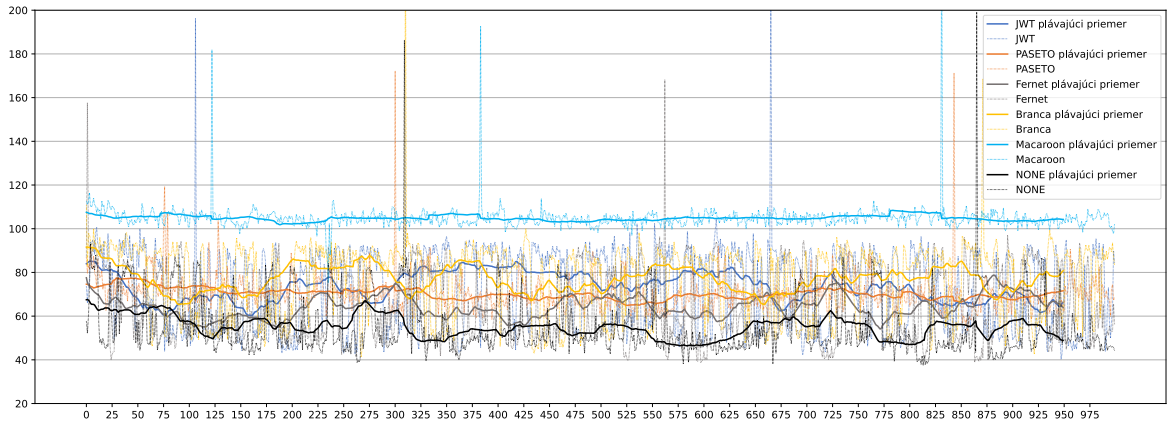

Obr. 4.4: Krabicový graf zobrazujúci namerané hodnoty bez nepriehľadného tokenu pri validovaní tokenu

Pomalosť nepriehľadného tokenu spôsobuje značnú neprehľadnosť grafov. Preto sme sa rozhodli zobraziť tokeny bez nepriehľadného tokenu. A taktiež bez odľahlých hodnôt, aby sme získali prehľadnejšie zobrazenie neextrémnych hodnôt. Tieto grafy sú zobrazené na obrázkoch 4.3 a 4.4.

Pri všetkých tokenoch vidíme zrýchlenie medzi generovaním a validáciou. Toto môže byť spôsobené tým, že veľa hodnôt používaných pri validácii aj generovaní si počítač odložil do medzipamäte (angl. cache) a pri validácií ich mal potom rýchlejšie k dispozícii. Najmenej výrazné zrýchlenie je pri Macaroons. To je pravdepodobne spôsobené tým, že Macaroons počíta viacero hodnôt pri generovaní aj validácii a taktiež má zložitú štruktúru, do ktorej sa hodnoty ukladajú. Teda manipulácia s nimi je ťažšie predvídateľná pre procesor.




Obr. 4.5: Čiarový graf zobrazujúci namerané hodnoty bez nepriehľadného tokenu pri generovaní tokenu

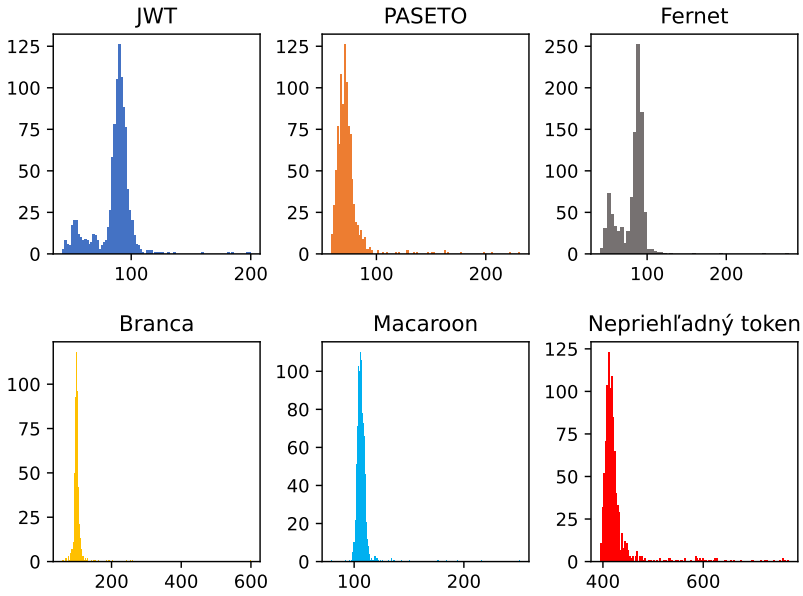



Obr. 4.6: Čiarový graf zobrazujúci namerané hodnoty bez nepriehľadného tokenu pri validovaní tokenu

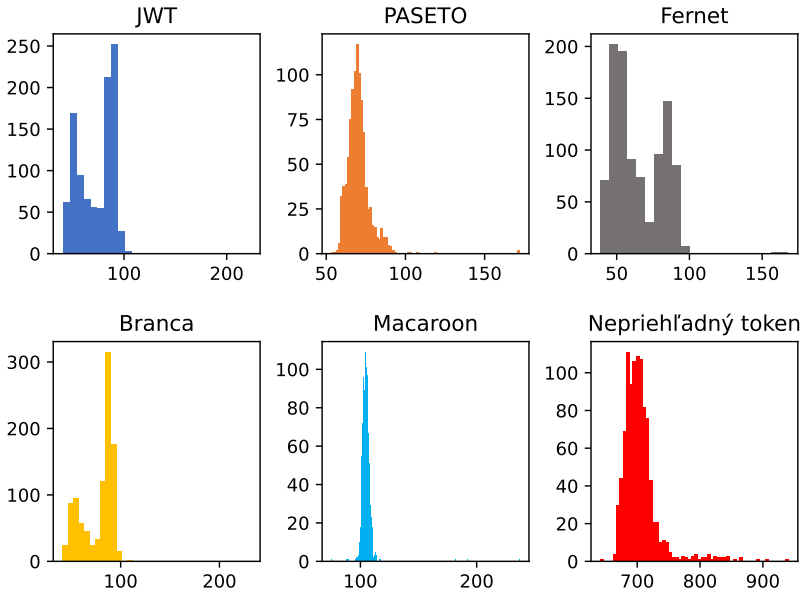
Na priamu vizualizáciu všetkých nameraných dát sme použili čiarový graf. Na ľavej osi je rýchlosť v milisekundách a na dolnej osi je poradové číslo merania. Pre vyššiu prehľadnosť grafu sme vypočítali a zobrali aj plávajúci priemer hodnôt. Plávajúci priemer je vypočítaný ako priemer 50 konsekutívnych hodnôt. Tento priemer je na grafe zobrazený plnou čiarou a všetky hodnoty sú zobrazené bodkovanými čiarami. Ako sme si všimli už pri krabicovom grafe, rýchlosť nepriehľadného tokenu je výrazne menšia ako rýchlosť ostatných tokenov. Preto sme sa rozhodli zobraziť graf bez nepriehľadného tokenu. Popísané čiarové grafy sú zobrazené na obrázkoch 4.5 a 4.6.

Vidíme, že namerané hodnoty výrazne skáču, čo dokazujú aj histogramy na obrázkoch 4.7 a 4.8. Histogramy majú na ľavej osi počet meraní a na dolnej osi rýchlosť v milisekundách. Toto správanie môžeme vysvetliť tým, že rozhranie aj klienta sme spúšťali na jednom počítači, na ktorom bežali aj iné procesy. Výkyvy v nameraných hodnotách sú teda pravdepodobne spôsobené prepínaním procesov a prideľovaním zdrojov na počítači. Až na niektoré extrémne odchýlky sú výkyvy v rámci všetkých tokenov podobné. Najmä na čiarovom grafe zobrazujúcom generovanie tokenu vidíme, že namerané hodnoty v prvých približne 25 meraniach sú vyššie ako v ostatných meraniach a postupne klesajú. Toto sa dá vysvetliť optimalizáciou často používaného kódu počas behu programu (angl. just in time compilation). Túto interpretáciu potvrdzuje aj to, že pri validácii tokenu sa tento jav nevyskytuje. Validácia sa totiž meria až po zmeraní generovania tokenu a v rámci jedného spustenia kódu rozhrania aj klienta. Táto optimalizácia sa teda už stihla vykonať.

Pre číselné porovnanie rýchlosti tokenov sme z čiarových grafov 4.5 a 4.6 odstránili bodkované čiary a nechali len plné čiary, reprezentujúce plávajúce priemery. Od hodnôt tokenov sme odpočítali hodnoty namerané s NONE tokenom, aby sme porovnávali čisto rýchlosť generovania a validácie tokenov bez réžie okolo spracovania požiadaviek. Tieto grafy sú zobrazené na obrázkoch 4.9 a 4.10.

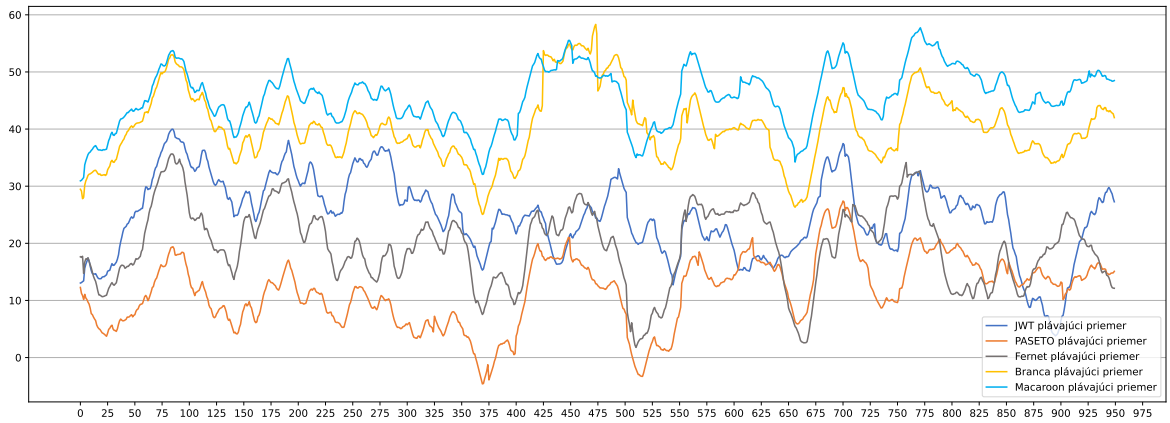



Obr. 4.7: Histogram zobrazujúci všetky namerané hodnoty pri generovaní tokenu

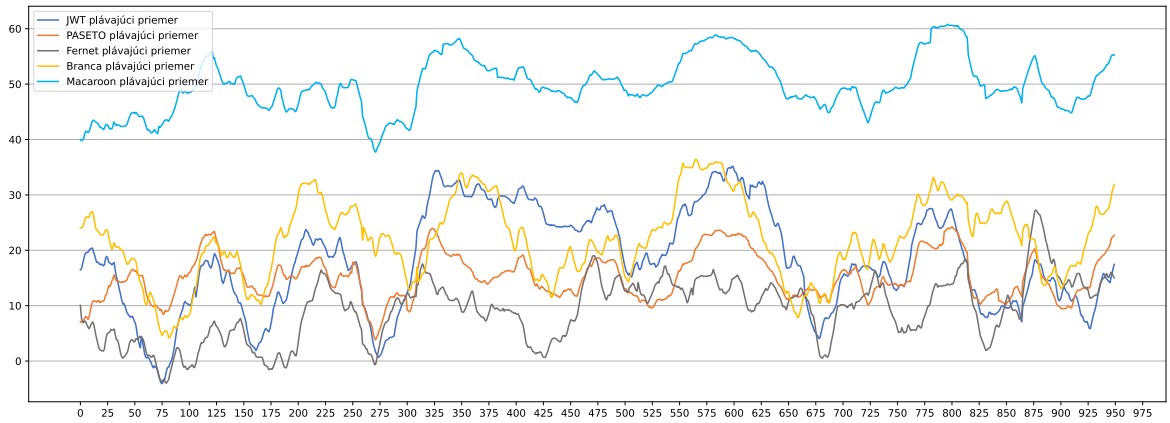



Obr. 4.8: Histogram zobrazujúci všetky namerané hodnoty pri validovaní tokenu

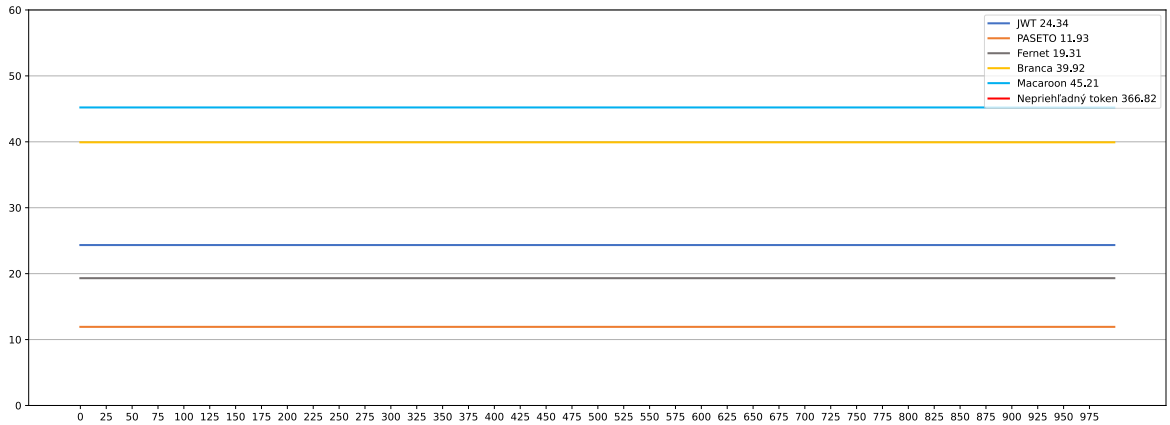
Aj hodnoty plávajúcich priemerov nie sú úplne rovnomerné a nedá sa z nich priamo porovnať rýchlosť tokenov. Preto sme sa rozhodli porovnať rýchlosť na základe aritmetického priemeru všetkých nameraných hodnôt (tiež po odpočítaní NONE tokenu). Toto porovnanie je zobrazené v grafoch 4.11 a 4.12. Graf má odrezanú ľavú os tak, aby nezobrazoval nepriehľadný token. V takomto prípade sú prehľadnejšie vidieť rozdiely medzi ostatnými tokenmi. V legendách je zobrazený aritmetický priemer všetkých nameraných hodnôt. V legende je uvedený aj aritmetický priemer rýchlosti nepriehľadného tokenu.



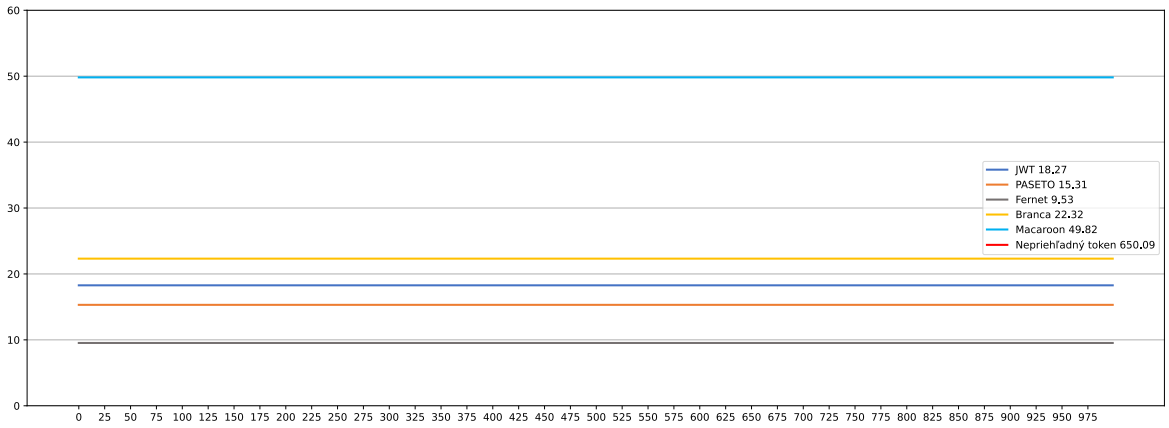

Obr. 4.9: Čiarový graf zobrazujúci namerané hodnoty s odpočítaným NONE tokenom bez nepriehľadného tokenu pri generovaní tokenu




Obr. 4.10: Čiarový graf zobrazujúci namerané hodnoty s odpočítaným NONE tokenom bez nepriehľadného tokenu pri validovaní tokenu




Obr. 4.11: Čiarový graf zobrazujúci aritmetický priemer nameraných hodnôt s odpočítaným NONE tokenom bez nepriehľadného tokenu pri generovaní tokenu




Obr. 4.12: Čiarový graf zobrazujúci aritmetický priemer nameraných hodnôt s odpočítaným NONE tokenom bez nepriehľadného tokenu pri validovaní tokenu

Ako sme spomenuli už vyššie, nepriehľadný token je pre využívanie databázy výrazne pomalší ako ostatné tokeny. Konkrétne je v priemere v prípade generovania 8-30 krát pomalší a v prípade validácie 13-68 krát pomalší.

Prekvapivým výsledkom je, že Macaroons je okrem nepriehľadného tokenu najpomalší pri generovaní aj validácii, pretože okrem nepriehľadného tokenu ide o jediný token, ktorého telo sa nešifruje. Tento výsledok je možné vysvetliť tým, že Macaroons je najkomplexnejší token, formátom aj procesom generovania. Pri generovaní sa aplikuje podpisová funkcia raz pri vytvorení nového tokenu a následne toľkokrát koľko pravidiel do tokenu pridávame, v našom prípade štyrikrát. Podobne pri validácii tokenu sa musí vypočítať podpis tokenu postupnou aplikáciou podpisovej funkcie na každé pravidlo v tele tokenu. Macaroons bol vytvorený pre komplexné systémy pre jeho flexibilitu spočívajúcu v možnosti pridávať pravidlá ľubovoľnou entitou a elegantného riešenia zapojenia tretích strán do autorizácie pomocou pravidiel tretích strán. Preto je pre jednoduchý systém, ako je nami implementované rozhranie, Macaroons nevhodný.

Zaujímavým pozorovaním je výrazné zrýchlenie validácie nepriehľadného tokenu a Branca tokenu oproti jeho generovaniu. Vyššie sme odôvodnili prečo pri všetkých tokenoch vidíme zrýchlenie validácie oproti generovaniu. V prípade týchto tokenov je ale zrýchlenie najvýraznejšie. V oboch prípadoch je takmer dvojnásobné. Rozhranie využíva jednoduchú súborovú databázu sqlite. Pri generovaní nepriehľadného tokenu sa teda pravdepodobne databáza načíta do operačnej pamäti z disku a ďalej pri validácii sa už číta iba z operačnej pamäte, ktorá je výrazne rýchlejšia.

Pri Branca tokene predpokladáme, že ide o rozdiel rýchlosti šifrovacej funkcie pri šifrovaní a dešifrovaní. Táto funkcie je totiž odlišná od funkcií použitých v ostatných tokenoch.

Celkovo najrýchlejší je pri generovaní PASETO, konkrétne 1,6-3,8 krát rýchlejší. Pri validácii je najrýchlejší Fernet, konkrétne 1,6-5,2 krát rýchlejší. V oboch prípadoch

porovnávame najrýchlejší token s ostatnými tokenmi okrem nepriehľadného tokenu. Uvedomme si však, že najmä v prípade validácie sú rozdiely medzi JWT, PASETO, Fernet a Branca takmer zanedbateľné. Rozdiel ich rýchlostí je najviac 12,78 ms pri súčte 100 meraní. Teda 0,13 ms na jednu požiadavku.

# Záver

V práci sme sa venovali problematike autentifikácie a autorizácie so zameraním na tokeny. Práca sa dá pomyselne rozdeliť na dva logické celky. V prvej časti sme sa venovali teoretickým poznatkom, ktoré sú potrebné pre pochopenie problematiky.

Konkrétne v kapitole 1 sme uviedli rôzne známe prístupy k architektúre schémy zabezpečenia. Jednotlivé prístupy sme stručne charakterizovali a uviedli ich hlavné výhody a nevýhody. Okrem toho sme, pre prehľad v problematike tokenov, popísali známe typy a formáty tokenov. Ďalej sme sa venovali iba schéme zabezpečenia využívajúcej tokeny.

V kapitole 2 sme predstavili konkrétne tokeny, ktoré sme ďalej v práci porovnávali. Venovali sme sa najmä charakteristikám a parametrom tokenov, ktoré sú nevyhnutné pre analýzu ich vlastností a porovnávanie s inými tokenmi.

V druhej časti práce sme sa venovali analýze, teoretickému a potom aj praktickému porovnaniu tokenov. V kapitole 3 sme porovnali tokeny z pohľadu bezpečnosti, flexibility a popularity. Výsledkom tohto porovnania je prehľadová tabuľka 3.1. V kapitole 4 sme navrhli a implementovali jednoduché rozhranie, na ktorom sme demonštrovali použitie jednotlivých tokenov v schéme zabezpečenia. Taktiež sme implementovali jednoduchého klienta, ktorý vykonával požiadavky na rozhranie a meral čas ich vykonania. Merali sme dva typy požiadaviek. Pri prvom type muselo rozhranie vygenerovať token a pri druhom validovať token. Namerané hodnoty a ich interpretácia pomocou rôznych grafov je v sekcii 4.6. Z meraní sme zistili, že najrýchlejšie tokeny sú Fernet a PASETO, no rozdiel v rýchlostiach tokenov JWT, PASETO, Branca a Fernet je takmer zanedbateľný. Pri validácii tokenu je tento rozdiel najviac 0,13 ms na jednu požiadavku. Naopak výrazne najpomalší bol nepriehľadný token z dôvodu potreby použitia databázy pri generovaní aj validácii tokenu. Okrem nepriehľadného tokenu bol najpomalší Macaroons. Ide však o veľmi flexibilný token s komplexnou štruktúrou, preto nie je vhodný na použitie v jednoduchom rozhraní, ktoré sme implementovali.

Hlavným cieľom práce bolo porovnať rôzne tokeny. Ako výsledok tohto porovnania uvádzame tabuľku 4.3, ktorá je spojením tabuliek 3.1, 4.2 a nameraných priemerných rýchlostí tokenov. Podrobnejšia interpretácia uvedených hodnôt je v kapitolách 3 a 4.

Tabuľka 4.3: Porovnanie tokenov

| Vlastnosť | Nepriehľadný | JWT | PASETO | Fernet | Branca | Macaroons | Biscuits |
|---|---|---|---|---|---|---|---|
| Počet kryptografických funkcií | 1 | 30 | 6 | 1 | 1 | 1 | 1 |
| Určenie podpisového alg. z tokenu | ⊘ | ● | ● | ○ | ○ | ○ | ○ |
| Náchylnosť na útok pomýlením algortimu | ⊘ | ● | ◐ | ○ | ○ | ○ | ○ |
| Riešenie problému odvolania | ● | ○ | ○ | ○ | ○ | ◐ | ◐ |
| Náchylnosť na útok opakovaním | ⊘ | ● | ● | ● | ● | ◐ | ◐ |
| Ochrana dôvernosti | ⊘ | ◐ | ◐ | ● | ● | ○ | ○ |
| Overenie autenticity a integrity hocikým | ○ | ◐ | ● | ● | ● | ● | ● |
| zoslabenie tokenu hocikým | ⊘ | ⊘ | ⊘ | ⊘ | ⊘ | ● | ● |
| Autorizačná schéma v tokene | ⊘ | ◐ | ◐ | ◐ | ◐ | ● | ● |
| Bezstavová validácia | ⊘ | ● | ● | ● | ● | ● | ● |
| Štandardná validácia | ⊘ | ● | ● | ○ | ○ | ◐ | ● |
| Popularita | ⊘ | ● | ◐ | ◐ | ○ | ◐ | ○ |
| Jednoduchosť práce | ⊘ | ● | ● | ○ | ○ | ◐ | ⊘ |
| Generovanie priemer [ms] | 366,82 | 24,34 | 11,93 | 19,31 | 39,92 | 45,21 | ⊘ |
| Validácia priemer [ms] | 650,09 | 18,27 | 15,31 | 9,53 | 22,31 | 49,82 | ⊘ |

Pri porovnaní rýchlosti tokenov na testovacom API si však uvedomujeme, že porovnávame iba konkrétne implementácie daných tokenov. Implementácie sme vybrali na základe popularity a udržiavanosti knižníc, ktoré ich sprostredkúvajú. Detailne analyzovať konkrétne implementácie tokenov presahuje rozsah tejto práce. Jednou z možností ako nadviazať na túto prácu je preto porovnanie viacerých implementácií každého tokenu a analýza a porovnanie týchto implementácií.

Okrem toho je v budúcnosti možné nadviazať na našu prácu buď porovnaním ďalších tokenov, ktoré vzniknú v budúcnosti alebo porovnaním tokenov, ktoré sme v práci nespomenuli (napríklad token na základe certifikátov verejných kľúčov a Crypto auth token (CAT), oba vytvorené [39] a používané spoločnosťou Meta). Ďalej je možné nadviazať na našu prácu rozšírením rozhrania o ďalšie funkcie, ktoré by boli zaujímavé z pohľadu porovnania tokenov, napríklad pridať tretie strany do procesu autorizácie požiadavky.

# Literatúra


[1] R. Gunawan A. Rahmatulloh and F. M. S. Nursuwars. Performance comparison of signed algorithms on json web token. https://iopscience.iop.org/article/10.1088/1757-899X/550/1/012023/pdf. [Citované 19.4.2023].

[2] S. Arciszewski. AEAD XChaCha20 Poly1305. https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-xchacha-03. [Citované 23.3.2023].

[3] S. Arciszewski. PASETO: Platform-agnostic security tokens. https://paseto.io/rfc/. [Citované 20.3.2023].

[4] AWS. Signing AWS API requests. https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_aws-signing.html. [Citované 2.5.2023].

[5] Nanne Baars. PASETO Java knižnica. https://github.com/nbaars/paseto4j. [Citované 19.4.2023].

[6] Richard Barnes. Use Cases and Requirements for JSON Object Signing and Encryption (JOSE). RFC 7165, Apríl 2014.

[7] Arnar Birgisson, Joe Gibbs Politz, Úlfar Erlingsson, Ankur Taly, Michael Vrable, and Mark Lentczner. Macaroons: Cookies with contextual caveats for decentralized authorization in the cloud. In *Network and Distributed System Security Symposium*, 2014.

[8] Clever Cloud. Bisquits repository. https://github.com/biscuit-auth/biscuit. [Citované 8.4.2023].

[9] Geoffroy Couprie. Biscuit, the foundation for your authorization systems. https://www.clever-cloud.com/blog/engineering/2021/04/12/introduction-to-biscuit/. [Citované 8.4.2023].

[10] Geoffroy Couprie. Biscuits Java knižnica. https://github.com/biscuit-auth/biscuit-java. [Citované 19.4.2023].

[11] Curity. The phantom token approach. https://curity.io/resources/learn/phantom-token-pattern/. [Citované 8.4.2023].

[12] Curity. The split token approach. https://curity.io/resources/learn/split-token-pattern/. [Citované 8.4.2023].

[13] Ajitomi Daisuke. PASETO Python knižnica. https://github.com/dajiaji/pyseto. [Citované 19.4.2023].

[14] Zoran Davidović. Fernet JavaScript knižnica. https://github.com/zoran-php/fernet-nodejs. [Citované 19.4.2023].

[15] Robert Escriva. libmacaroons. https://github.com/rescrv/libmacaroons. [Citované 8.4.2023].

[16] Robert Escriva. Macaroons Python knižnica. https://github.com/rescrv/libmacaroons. [Citované 19.4.2023].

[17] Amit Eyal. C++ Fernet implementation. https://github.com/IamAmitE/FernetCpp/. [Citované 23.3.2023].

[18] Fallible. We reverse engineered 16k apps, here's what we found. https://hackernoon.com/we-reverse-engineered-16k-apps-heres-what-we-found-51bdf3b456bb#.io6e11q6n. [Citované 23.2.2023].

[19] Sheila Frankel, K. Robert Glenn, and Scott G. Kelly. The AES-CBC Cipher Algorithm and Its Use with IPsec. RFC 3602, September 2003.

[20] Sheila Frankel and Scott G. Kelly. Using HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512 with IPsec. RFC 4868, May 2007.

[21] Harold Giménez. Fernet legacy specification. https://github.com/heroku/legacy-fernet. [Citované 23.3.2023].

[22] Harold Giménez. Fernet specification. https://github.com/fernet/spec/blob/f16a35d3cfd8cdb2d8c7f7d10ce6c4d6058b19d2/Spec.md. [Citované 23.3.2023].

[23] Google. Belay research project. https://sites.google.com/site/belayresearchproject/home. [Citované 8.4.2023].

[24] Google. Protocol buffers documentation. https://protobuf.dev/overview/. [Citované 8.4.2023].

[25] Weili Han, Zhigong Li, Minyue Ni, Guofei Gu, and Wenyuan Xu. Shadow attacks based on password reuses: A quantitative empirical analysis. *IEEE Transactions on Dependable and Secure Computing*, 15(2):309–320, 2018.

[26] Dick Hardt. The OAuth 2.0 Authorization Framework. RFC 6749, 2012.

[27] Les Hazlewood. JWT Java knižnica. `https://github.com/jwtk/jjwt`. [Citované 19.4.2023].

[28] Kejing He, Xiancheng Xu, and Qiang Yue. A secure, lossless, and compressed base62 encoding. In *2008 11th IEEE Singapore International Conference on Communication Systems*, pages 761–765, 2008.

[29] Michael Jones. JSON Web Algorithms (JWA). RFC 7518, Máj 2015.

[30] Michael Jones. JSON Web Key (JWK). RFC 7517, Máj 2015.

[31] Michael Jones, John Bradley, and Nat Sakimura. JSON Web Signature (JWS). RFC 7515, Máj 2015.

[32] Michael Jones, John Bradley, and Nat Sakimura. JSON Web Token (JWT). RFC 7519, Máj 2015.

[33] Michael Jones and Joe Hildebrand. JSON Web Encryption (JWE). RFC 7516, Máj 2015.

[34] Simon Josefsson. The Base16, Base32, and Base64 Data Encodings. RFC 4648, Október 2006.

[35] Simon Josefsson and Ilari Liusvaara. Edwards-Curve Digital Signature Algorithm (EdDSA). RFC 8032, Január 2017.

[36] Martin W. Kirst. Macaroons Java knižnica. `https://github.com/nitram509/jmacaroons`. [Citované 19.4.2023].

[37] Dr. Hugo Krawczyk, Mihir Bellare, and Ran Canetti. HMAC: Keyed-Hashing for Message Authentication. RFC 2104, Február 1997.

[38] Butler Lampson, Martín Abadi, Michael Burrows, and Ted Wobber. Authentication in distributed systems: Theory and practice. volume 10, pages 165–182, Október 1991.

[39] Kevin Lewi, Callen Rain, Stephen A. Weis, Yueting Lee, Haozhi Xiong, and Benjamin Yang. Scaling backend authentication at facebook. *IACR Cryptol. ePrint Arch.*, 2018:413, 2018.

[40] Ninghui Li and John C. Mitchell. Datalog with constraints: A foundation for trust management languages. In *Proceedings of the Fifth International Symposium on Practical Aspects of Declarative Languages*, Január 2003.

[41] Luis Lizama, Leonardo Arrieta, Flor Mendoza, Luis Servín, and Eric Simancas-Acevedo. Public hash signature for mobile network devices. *Ingeniería Investigación y Tecnología*, 20:1–10, Apríl 2019.

[42] Rodney Lorrimar. Haskell Fernet implementation. `https://github.com/IamAmitE/FernetCpp/`. [Citované 23.3.2023].

[43] Carlos Macasaet. Fernet Java knižnica. `https://github.com/l0s/fernet-java8`. [Citované 19.4.2023].

[44] Arvind Narayanan and Vitaly Shmatikov. Fast dictionary attacks on passwords using time-space tradeoff. In *Proceedings of the 12th ACM Conference on Computer and Communications Security*, CCS '05, page 364–372, New York, NY, USA, 2005. Association for Computing Machinery.

[45] Okta. Access token: Definition, architecture, usage and more. `https://www.okta.com/identity-101/access-token/`. [Citované 8.4.2023].

[46] Okta. Critical vulnerabilities in json web token libraries. `https://auth0.com/blog/critical-vulnerabilities-in-json-web-token-libraries/#Meet-the--None--Algorithm`. [Citované 17.4.2023].

[47] José Padilla. JWT Python knižnica. `https://github.com/jpadilla/pyjwt/`. [Citované 19.4.2023].

[48] Roger Peppe. Macaroons JavaScript knižnica. `https://github.com/go-macaroon/js-macaroon`. [Citované 19.4.2023].

[49] Thomas Pornin and Julien P. Stern. Digital signatures do not guarantee exclusive ownership. `http://www.bolet.org/~pornin/2005-acns-pornin+stern.pdf`. [Citované 20.3.2023].

[50] Agathoklis Prodromou. TLS security 6: Examples of TLS vulnerabilities and attacks. `https://www.acunetix.com/blog/articles/tls-vulnerabilities-attacks-final-part/`. [Citované 27.2. 2023].

[51] PYCA. Fernet Python knižnica. `https://github.com/pyca/cryptography/blob/main/src/cryptography/fernet.py`. [Citované 19.4.2023].

[52] PYCA. Python cryptography library. `https://github.com/pyca/cryptography`. [Citované 23.3.2023].

[53] Redis. Redis. `https://redis.io/`. [Citované 8.4.2023].

[54] N. Sakimura, NRI, J. Bradley, Ping Identity, Microsoft, M. Jones, B. de Medeiros, Google, and C. Mortimore. OpenID Connect Core 1.0 incorporating errata set 1. `https://openid.net/specs/openid-connect-core-1_0.html#IDToken`. [Citované 23.2.2023].

[55] Filip Skokan. JWT JavaScript knižnica. `https://github.com/panva/jose`. [Citované 19.4.2023].

[56] Filip Skokan. PASETO JavaScript knižnica. `https://github.com/panva/paseto`. [Citované 19.4.2023].

[57] P.I.E. Security Team. PASETO algorithm lucidity. `https://github.com/paseto-standard/paseto-spec/blob/master/docs/02-Implementation-Guide/03-Algorithm-Lucidity.md`. [Citované 19.4.2023].

[58] P.I.E. Security Team. PASETO specification. `https://github.com/paseto-standard/paseto-spec`. [Citované 20.3.2023].

[59] Mika Tuupola. Branca JavaScript knižnica. `https://github.com/tuupola/branca-js`. [Citované 19.4.2023].

[60] Mika Tuupola. Branca Python knižnica. `https://github.com/tuupola/pybranca`. [Citované 19.4.2023].

[61] Mika Tuupola. Branca specification. `https://github.com/tuupola/branca-spec`. [Citované 23.3.2023].

[62] Paul Wagenseil. Digital epochalypse could bring world to grinding halt. `https://www.tomsguide.com/us/2038-bug-bh2017,news-25551.html`. [Citované 19.4.2023].

[63] Bjoern Weidlich. Branca Java knižnica. `https://github.com/bjoernw/jbranca`. [Citované 19.4.2023].

# Príloha A: obsah elektronickej prílohy

V elektronickej prílohe priloženej k práci sa nachádza zdrojový kód programu a súbory s výsledkami experimentov. Zdrojový kód je zverejnený aj na stránke `https://github.com/jitka1997/bachelor-thesis/tree/main/simpleAPI`.

Obsah tejto prílohy je testovacie API a klient, obe navrhnuté v kapitole 4. Okrem toho príloha obsahuje skript na generovanie grafov a HTML reportu z nameraných hodnôt.

Konkrétne použitie testovacieho API a následného generovanie reportu je bližšie popísané v README.md súbore v prílohe.