Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

FIIT-16768-120762

Peter Brenkus

Zvýšenie efektivity OCSP komunikácie

Bakalárska práca

Študijný program:	Informatika
Študijný odbor:	9.2.1 Informatika
Miesto vypracovania:	Ústav počítačového inžinierstva a aplikovanej informatiky
Vedúci práce:	Ing. Norbert Varga

máj 2025

Slovenská technická univerzita v Bratislave
Ústav počítačového inžinierstva a aplikovanej informatiky

Fakulta informatiky a informačných technológií
Akademický rok: 2024/2025
Evidenčné číslo: FIIT-16768-120762

ZADANIE BAKALÁRSKEJ PRÁCE

Študent: **Peter Brenkus**
ID študenta: 120762
Študijný program: informatika
Študijný odbor: informatika
Vedúci práce: Ing. Norbert Varga
Vedúci pracoviska: Ing. Katarína Jelemenská, PhD.

Názov práce: **Zvýšenie efektivity OCSP komunikácie**

Jazyk, v ktorom sa práca vypracuje: slovenský jazyk

Špecifikácia zadania:

Protokol OCSP, ktorý sa dlhodobo používa pri efektívnom zisťovaní revokačných údajov k certifikátom verejného kľúča, má svoje pevné miesto v oblasti PKI. Tento protokol môže pracovať rôznymi spôsobmi a rovnako existujú rôzne pristupy k zvyšovaniu jeho efektivity. Analyzujte protokol OCSP, ako aj prístupy k zvyšovaniu jeho efektivity, vrátane metódy OCSP Stapling a rôznych metód vyvažovania zátaže. Navrhnite vlastné riešenie, ktoré zefektívni OCSP komunikáciu, implementujte ho a porovnajte s inými existujúcimi riešeniami. Funkčnosť a prínosy vlastného riešenia overte a tento proces, ako aj jeho výsledky, zdokumentujte.

Rozsah práce: 40

Termín odovzdania bakalárskej práce: 12. 05. 2025
Dátum schválenia zadania bakalárskej práce: 15. 04. 2025
Zadanie bakalárskej práce schválil: doc. Ing. Ján Lang, PhD. – garant študijného programu

Čestne vyhlasujem, že som túto prácu vypracoval samostatne, na základe konzultácií a s použitím uvedenej literatúry.

V Bratislave, 12.5.2025

Peter Brenkus

Poďakovanie

Rád by som poďakoval svojej rodine za podporu, trpezlivosť a povzbudenie počas celej doby štúdia. Bez ich pomoci a porozumenia by táto práca nevznikla.
Rád by som poďakoval aj vedúcemu tejto práce, Norbertovi Vargovi, za vynikajúce vedenie, odborné znalosti a ochotu podeliť sa o cenné rady počas celého procesu tvorby tejto práce.

Anotácia

Slovenská technická univerzita v Bratislave
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

Študijný program:	Informatika
Autor:	Peter Brenkus
Bakalárska práca:	Zvýšenie efektivity OCSP komunikácie
Vedúci bakalárskej práce:	Ing. Norbert Varga
máj 2025	

Cieľom tejto práce je zlepšenie efektivity OCSP komunikácie, ktorá sa používa na zistenie stavu odvolania digitálnych certifikátov v reálnom čase.

Práca obsahuje analýzu problematiky, ktorej súčasťou je podrobný opis jeho architektúry, nedostatkov a ich existujúcich riešení ako OCSP Stapling, OCSP Must-Staple, OCSP Multi-Staple, CCSP a TinyOCSP. Okrem toho analýza rozoberá aj relevantné časti HTTP protokolu a manažérsky pohľad na problematiku. Na základe tejto analýzy boli sformulované požiadavky a návrh riešenia.

Súčasťou práce je návrh vlastného riešenia, ktoré implementuje HTTP server s cache pre POST požiadavky pre účely OCSP, čím znižuje záťaž OCSP serverov a zlepšuje efektivitu celého systému.

Implementované riešenie bolo podrobené testovaniu, v ktorom sa merala rýchlosť odozvy a priepustnosť systému. Výsledky testovania preukázali významné zlepšenie výkonu, konkrétne zníženie času odozvy pre POST požiadavky o 46.5% a zvýšenie priepustnosti systému až o 84.1% v porovnaní s konfiguráciou bez cache. Tieto výsledky potvrdzujú efektívnosť navrhnutého riešenia pri optimalizácii OCSP komunikácie.

Annotation

Slovak University of Technology in Bratislava
FACULTY OF INFORMATICS AND INFORMATION TECHNOLOGIES

Degree course: Informatics

Author: Peter Brenkus
Bachelor's Thesis: Increasing the efficiency of OCSP communication
Supervisor: Norbert Varga
2025, May

The aim of this thesis is to improve the efficiency of the OCSP protocol, which is used for real-time validation of the revocation status of digital certificates.

The thesis includes an analysis of the problem, comprising of a detailed description of the protocol's architecture, its shortcomings, and existing solutions such as OCSP Stapling, OCSP Must-Staple, OCSP Multi-Stapling, CCSP and TinyOCSP. The analysis also includes the description of relevant parts of the HTTP protocol and a managerial perspective on the issue. Based on this analysis, requirements were formulated, followed by the proposal of a solution.

The thesis also presents the design of a custom solution that implements an HTTP server for caching POST requests for OCSP purposes, thereby reducing the load on OCSP responders and improving the overall efficiency of the system.

The implemented solution was thoroughly tested, measuring response time and system throughput. Testing results demonstrated significant performance improvements, specifically a 46.5% reduction in response time for POST requests and an 84.1% increase in system throughput compared to configurations without caching. These results confirm the effectiveness of the proposed solution in optimizing OCSP communication.

Obsah

Zoznam obrázkov

Tabuľka 1: Použité skratky

Skratka	**Vysvetlenie**
OCSP	Online Certificate Status Protocol
CCSP	Compressed Certificate Status Protocol
CRL	Certificate Revocation List
CA	Certificate Authority
DN	Distinguished Name
PKI	Public Key Infrastructure
TLS	Transport Layer Security
SSL	Secure Sockets Layer
ASN.1	Abstract Syntax Notation One
DER	Distinguished Encoding Rules
TLV	Tag Length Value
CBOR	Concise Binary Object Representation
AKID	Authority Key Identifier
IoT	Internet of Things
IIoT	Industrial Internet of Things
DTLS	Datagram Transport Layer Security
HTTP	Hypertext Transfer Protocol
URL	Uniform Resource Locator
CDN	Content Delivery Network

1 Úvod

S rastúcim využívaním digitálnych technológií v každodennom živote sa zvyšujú aj nároky na bezpečnosť online komunikácie. Jedným z prvkov, ktoré zaisťujú túto bezpečnosť, sú digitálne certifikáty vydávané a spravované certifikačnými autoritami. Certifikačné autority overujú identitu subjektov a vydávajú im certifikáty, ktoré sú nevyhnutné pre bezpečné šifrované spojenia. Na overovanie platnosti týchto certifikátov v reálnom čase sa používa viacero metód, jednou z nich je protokol OCSP [1].

Vyvinutý ako alternatíva k CRL, OCSP poskytuje rýchlejší a efektívnejší spôsob overovania certifikátov. Klient, napríklad webový prehliadač, môže poslať požiadavku na OCSP server a overiť konkrétny certifikát bez toho, aby musel sťahovať veľký zoznam identifikátorov certifikátov, ako je tomu v prípade CRL [1].

OCSP hrá dôležitú úlohu v bezpečnostných mechanizmoch na internete. Z pohľadu efektivity a rýchlosti komunikácie však stále existuje priestor na jeho zlepšenie. Cieľom tejto práce je analyzovať súčasný stav protokolu OCSP, identifikovať jeho nedostatky a navrhnúť efektívnejšie poskytnutie služby overenia platnosti digitálnych certifikátov.

Kapitola 2 obsahuje analýzu problematiky. Konkrétne úvod do oblasti PKI, digitálnych certifikátov, kontroly ich revokácie a komplexnú analýzu protokolu OCSP. Kapitola 3 obsahuje špecifikáciu funkčných a nie-funkčných požiadaviek. Kapitola 4 obsahuje návrh riešenia. V piatej kapitole je popísaná implementácia riešenia spolu so zmenami oproti návrhu. Kapitola 6 sa venuje overeniu implementovaného riešenia a posledná kapitola obsahuje zhodnotenie práce.

2 Analýza problematiky

V tejto časti najprv uvedieme relevantné teoretické oblasti, potom analyzujeme ako presne protokol OCSP funguje a aké sú jeho slabé stránky. Ďalej sa pozrieme aké vylepšenia už existujú a zameriame sa na oblasti možného ďalšieho vylepšenia.

2.1 PKI

PKI je systém, ktorý umožňuje bezpečnú výmenu informácií na internete prostredníctvom kryptografických kľúčov [2]. Hlavným cieľom PKI je zaručiť bezpečnosť údajov tým, že poskytuje mechanizmy na overenie identity a šifrovanie dát [3]. Tento systém vychádza z asymetrickej kryptografie, ktorá používa dva druhy kľúčov – verejný a súkromný. Verejný kľúč je zdieľaný verejne a slúži na šifrovanie informácií alebo na overenie podpisu [2]. Súkromný kľúč je známy len majiteľovi a slúži na dešifrovanie informácií alebo vytváranie digitálnych podpisov [3].

Digitálne certifikáty

Digitálny certifikát je kľúčovým komponentom PKI [4]. Je to dátová štruktúra, ktorá spája verejný kľúč s identitou osoby alebo organizácie. Certifikáty vydáva dôveryhodná tretia strana, nazývaná tiež ako certifikačná autorita (CA). Existujú rôzne štandardy, ktoré definujú štruktúru digitálnych certifikátov, najpoužívanejší štandard je X.509 [4]. Tento štandard definuje 3 povinné polia pomocou ASN.1 [5].

```
Certificate  ::=  SEQUENCE  {
    tbsCertificate       TBSCertificate,
    signatureAlgorithm   AlgorithmIdentifier,
    signatureValue       BIT STRING
}
```

Obr. 1: Štruktúra X.509 certifikátu

Každé pole má stanovenú presnú syntax a je zakódované do sekvencie bajtov. Na kódovanie sa používa ASN.1 DER, čo je systém založený na kódovaní štýlom TLV. Znamená to, že každý prvok v štruktúre certifikátu je zakódovaný samostatne pomocou troch zložiek [6]:

- **Tag** identifikuje typ dát (napr. celé číslo, reťazec, sekvencia)
- **Length** udáva dĺžku daného prvku v bajtoch
- **Value** obsahuje samotné údaje daného prvku.

Prvé povinné pole obsahuje údaje o CA, ktorá certifikát vydala a subjekte, ktorému bol certifikát vydaný. Hlavnými údajmi sú názvy CA a subjektu, verejný kľúč subjektu, perióda platnosti certifikátu, číslo verzie cetifikátu a sériové číslo certifikátu. Môžu tu však byť aj ďalšie nepovinné polia [5].

```
TBSCertificate  ::=  SEQUENCE  {
    version          [0]  EXPLICIT Version DEFAULT v1,
    serialNumber          CertificateSerialNumber,
    signature             AlgorithmIdentifier,
    issuer                Name,
    validity              Validity,
    subject               Name,
    subjectPublicKeyInfo  SubjectPublicKeyInfo,
    issuerUniqueID   [1]  IMPLICIT UniqueIdentifier OPTIONAL
       ,
    subjectUniqueID  [2]  IMPLICIT UniqueIdentifier OPTIONAL
       ,
    extensions       [3]  EXPLICIT Extensions OPTIONAL
}
```

Obr. 2: Štruktúra poľa TBSCertificate

X.509 certifikáty môžu byť v troch rôznych verziách: v1, v2, a v3. Verzia v1 bola pôvodnou špecifikáciou, avšak bola nedostatočná pre moderné potreby, najmä kvôli absencii rozšírení. Verzia v2 pridala podporu pre identifikátory vydavateľa a subjektu, čo pomáha rozlíšiť napríklad certifikáty s rovnakým menom subjektu, ale vydané rôznymi certifikačnými autoritami. Verzia v3 je najnovšia a pridáva možnosť rozšírení, ktoré umožňujú špecifikovať ďalšie atribúty, napríklad obmedzenie účelu certifikátu alebo detaily o politikách certifikácie. Rozšírenia sú kľúčové pre flexibilitu a prispôsobenie certifikátov špecifickým potrebám [7].

Každý certifikát má jedinečné sériové číslo pridelené CA, ktorá ho vydala. Spolu s identifikátorom danej CA toto číslo zaručuje jeho jednoznačnú identifikáciu. To je dôležité nielen pre správu certifikátov, ale aj pre systémy revokácie [5].

Podpis certifikátu je kľúčovým prvkom, ktorý zaisťuje jeho dôveryhodnosť. Tento podpis generuje certifikačná autorita pomocou svojho privátneho kľúča. Algoritmy použité na tento účel sú asymetrické, čiže verejný kľúč CA môže byť použitý na overenie, či bol certifikát skutočne podpísaný CA [5].

```
Version  ::=  INTEGER  {  v1(0), v2(1), v3(2)  }
CertificateSerialNumber  ::=  INTEGER
Validity ::= SEQUENCE {
    notBefore      Time,
    notAfter       Time
}
Time ::= CHOICE {
    utcTime        UTCTime,
    generalTime    GeneralizedTime
}
UniqueIdentifier  ::=  BIT STRING
SubjectPublicKeyInfo  ::=  SEQUENCE  {
    algorithm            AlgorithmIdentifier,
    subjectPublicKey     BIT STRING
}
Extensions  ::=  SEQUENCE SIZE (1..MAX) OF Extension
Extension  ::=  SEQUENCE  {
    extnID       OBJECT IDENTIFIER,
    critical     BOOLEAN DEFAULT FALSE,
    extnValue    OCTET STRING
}
```

Obr. 3: Štruktúra atribútov TBSCertificate

Atribút issuer obsahuje informácie o certifikačnej autorite, ktorá certifikát vydala. Toto pole je dôležité pri overovaní dôveryhodnosti certifikátu, pretože klienti musia dôverovať certifikačnej autorite, ktorá certifikát vydala [5].

Časové obdobie, počas ktorého je certifikát platný, obsahuje dve hodnoty: `notBefore` a `notAfter`, ktoré určujú začiatok a koniec platnosti certifikátu. Certifikáty s exspirovanou platnosťou môžu predstavovať bezpečnostné riziko, preto je dôležité, aby certifikát nebol platný dlhšie, než je nevyhnutné. Krátka doba platnosti znižuje riziko kompromitácie a zjednodušuje správu certifikátov [5].

Entita, ktorá je držiteľom certifikátu, môže byť fyzická osoba, organizácia, alebo webová stránka. Tieto informácie slúžia na identifikáciu subjektu, ktorému bol certifikát vystavený [5].

Verejný kľúč subjektu, ktorý je držiteľom certifikátu je spojený s algoritmom použitým na kryptografické operácie. Táto informácia je nevyhnutná pre asymetrické šifrovanie [5].

Druhé povinné pole obsahuje identifikáčné údaje a parametre kryptografického algoritmu, ktorý CA použila na podpis tohto certifikátu. Identifikátor algoritmu sa musí zhodovať s tým, ktorý je zadaný v atribúte signature v poli tbsCertificate [5].

```
AlgorithmIdentifier  ::=  SEQUENCE  {
    algorithm     OBJECT IDENTIFIER,
    parameters    ANY DEFINED BY algorithm OPTIONAL  }
}
```

Obr. 4: Štruktúra poľa signatureAlgorithm

Posledné pole obsahuje digitálny podpis CA, ktorý je vypočítaný zo zakódovanej hodnoty prvého poľa. Týmto podpisom CA potvrdzuje, že verejný kľúč naozaj patrí danému subjektu [5].

Takto vydané certifikáty sa ďalej používajú v TLS a SSL protokoloch na prezentačnej vrstve OSI modelu na zabezpečenie komunikácie medzi klientom a serverom. V rámci TLS/SSL handshake procesu server pošle klientovi svoj certifikát na overenie svojej totožnosti. Stav odvolania certifikátu zaslaného serverom si klient môže overiť dvomi najčastejšími spôsobmi, a to pomocou CRL alebo cez OCSP protokol [1].

2.2 CRL

Ako už z jeho mena vyplýva, CRL je list revokovaných certifikátov. Tento spôsob overenia platnosti certifikátov je definovaný priamo v X.509 v2 štandarde a vydáva ho CA, alebo entita poverená danou CA na tento účel. Každý CRL má určitý rozsah, čiže súbor certifikátov, ktoré v ňom môžu byť uvedené. Rozsahom môžu byť napríklad všetky certifikáty vydané danou CA, alebo aj niečo špecifickejšie, ako napríklad "všetky certifikáty vydané zamestnancom spoločnosti X nachádzajúcim sa v meste Y". CRL má tiež presne definovanú syntax a kóduje sa pomocou ASN.1 DER [7].

```
CertificateList  ::=  SEQUENCE  {
    tbsCertList           TBSCertList,
    signatureAlgorithm    AlgorithmIdentifier,
    signatureValue        BIT STRING
}
```

Obr. 5: Štruktúra CRL

Prvé pole obsahuje verziu, identifikátor algoritmu použitého na podpis, názov entity, ktorá CRL vydala, čas vydania tejto verzie, čas vydania ďalšej verzie, odvolané certifikáty a prípadné nepovinné rozšírenia. Môže sa stať aj, že nie sú žiadne odvolané certifikáty, v takom prípade je ich zoznam opomenutý, preto má označenie OPTIONAL [7].

```
TBSCertList  ::=  SEQUENCE  {
    version                 Version OPTIONAL,
    signature               AlgorithmIdentifier,
    issuer                  Name,
    thisUpdate              Time,
    nextUpdate              Time OPTIONAL,
    revokedCertificates     SEQUENCE OF SEQUENCE  {
        userCertificate         CertificateSerialNumber,
        revocationDate          Time,
        crlEntryExtensions      Extensions OPTIONAL
    }  OPTIONAL,
    crlExtensions           [0]  EXPLICIT Extensions
       OPTIONAL
}
```

Obr. 6: Štruktúra poľa tbsCertList

Druhé a tretie pole sú rovnaké ako pri certifikátoch, teda obsahujú identifikátor algortimu použitého na podpis a vypočítanú hodnotu nad zakódovaným prvým poľom [7].

Obr. 7: CRL komunikácia

Ak sa klient rozhodne certifikát servera overiť touto metódou, stiahne CRL, ak ho ešte stiahnutý nemá. Potom si overí, či certifikát, ktorý mu server poslal nie je v tomto zozname. Výhodou tohto systému je jeho jednoduchosť. Zložitosť implementácie takéhoto procesu je minimálna a prevádzkové náklady sú nízke. Vznikajú tu však aj

problémy. CA pravidelne aktualizujú tieto listy, no medzi aktualizáciami môže uplynúť určitý čas. Počas tohto obdobia môžu byť niektoré certifikáty revokované, ale ešte neboli pridané do CRL. Zároveň klient pre účely jedného bezpečného spojenia nepotrebuje celý zoznam revokovaných certifikátov, potrebuje overiť plantosť len jedného. S rastúcou veľkosťou CRL prichádzajú problémy s výkonom a efektivitou [1, 7].

2.3 OCSP

Ďalším bežne používaným spôsobom overenia platnosti certifikátov je OCSP protokol. Tento protokol bol navrhnutý ako alternatíva ku CRL, aby odstránil niektoré obmedzenia spojené s overovaním platnosti certifikátov pomocou CRL, ako je napríklad veľkosť zoznamov a oneskorenie pri ich aktualizácii. OCSP poskytuje dynamickejší a rýchlejší mechanizmus, pretože umožňuje klientovi získať informácie o stave konkrétneho certifikátu v reálnom čase priamo od OCSP respondéra, namiesto sťahovania celého zoznamu odvolaných certifikátov [1]. Je to súhrn pravidiel o tom, ako majú vyzerať dáta, ktoré si musí vymeniť klient s OCSP respondérom na to, aby zistil stav jedného alebo viacerých certifikátov. Skladá sa z dvoch hlavných častí [1, 8].

- OSCP požiadavka
- OCSP odpoveď

OCSP požiadavka

Táto časť protokolu definuje formát a obsah dát, ktoré klient odosiela OCSP respondéru pri overovaní stavu certifikátu. OCSP požiadavka pozostáva z dvoch častí: hlavnej časti (tela požiadavky), ktorá obsahuje všetky potrebné informácie o certifikáte, prípadne certifikátoch, ktoré sa majú overiť, a nepovinného digitálneho podpisu [8].

```
OCSPRequest     ::=     SEQUENCE {
    tbsRequest                  TBSRequest,
    optionalSignature   [0]     EXPLICIT Signature OPTIONAL
}
```

Obr. 8: Štruktúra OCSP požiadavky

Prvé pole požiadavky sa týka základných údajov o verzii OCSP protokolu, ktorú klient používa, a ďalších voliteľných informácií, ako napríklad meno žiadateľa, ktoré môže byť potrebné v závislosti od konfigurácie OCSP respondéra. Pole obsahuje aj identifikátory certifikátov, ktoré sa majú overiť, a ďalšie voliteľné rozšírenia, ktoré môžu rozšíriť

funkcionalitu požiadavky. V prípade, že rozšírenia v požiadavke nie sú podporované OCSP respondérom, ten ich môže ignorovať. Je dôležité poznamenať, že jedna OCSP požiadavka môže zahŕňať viacero certifikátov, čím sa optimalizuje komunikácia medzi klientom a respondérom. Jednotlivé komponenty tela OCSP požiadavky sú definované cez ASN.1 [8].

```
TBSRequest      ::=     SEQUENCE {
    version             [0]   EXPLICIT Version DEFAULT v1,
    requestorName       [1]   EXPLICIT GeneralName OPTIONAL,
    requestList               SEQUENCE OF Request,
    requestExtensions   [2]   EXPLICIT Extensions OPTIONAL
}
```

Obr. 9: Štruktúra poľa TBSRequest

Prvý atribút určuje verziu OCSP protokolu, ktorú klient používa. V súčasnosti sa používa verzia 1 (v1), ktorá je prednastavená ako základná verzia protokolu. V budúcnosti môžu byť pridané ďalšie verzie s vylepšeniami [8].

```
    Version         ::=     INTEGER  { v1(0) }
```

Obr. 10: Štruktúra atribútu version

Druhý, voliteľný atribút umožňuje klientovi zahrnúť meno alebo identifikátor, ktorým sa klient identifikuje voči OCSP respondéru [8].

Tretí atribút je nevyhnutnou časťou požiadavky, ktorá obsahuje zoznam certifikátov, ktoré klient chce overiť. Skladá sa zo sekvencie polí `Request`, ktoré sa zasa skladajú z `CertID` objektu a nepovinných rozšírení [8].

```
Request          ::=    SEQUENCE {
   reqCert                    CertID,
   singleRequestExtensions [0] EXPLICIT Extensions
      OPTIONAL
}
CertID           ::=     SEQUENCE {
   hashAlgorithm        AlgorithmIdentifier,
   issuerNameHash       OCTET STRING,
   issuerKeyHash        OCTET STRING,
   serialNumber         CertificateSerialNumber
}
```

Obr. 11: Štruktúra atribútu requestList

Atribút CertID sa skladá z:

- **hashAlgorithm**: Tento údaj špecifikuje kryptografický algoritmus, ktorý sa použije na výpočet hešu potrebného na identifikáciu certifikátu.
- **issuerNameHash**: Heš mena vydavateľa certifikátu. Tento heš je vypočítaný na základe plného DN vydavateľa, ktoré je použité na jednoznačnú identifikáciu CA.
- **issuerKeyHash**: Heš verejného kľúča vydavateľa, čo je ďalší údaj, ktorý pomáha jednoznačne identifikovať vydavateľa certifikátu.
- **serialNumber**: Sériové číslo certifikátu, ktoré jednoznačne identifikuje konkrétny certifikát vydaný danou certifikačnou autoritou.

Posledný atribút poľa TBSRequest je voliteľný a umožňuje zahrnúť rozšírenia špecifické pre OCSP požiadavku. Tieto rozšírenia môžu poskytovať dodatočné informácie alebo požiadavky, ktoré respondér môže, ale nemusí spracovať [8].

Keď respondér obdrží požiadavku, vyhodnotí, či má správnu štruktúru, ďalej vyhodnotí, či je nakonfigurovaný tak, aby mohol poskytnúť požadovanú službu a nakoniec vyhodnotí, či požiadavka obsahuje potrebné povinné údaje. Potom požiadavku spracuje a odošle klientovi odpoveď [8].

OCSP odpoveď

OCSP odpovede sa skladajú z typu odpovede a samotného obsahu odpovede [8]. Môžu mať rôzne typy, ale je definovaný jeden základný, ktorý musí podporovať každý OCSP respondér aj klient a všetky definitívne odpovede musia byť podpísané. Na podpis odpovede môžu byť použité verejné kľúče jednej z troch entít:

- CA, ktorá daný certifikát vydala
- Dôveryhodný respondér, ktorému klient verí
- Autorizovaný respondér, ktorého poverila daná CA

```
OCSPResponse ::= SEQUENCE {
    responseStatus         OCSPResponseStatus,
    responseBytes          [0] EXPLICIT ResponseBytes
       OPTIONAL
}
```

Obr. 12: Štruktúra OCSP odpovede

Atribút responseStatus určuje stav odpovede a môže mať jednu z preddefinovaných hodnôt. Tieto hodnoty informujú, či bola odpoveď úspešne vytvorená alebo či došlo k chybe [8].

```
OCSPResponseStatus ::= ENUMERATED {
    successful          (0),
    malformedRequest    (1),
    internalError       (2),
    tryLater            (3),
                    -- (4) is not used
    sigRequired         (5),
    unauthorized        (6)
   }
```

Obr. 13: Štruktúra atribútu responseStatus

Druhý atribút responseBytes obsahuje konkrétne údaje týkajúce sa odpovede, ak bola úspešne vytvorená (ak bol stav responseStatus 0). Toto pole je voliteľné, čo znamená, že sa použije len v prípade úspešnej odpovede. Ak odpoveď nie je úspešná, toto pole môže byť prázdne [8].

```
ResponseBytes ::=       SEQUENCE {
    responseType   OBJECT IDENTIFIER,
    response       OCTET STRING
}
```

Obr. 14: Štruktúra atribútu responseBytes

Prvý atribút v responseType je identifikátor objektu, ktorý označuje typ odpovede. Pre základnú OCSP odpoveď, ktorú musí každý OCSP respondér podporovať, je hodnota tohoto atribútu id-pkix-ocsp-basic. Tento identifikátor určuje, že odpoveď je štandardného typu [8].

```
BasicOCSPResponse       ::= SEQUENCE {
    tbsResponseData      ResponseData,
    signatureAlgorithm   AlgorithmIdentifier,
    signature            BIT STRING,
    certs            [0] EXPLICIT SEQUENCE OF Certificate
       OPTIONAL
}
```

Obr. 15: Štruktúra atribútu response

Atribút response v poli responseBytes obsahuje údaje odpovede. Pri základnej odpovedi BasicOCSPResponse sú definované štyri hlavné atribúty: údaje odpovede, použitý algoritmus na podpisovanie, samotný podpis a prípadne zoznam certifikátov použitých na overenie podpisu [8].

```
ResponseData ::= SEQUENCE {
    version             [0] EXPLICIT Version DEFAULT v1,
    responderID             ResponderID,
    producedAt              GeneralizedTime,
    responses               SEQUENCE OF SingleResponse,
    responseExtensions  [1] EXPLICIT Extensions OPTIONAL
}
```

Obr. 16: Štruktúra atribútov ResponseData a ResponderID

Atribút tbsResponseData v BasicOCSPResponse obsahuje údaje o odpovedi, ktoré sa podpisujú. Tieto údaje obsahujú verziu OCSP odpovede, identifikátor respondéra, ktorý vytvoril odpoveď, čas, kedy bola odpoveď vytvorená, zoznam jednotlivých odpovedí týkajúcich sa certifikátov, ktoré boli predmetom požiadavky a prípadné rozšírenia. Aj tieto majú presne definovanú štruktúru [8].

Atribút ResponderID identifikuje OCSP respondéra, ktorý vytvára odpoveď. Sú dva spôsoby, ako môže byť respondér identifikovaný, a to buď pomocou mena alebo hešu jeho verejného kľúča [8].

Atribút SingleResponse je odpoveď na jeden certifikát. Obsahuje identifikátor certifikátu, ktorého stav sa overuje, aktuálny stav certifikátu, časovú pečiatku, ktorá in-

dikuje, kedy bol stav certifikátu naposledy overený, prípadnú časovú pečiatku ďalšej očakávanej aktualizácie stavu certifikátu a voliteľné rozšírenia [8].

```
ResponderID ::= CHOICE {
    byName               [1] Name,
    byKey                [2] KeyHash
}
KeyHash ::= OCTET STRING
SingleResponse ::= SEQUENCE {
    certID                       CertID,
    certStatus                   CertStatus,
    thisUpdate                   GeneralizedTime,
    nextUpdate          [0]      EXPLICIT GeneralizedTime
       OPTIONAL,
    singleExtensions    [1]      EXPLICIT Extensions
       OPTIONAL
}
CertStatus ::= CHOICE {
    good        [0]     IMPLICIT NULL,
    revoked     [1]     IMPLICIT RevokedInfo,
    unknown     [2]     IMPLICIT UnknownInfo
}
RevokedInfo ::= SEQUENCE {
    revocationTime              GeneralizedTime,
    revocationReason    [0]     EXPLICIT CRLReason OPTIONAL
}
```

Obr. 17: Štruktúra atribútov ResponseData a ResponderID

Stav certifikátu môže byť v poriadku, odvolaný alebo neznámy. Ak je certifikát odvolaný, atribút RevokedInfo obsahuje informácie o čase a dôvode revokácie. Takáto odpoveď indikuje, že daný certifikát má byť odmietnutý. Neznáma odpoveď znamená, že respondér nevie o danom certifikáte, čiže buď nepozná CA, ktorá vydala daný certifikát, alebo jednoducho danú CA neobsluhuje. Stav v poriadku znamená, že certifikát s daným identifikátorom a periódou platnosti nebol revokovaný. Nepotvrdzuje to však, že certifikát bol vôbec niekedy vydaný, alebo že čas, kedy bola odpoveď vygenerovaná, je v rámci intervalu platnosti certifikátu. Na posúdenie týchto vlastností môžu však byť použité nepovinné rozšírenia [8].

2.4 Nedostatky OCSP protokolu

Protokol OCSP bol navrhnutý ako rýchlejšia a efektívnejšia alternatíva ku CRL. Napriek svojim výhodám však obsahuje určité slabiny a neefektívne miesta, ktoré môžu negatívne ovplyvniť jeho použitie v praxi. Táto podkapitola sa zameriava na hlavné výzvy OCSP protokolu, vrátane výkonnostných a bezpečnostných problémov a problémov s podporou.

Problémy s výkonom

Výkonnosť je jedným z kľúčových aspektov, ktoré môžu ovplyvniť použiteľnosť a spoľahlivosť OCSP protokolu. Hoci je OCSP navrhnutý tak, aby bol rýchlejší a efektívnejší ako CRL, jeho implementácia a architektúra môžu viesť k viacerým výkonnostným problémom, najmä v prostrediach s vysokou záťažou.

Latencia spôsobená dodatočnými sieťovými požiadavkami

Jedným z hlavných výkonnostných problémov OCSP protokolu je potreba vytvorenia nového sieťového spojenia pre každú požiadavku klienta. Pri každom pokuse o overenie certifikátu klient posiela požiadavku na OCSP respondér a čaká na odpoveď. Táto dodatočná komunikácia predlžuje čas potrebný na dokončenie procesov, ako je TLS handshake a načítanie webovej stránky. Latencia sa taktiež môže zvýšiť, ak je OCSP respondér geograficky vzdialený od klienta alebo ak je spojenie medzi nimi pomalé. Atutxa et al. vo svojej práci z roku 2023 ukázali, že v IIoT prostredí, kde sa využíva DTLS protokol na zabezpečenie spojenia, overenie certifikátu cez OCSP zaberá viac ako 50% celkového času handshake [9].

Škálovateľnosť OCSP respondérov

OCSP respondéry musia byť schopné zvládnuť veľké množstvo požiadaviek v reálnom čase, čo môže predstavovať značný výkonnostný problém. V prostrediach s vysokou záťažou, ako sú veľké webové platformy alebo systémy s miliónmi používateľov, sa OCSP respondéry stávajú úzkym miestom. Ak nie sú dostatočne dimenzované alebo optimalizované, môžu sa stať neodpovedajúcimi alebo reagovať pomaly, čo ovplyvňuje nielen klientov, ale aj celkovú dôveryhodnosť systému [10, 11].

Problémy s bezpečnosťou

OCSP protokol bol navrhnutý s cieľom poskytnúť dynamické a spoľahlivé overovanie platnosti certifikátov. Napriek tomu obsahuje niekoľko bezpečnostných slabín, ktoré môžu ohroziť dostupnosť a integritu tohto procesu.

Závislosť na dostupnosti OCSP respondérov

Jednou z hlavných bezpečnostných slabín OCSP protokolu je jeho závislosť na dostupnosti OCSP respondérov. Ak respondér nie je dostupný, klient nie je schopný overiť

stav certifikátu. To vedie k situáciám, kedy mnoho klientov a webových prehliadačov používa tzv. "soft fail" mechanizmus. Tento prístup znamená, že ak OCSP odpoveď nie je dostupná, klient predpokladá, že certifikát je platný. Soft fail je problematický, pretože umožňuje útočníkovi narušiť spojenie medzi klientom a OCSP respondérom a zabrániť tak overeniu stavu certifikátu. Týmto spôsobom môže byť certifikát, ktorý bol odvolaný, stále akceptovaný klientom, čo vážne ohrozuje bezpečnosť komunikácie [12].

Odhalenie súkromných informácií klienta

OCSP protokol môže neúmyselne ohroziť súkromie používateľov. Pri každej požiadavke na OCSP respondér klient odosiela informácie o konkrétnom certifikáte, ktorý chce overiť. Tieto požiadavky môžu byť sledované, čo umožňuje OCSP respondéru alebo potenciálnym útočníkom získať informácie o tom, aké webové stránky alebo služby klient používa. Táto vlastnosť môže byť využitá na sledovanie online aktivity používateľov a predstavuje významný problém v kontexte ochrany súkromia. Napríklad, ak sa OCSP respondér nachádza v jurisdikcii, ktorá umožňuje monitorovanie siete, údaje o požiadavkách môžu byť ľahko zneužité [13].

Náchylnosť na útoky opakovaným prehratím

Hoci sú OCSP odpovede digitálne podpísané, čo bráni ich priamej modifikácii útočníkom, protokol môže byť náchylný na útoky opakovaným prehratím, pokiaľ nie je v OCSP požiadavkách a odpovediach použitá unikátna hodnota `nonce` [8]. V takomto scenári môže útočník zachytiť platnú, podpísanú OCSP odpoveď, potvrdzujúcu, že certifikát je v poriadku, a neskôr ju opakovane prehrať klientovi. Klient by tak mohol akceptovať zastaranú informáciu o stave certifikátu, aj keď ten už bol medzitým revokovaný [12].

Nedostatočná podpora

Ďalšou prekážkou pri OCSP komunikácii je neúplná alebo nedostatočná podpora zo strany rôznych komponentov infraštruktúry. Tieto problémy môžu brániť jeho širšej adopcii a ovplyvňovať jeho účinnosť a spoľahlivosť v reálnych aplikáciách.

Slabá miera implementácie na strane klientov

Podpora OCSP protokolu na strane klientov, ako sú webové prehliadače a aplikácie, nie je vždy konzistentná. Niektoré prehliadače, napríklad Google Chrome alebo Mozilla Firefox v predvolenom nastavení nepoužívajú OCSP protokol na overovanie certifikátov. Namiesto toho sa spoliehajú na alternatívne mechanizmy, ako je CRLset alebo OneCRL [14]. Táto slabá podpora môže viesť k situáciám, kedy OCSP protokol nie je vôbec využívaný, aj keď by mohol zlepšiť bezpečnosť a efektivitu komunikácie.

Zhrnutie

Aj keď OCSP má potenciál priniesť výhody v oblasti efektivity a rýchlosti, v praxi sa stretáva s viacerými obmedzeniami, ktoré ovplyvňujú jeho výkon, bezpečnosť a podporu. Výkonnostné problémy súvisia najmä s latenciou a vysokými nárokmi na škálovateľnosť OCSP respondérov. Tieto faktory môžu viesť k spomaleniu komunikácie a zvýšeniu prevádzkových nákladov.

Z bezpečnostného hľadiska je OCSP zraniteľný voči viacerým hrozbám, ako sú útoky typu opakovaným prehratím, odhalenie súkromných informácií používateľov či dôsledky "soft fail" mechanizmu pri výpadku OCSP respondérov. Tieto slabiny môžu výrazne ohroziť dôveryhodnosť protokolu a integritu procesu overovania certifikátov. Nedostatočná podpora na strane klientov, najmä v niektorých webových prehliadačoch, ešte viac obmedzuje efektívne využitie tohto protokolu v praxi.

2.5 Existujúce riešenia

Pre zlepšenie efektivity a širšie prijatie OCSP je potrebné riešenie týchto nedostatkov. Odborná komunita už prišla s rôznymi významnými vylepšeniami, ktoré zlepšujú OCSP v týchto oblastiach. V tejto kapitole sú uvedené hlavné existujúce vylepšenia pre OCSP.

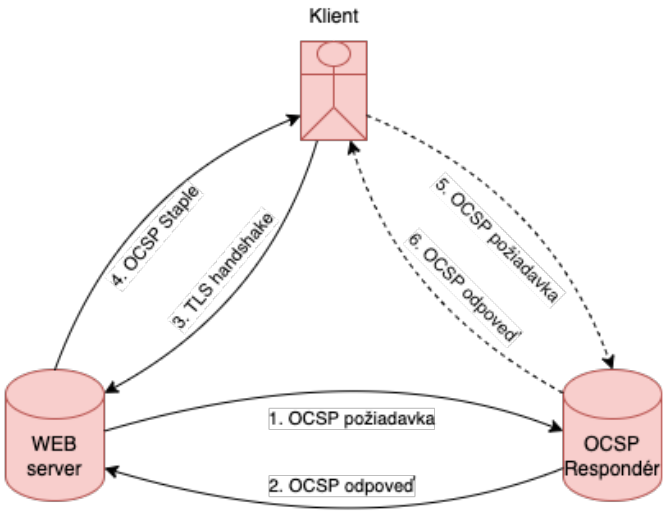
OCSP Stapling

OCSP Stapling je jedným z najvýznamnejších vylepšení OCSP protokolu, ktoré rieši viaceré jeho výkonové a bezpečnostné nedostatky. Táto metóda umožňuje serveru zahrnúť OCSP odpoveď priamo do TLS handshake, čím eliminuje potrebu klienta samostatne kontaktovať OCSP respondér. Zavedenie OCSP Stapling prináša niekoľko výhod, ktoré zvyšujú efektivitu, spoľahlivosť a bezpečnosť procesu overovania platnosti certifikátov [12, 13, 15, 16].

Princíp fungovania OCSP Stapling

V tradičnom OCSP modeli klient posiela požiadavku na OCSP respondér, aby získal informácie o stave konkrétneho certifikátu. Pri OCSP Stapling však server sám periodicky kontaktuje OCSP respondér, získava aktuálnu odpoveď a túto odpoveď následne "pripne" (angl. "staple") k svojmu certifikátu počas TLS handshake komunikácie s klientom. Klient tak dostane OCSP odpoveď priamo od servera a nemusí samostatne komunikovať s OCSP respondérom [12, 15].

Klient si stav certifikátu stále môže overiť aj sám cez OCSP respondér. To je užitočné pre prípady, keď server z akéhokoľvek dôvodu zlyhá v pripojení OCSP odpovede do TLS handshake. V takýchto prípadoch sa systém vracia k obyčajnému OCSP modelu [16].



Obr. 18: Schéma OCSP Stapling komunikácie.

Výhody a nevýhody OCSP Stapling

OCSP Stapling prináša niekoľko kľúčových výhod:

- **Zníženie latencie**
  Eliminuje dodatočné sieťové spojenie medzi klientom a OCSP respondérom, čím skracuje čas potrebný na dokončenie TLS handshake. To zlepšuje používateľskú skúsenosť, najmä pri webových aplikáciách citlivých na výkon [17].

- **Zlepšenie ochrany súkromia**
  Keďže klient nekomunikuje priamo s OCSP respondérom, eliminuje sa možnosť sledovania aktivity používateľa na základe OCSP požiadaviek. Vďaka tomu OCSP Stapling výrazne napomáha k súkromiu koncového užívateľa [15, 18].

Okrem významných, vyššie spomenutých výhod prináša OCSP Stapling so sebou aj nové obmedzenia a výzvy.

- **Presun OCSP komunikácie z klienta na server**
  V praxi sa môžu architektúry serverov rôznych webových stránok líšiť v mnohých aspektoch, keďže každý je uspôsobný na mieru pre svoj používateľský scenár. Vzhľadom na to, že OCSP Stapling pridáva logiku navyše pre tieto servery, aj ich schopnosť na adaptáciu tejto novej logiky môže byť odlišná.

- **Riziko útoku so znížením ochrany**
  Pri využití OCSP Stapling klient nemá spôsob, ako si overiť, či odpoveď bola skutočne priložená alebo nie. Útočník môže zachytiť TLS komunikáciu a odstrániť priloženú OCSP odpoveď zo správy, čím môže vyvolať soft-fail scenár u klienta.

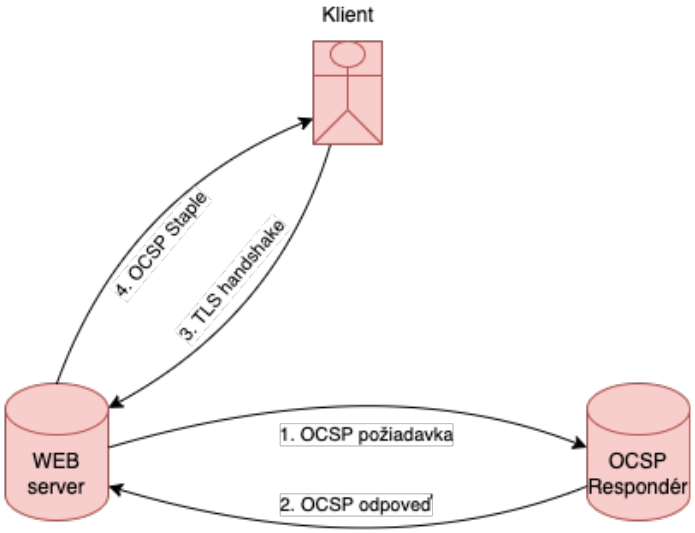
Tento problém bol jednou z hlavných motivácií pre zavedenie OCSP Must-Staple rozšírenia, ktorému sa budeme venovať ďalej.

Aj napriek významným vylepšeniam, ktoré OCSP Stapling prináša oproti základnému OCSP modelu, je táto metóda stále málo používaná. Zulfiqar et al. vo svojom výskume z roku 2021, zameranom na prijatie bezpečnostných mechanizmov u X.509 certifikátov ukázali, že v roku 2020 iba 29.67 % z 50 000 najpoužívanejších webových stránok podporovalo OCSP Stapling [17]. Takáto nízka miera adopcie naznačuje, že ekosystém certifikátov a súvisiace infraštruktúry ešte nie sú dostatočne pripravené na plnú implementáciu OCSP Stapling. Na podporu širšieho prijatia tejto technológie budú potrebné ďalšie vylepšenia, ako napríklad lepšia podpora na strane serverov a klientov a efektívnejšia implementácia protokolu.

OCSP Must-Staple

Ďalším vylepšením OCSP protokolu je OCSP Must-Staple, ktoré zvyšuje jeho bezpečnosť a spoľahlivosť tým, že priamo vyžaduje priloženie OCSP odpovede k certifikátu počas TLS handshake. Táto funkcionalita je implementovaná ako rozšírenie v certifikáte a zaväzuje server k poskytovaniu aktuálnej OCSP odpovede. V prípade, že server OCSP odpoveď nepriloží, klient automaticky odmietne nadviazať spojenie [13, 14].

Princíp fungovania OCSP Must-Staple



Obr. 19: Schéma OCSP Must-Staple komunikácie.

OCSP Must-Staple je realizovaný pomocou špeciálneho rozšírenia X.509 certifikátu s názvom TLS Feature Extension. Toto rozšírenie obsahuje špecifikáciu, že server musí poskytovať OCSP odpoveď ako súčasť TLS handshake. Klient, ktorý prijme certifikát

s týmto rozšírením, overí prítomnosť OCSP odpovede. Ak odpoveď chýba, spojenie je zamietnuté ako neplatné. Mechanizmus teda funguje ako striktné vynucovanie OCSP Stapling, čím eliminuje riziko nastatia "soft fail" scenárov [13, 16].

Výhody a nevýhody OCSP Must-Staple

Vzhľadom na to, že tento model vychádza z OCSP Stapling, prináša so sebou všetky jeho výhody. Okrem nich však Must-Staple poskytuje aj jedno významné bezpečnostné vylepšenie. Základný OCSP model aj OCSP Stapling umožňujú "soft fail", pri ktorom prehliadač povolí spojenie, aj keď nie je možné overiť stav certifikátu, napríklad pri výpadku OCSP respondéra. OCSP Must-Staple tento problém rieši tým, že prítomnosť OCSP odpovede je povinná. Ak server nepriloží aktuálnu OCSP odpoveď, spojenie je automaticky odmietnuté, čím sa eliminuje riziko takýchto scenárov [16].

Tak ako pri OCSP Stapling, aj Must-Staple si vyžaduje dôslednú implementáciu na strane serverov aj klientov, no jeho podpora je stále obmedzená. Staršie webové servery a prehliadače často nepoznajú OCSP Must-Staple alebo ho nepodporujú, čo obmedzuje jeho praktické využitie. Friess et. al. vo svojom výskume z roku 2023 zistili, že až 89.5 % z 530 000 klientov jednoducho odignorovalo Must-Staple rozšírenie certifikátov pri načítavaní webovej stránky [12].

Aj napriek početným vylepšeniam oproti základnému OCSP modelu je aktuálna adopcia tejto metódy minimálna. Pre dosiahnutie plného potenciálu OCSP Must-Staple je preto nevyhnutná lepšia podpora v infraštruktúre a zvýšená adopcia zo strany správcov webových aplikácií a klientov.

OCSP Multi-Stapling

Podobne ako Must-Staple, aj OCSP Multi-Stapling je rozšírenie OCSP Stapling. Toto rozšírenie umožňuje serveru priložiť viacero OCSP odpovedí v rámci jedného TLS handshake. To je užitočné najmä v situáciách, keď je potrebné overiť celý certifikátový reťazec, nielen koncový certifikát [13].

V praxi sú certifikáty totiž často podpísané certifikátovými reťazcami. Certifikáty v takomto reťazci sú hierarchicky usporiadané a každý certifikát je podpísaný nadriadeným certifikátom až po koreňový certifikát, ktorý je dôveryhodný a patrí CA. Ak klient overuje platnosť koncového certifikátu, často potrebuje overiť aj ostatné certifikáty v takomto reťazci. V základnom OCSP Stapling modeli toto nie je domyslené, a napriek tomu, že server pošle OCSP odpoveď pre svoj certifikát, klient si v takýchto prípadoch musí sám overiť platnosť ostatných certifikátov v reťazci [13, 19].

OCSP Multi-Stapling tento problém rieši tým, že server pripraví a priloží OCSP odpovede pre viacero certifikátov v reťazci v rámci jednej komunikácie s klientom.

Princíp fungovania OCSP Multi-Stapling

1. **Získanie OCSP odpovedí**
   Server periodicky získava OCSP odpovede pre všetky certifikáty v certifikátovom reťazci od príslušných OCSP respondérov.

2. **Priloženie OCSP odpovedí**
   Počas TLS handshake server priloží OCSP odpovede nielen pre svoj koncový certifikát, ale aj pre všetky ostatné certifikáty v reťazci.

3. **Overenie klientom**
   Klient prijme OCSP odpovede v rámci TLS komunikácie a vie si overiť platnosť koncového certifikátu spolu s celým reťazcom.

Týmto spôsobom klient získa všetky potrebné informácie na overenie platnosti certifikátov naraz, bez potreby dodatočnej komunikácie s OCSP respondérmi.

Výhody a nevýhody OCSP Multi-Stapling

Aj táto metóda so sebou prináša všetky výhody OCSP Stapling. Okrem toho však má klient okamžite prístup k OCSP odpovediam pre všetky certifikáty v reťazci, čo skracuje čas potrebný na ich overenie. Tento prístup je efektívny najmä v systémoch, kde je certifikátový reťazec zložený z viacerých certifikátov.

Nevýhodou tohto prístupu je zväčšenie veľkosti TLS handshake. Okrem toho tu tiež platí závislosť na podpore zo strany serverov a klientov [11].

Compressed Certificate Status Protocol

Ďalším existujúcim vylepšením je CCSP, ktoré sa zameriava na zníženie dátovej náročnosti a zrýchlenie procesu overovania platnosti certifikátov. Cieľom CCSP je komprimovať OCSP odpovede, aby sa zmenšila veľkosť prenášaných dát a zároveň sa zachovala bezpečnosť a integrita procesu.

CCSP prináša novú abstrakciu, tzv. "podpísané kolekcie" v ktorých sú informácie o stave viacerých certifikátov uchované v jednej odpovedi, čím sa znižuje réžia. Okrem toho sú tieto kolekcie následne komprimované podľa priestoru a času [20].

Princíp fungovania CCSP

1. **Podpísané kolekcie**
   Podpísané kolekcie sú reprezentované bitovou mapou, v ktorej každý bit reprezentuje jeden certifikát a jeho hodnota predstavuje informáciu o jeho platnosti. Kolekcia sa následne podpíše a vydá [20].

2. **Optimalizácia podľa pamäťového priestoru**
   Keďže väčšina certifikátov ktorejkoľvek CA nie je revokovaná [20], veľká časť

takejto bitmapy má rovnaké hodnoty. To vytvára priestor pre kompresiu samotnej bitmapy. Na tento účel autori implementovali podporu pre algoritmy DEFLATE a Golomb [20].

3. **Optimalizácia podľa času**
   Každá kolekcia sa podpisuje a vydáva periodicky, interval medzi jednotlivými vydaniami je v rozmedzí sekúnd až minút. Keďže rozdiely mezdi jednotlivými verziami sú v tak malom čase minimálne, ak vôbec nejaké, autori zaviedli systém pre optimalizáciu podľa času. Namiesto vydávania novej podpísanej kolekcie v každej iterácii cyklu, sa väčšinu času vydávajú iba jej zmeny oproti poslednej verzii. Celá nová podpísaná kolekcia sa vydáva v intervaloch niekoľkých hodín [20].

4. **Podpora pre OCSP Stapling**
   Pre ešte väčšie zlepšenie autori zahrnuli do návrhu mechanizmus pre pripnutie podpísaných kolekcií alebo ich posledných zmien rovno ku certifikátu počas TLS handshake, čím využívajú aj výhody, ktoré prináša OCSP Stapling [20].

Výhody a nevýhody CCSP

Hlavnou výhodou tohoto riešenia je použitie komprimovaných podpísaných kolekcií, ktoré významne redukujú objem prenášaných dát. Autori vo svojom článku ukázali, že ich riešenie spotrebuje vyše stonásobne menej úložiska v porovnaní s tradičnými CRL a znižuje počet podpisov o viac než milión-násobok oproti klasickému OCSP Stapling [20].

Ďalšou výhodou tohto riešenia je kompatibilita s OCSP Stapling, čo prináša ďalšie výhody, najmä ochranu súkromia používateľa [20].

Nevýhodou tohto prístupu je opäť to, že vyžaduje úpravy na strane OCSP respondérov a serverov na generovanie a správu komprimovaných odpovedí ako aj na strane klientov na spracovávanie podpísaných kolekcií v odpovediach.

TinyOCSP

TinyOCSP je vylepšenie zamerané na minimalizáciu veľkosti OCSP požiadaviek a odpovedí, čo je kritické pre zariadenia v IoT prostredí s obmedzenými výpočtovými zdrojmi a sieťovými kapacitami. Cieľom TinyOCSP je zachovať funkcionalitu klasického OCSP pri výraznom znížení dátovej záťaže a spotreby energie, ktoré sú kritickými faktormi v aplikáciách pre nízkonákladové a batériovo napájané zariadenia [21].

Hlavné optimalizácie

TinyOCSP prináša niekoľko kľúčových zmien, ktoré umožňujú minimalizovať dátové požiadavky:

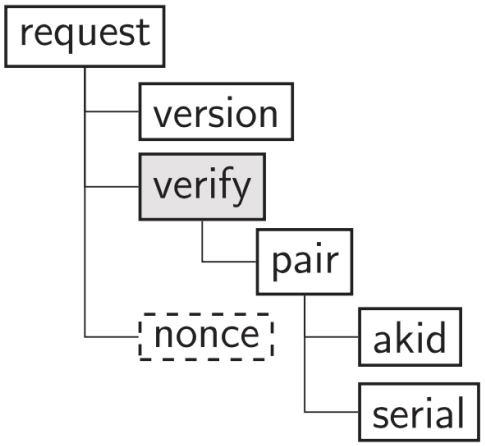
- **CBOR namiesto ASN.1**
  TinyOCSP používa na kódovanie požiadaviek a odpovedí CBOR, ktoré je oveľa kompaktnejšie a efektívnejšie pre IoT zariadenia ako ASN.1 DER kódovanie.

- **Využitie AKID rozšírenia pre X.509 certifikáty**
  Vo východiskovej definícii OCSP je identifikácia relevantnej CA v požiadavkách vyriešená pomocou identifikátoru hešovej funkcie a hešu identifikátora CA. Autori TinyOCSP ukázali, že v najlepšom prípade táto informácia zaberá až 53 bajtov, pričom pri zložiteších hešových funkciach to môže byť ešte viac. Riešením v TinyOCSP je využitie rozšírenia Authority Key Identifier z X.509 certifikátov, ktoré slúži na rovnaký účel a má podľa RFC 5280 §4.2.1.1 odporúčanú dĺžku iba 8 bajtov [7, 21].

- **Odstránenie redundantných dát z odpovedí**
  TinyOCSP v odpovedi neposiela znovu certifikát, ktorý sa overuje. Namiesto toho podpisuje a posiela konkatenáciu požiadavky a odpovede, čím autentizuje odpoveď bez opakovania týchto dát. V rámci optimalizácie sú z odpovedí ešte odstránené polia obsahujúce informácie, ktoré sú strane posielajúcej požiadavku už známe a redundantné časové údaje. Príkladom sú polia ako `responderID` a niektoré časové značky, napríklad `producedAt`. Týmto prístupom sa minimalizuje veľkosť OCSP odpovede bez straty funkčnosti, pretože redundantné informácie nie sú pre príjemcu prínosné a iba zbytočne zvyšujú dátový objem [21].

Štruktúra TinyOCSP správ

Požiadavka obsahuje tri hlavné atribúty, a to sú verzia, sekvencia certifikátov na overenie a nepovinná jednorázová náhodná hodnota [21].



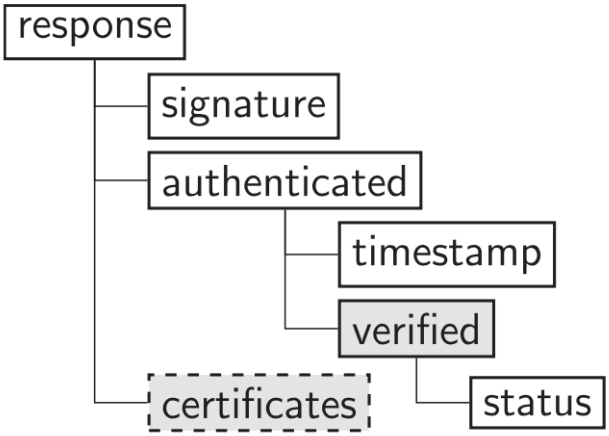
Obr. 20: Štruktúra TinyOCSP požiadavky zakódovaná pomocou CBOR.

Atribút verzia má veľkosť jedného bajtu a slúži klientovi na to, keď chce od res-

pondéra vyžiadať, aby ku odpovedi priložil svoj vlastný certifikát. Hoci autori nešpecifikovali aké konkrétne hodnoty môže nadobúdať, upresnili, že východiskový stav je ten, kedy sa certifikát respondéra nevyžaduje [21].

Certifikáty, ktoré klient chce overiť sú definované ako pár AKID a sériového čísla čísla certifikátu a dokopy majú dvanásť bajtov pre jeden certifikát [21].

Jednorázová náhodná hodnota `nonce` je jedinou nepovinnou súčasťou požiadavky. Slúži ako ochrana pred útokmi opätovného prehratia, pri ktorých by útočník mohol zachytiť platnú OCSP odpoveď a neskôr ju neoprávnene znovu použiť. Vďaka jedinečnej hodnote `nonce` môže klient overiť, že odpoveď od OCSP respondéra zodpovedá konkrétnej požiadavke a nie je zneužitá [21].



Obr. 21: Štruktúra TinyOCSP odpovede zakódovaná pomocou CBOR.

Odpoveď v TinyOCSP obsahuje tri hlavné atribúty, medzi ktoré patria podpis, sekvencia odpovedí na certifikáty z požiadavky, a nepovinný zoznam certifikátov samotného respondéra [21].

Podpis slúži na autentizáciu správy takisto ako pri bežnom OCSP. Počíta sa nad konkatenáciou celej požiadavky a sekvenciou stavov certifikátov v odpovedi a pre jeden certifikát má veľkosť 67 bajtov [21].

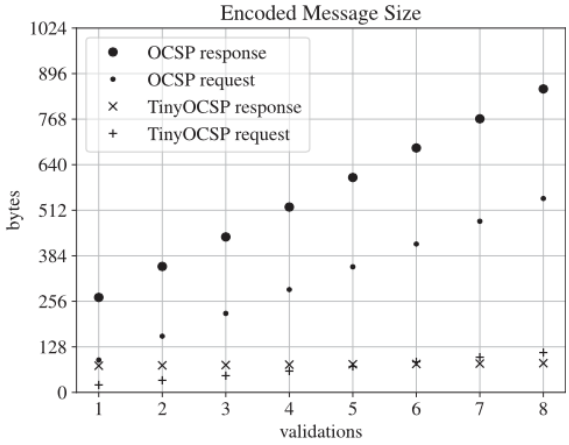
Atribút `authenticated` obsahuje časovú pečiatku a pole stavov, ktoré mapuje stavy jednotlivých certifikátov v takom poradí v akom boli v požiadavke. Časová pečiatka pečiatka udáva čas, kedy respondér naposledy aktualizoval informácie o danom certifikáte a je zakódovaná v POSIX štandarde, vďaka čomu má veľkosť iba štyri bajty. Celý tento atribút má tak pre jeden certifikát veľkosť päť bajtov [21].

Posledným atribútom je zoznam certifikátov respondéra, ktorý nie je povinný, respondér ho uvádza iba na vyžiadanie od klienta.

Porovnanie s bežným OCSP

Keďže základný OCSP model obsahuje rôzne nepovinné rozšírenia, ktoré môžu

pridať ďalšie bajty, veľkosti požiadaviek a odpovedí sú závislé na konkrétnych konfiguráciach. Všetky však musia obsahovať povinné polia definované v štandarde pre OCSP, z čoho vyplýva, že požiadavky aj odpovede rastú lineárne s počtom certifikátov, ktoré obsahujú [21].



Obr. 22: Porovnanie veľkostí správ v OCSP a TinyOCSP.

Autori pre účely porovnania zobrali konfiguráciu bežného OCSP, ktorá bola optimalizovná tak, aby správy mali čo najmenšiu veľkosť. Túto konfiguráciu následne porovnali s TinyOCSP modelom pre ilustráciu úšetrených dát [21].

V bežnom OCSP modeli sa v odpovediach opätovne posielajú celé identifikátory všetkých certifikátov, ktoré boli súčasťou požiadavky. Táto redundancia vedie k výraznému zväčšeniu správ, pri overovaní väčšieho počtu certifikátov [21].

Naopak, TinyOCSP eliminuje túto redundanciu tým, že pre každý dodatočne overovaný certifikát pridá do odpovede iba jeden bajt navyše, a síce stav certifikátu. Vďaka tomuto prístupu je rast veľkosti odpovedí v TinyOCSP minimálny, čo prináša významné úspory dátovej prevádzky. Ako ilustruje graf vyššie, rozdiel medzi bežným OCSP a TinyOCSP sa pri väčšom počte certifikátov stáva čoraz výraznejším [21].

TinyOCSP predstavuje efektívnu optimalizáciu klasického OCSP protokolu, ktorá sa zameriava na minimalizáciu veľkosti prenášaných správ a zlepšenie výkonu v prostredí s obmedzenými zdrojmi. Vďaka využitiu kompaktného CBOR kódovania, efektívnemu AKID rozšíreniu a odstráneniu redundantných dát z odpovedí dokáže TinyOCSP výrazne znížiť dátovú náročnosť komunikácie. Tieto optimalizácie umožňujú, aby veľkosť odpovedí rástla takmer konštantne aj pri zvyšujúcom sa počte certifikátov, čím TinyOCSP prekonáva tradičný OCSP model. Vďaka týmto vlastnostiam je TinyOCSP vhodným riešením pre IoT zariadenia a iné aplikácie, kde je kľúčové šetriť dátovú prevádzku a energetické zdroje [21].

Zhrnutie

Táto kapitola sa zaoberala rôznymi metódami, ktoré zlepšujú efektivitu a bezpečnosť OCSP protokolu. Medzi najvýznamnejšie patrí OCSP Stapling, ktorý rieši hlavné nevýhody tradičného OCSP, ako sú vysoká latencia a záťaž OCSP respondérov, tým, že umožňuje serveru poskytovať OCSP odpovede priamo počas TLS handshake. Rozšírenia ako OCSP Must-Staple a OCSP Multi-Stapling ďalej rozvíjajú tento koncept a poskytujú pridanú hodnotu v oblasti spoľahlivosti a schopnosti overovať celé certifikátové reťazce.

Metódy ako TinyOCSP a CCSP sa odlišujú tým, že upravujú samotný protokol, čím minimalizujú dátovú náročnosť a veľkosť OCSP správ. Tieto prístupy sú obzvlášť užitočné pre IoT zariadenia a prostredia s obmedzenými zdrojmi, pretože výrazne šetria dátovú prevádzku. Na rozdiel od OCSP Stapling a jeho rozšírení, ktoré pracujú so súčasnou infraštruktúrou, tieto riešenia si vyžadujú zásahy do protokolu, čo ich implementáciu robí náročnejšou.

2.6 Význam vyvažovania záťaže v sieti pre protokol OCSP

Vzhľadom na to, že overovanie platnosti certifikátov je súčasťou procesu nadviazania zabezpečeného spojenia a môže priamo ovplyvniť rýchlosť načítania webových stránok, je nevyhnutné zabezpečiť vysokú dostupnosť a rýchlu odozvu OCSP respondérov [22]. V tomto kontexte zohráva vyvažovanie záťaže v sieti kľúčovú úlohu pri optimalizácii výkonu a spoľahlivosti OCSP služieb. Je to obzvlášť užitočné v situáciách, kedy veľké množstvo klientov súčasne overuje stav certifikátov, alebo keď konkrétny OCSP respondér z akéhokoľvek dôvodu nefunguje.

Existujú rôzne metódy vyvažovania záťaže, ktoré sa bežne používajú v sieťových infraštruktúrach. Medzi najbežnejšie patria [23]:

- **Cyklické rozdeľovanie záťaže**
  Požiadavky sú priraďované serverom postupne v cyklickom poradí, čo zabezpečuje rovnomerné rozloženie záťaže bez ohľadu na aktuálnu zaťaženosť serverov.

- **Najmenší počet spojení**
  Nová požiadavka je smerovaná na server s najmenším počtom aktívnych spojení, čo je výhodné v prostrediach s nerovnomernou dĺžkou trvania spojení.

- **Hešovanie IP adresy**
  Priraďovanie požiadaviek je založené na hešovaní IP adresy klienta, čo zabezpečuje, že požiadavky od rovnakého klienta sú smerované na ten istý server, podporujúc tak konzistenciu spojení.

- **Cyklické rozdeľovanie s váhami**
  Každému serveru je priradená váha podľa jeho výkonu alebo kapacity; servery s vyššou váhou dostávajú viac požiadaviek, čo umožňuje efektívne využitie zdrojov.

- **Dynamické vyvažovanie záťaže**
  Táto metóda monitoruje aktuálnu záťaž a výkon serverov v reálnom čase a dynamicky priraďuje požiadavky na základe týchto informácií, čo zabezpečuje optimálnu distribúciu záťaže.

Implementácia vhodnej metódy vyvažovania záťaže je kľúčová pre zabezpečenie efektívneho a spoľahlivého fungovania OCSP služieb [21]. Pri OCSP požiadavkách, ktoré sú často krátke, no veľmi frekventované, je nízka latencia veľmi dôležitá. To znamená, že vyvažovač záťaže by mal byť schopný svižne reagovať na nárazové výkyvy v počte prichádzajúcich požiadaviek a zabezpečiť rovnomerné rozdelenie tak, aby sa minimalizovalo čakanie na odpoveď. Tým sa predchádza preťaženiu jednotlivých OCSP respondérov, znižuje sa riziko výpadkov a zabezpečuje sa, aby boli požiadavky spracované čo najrýchlejšie. Takto nastavené vyvažovanie nielen zvyšuje dostupnosť OCSP overovania, ale tiež prispieva k lepšej užívateľskej skúsenosti a vyššej bezpečnosti systémov, ktoré na OCSP protokole závisia.

2.7 Význam efektívneho overovania certifikátov pre organizácie

Efektívne overovanie platnosti digitálnych certifikátov má kľúčový význam pre bezpečnosť, spoľahlivosť a dôveryhodnosť informačných systémov organizácie. V modernom IT ekosystéme, kde digitálne certifikáty hrajú kľúčovú úlohu pri zabezpečení komunikácie, môže neefektívny alebo nespoľahlivý systém overovania platnosti certifikátov predstavovať významné riziko. Táto kapitola analyzuje širšie súvislosti a význam projektu z pohľadu manažmentu a organizácií, ktoré OCSP protokol využívajú.

Dôsledky zlyhania certifikačných služieb

Zlyhanie alebo oneskorenie OCSP overovania môže mať závažné dôsledky pre organizácie a ich používateľov. Ak klienti nedokážu overiť platnosť certifikátov, môžu byť vystavení riziku útokov. To môže mať za následok únik citlivých informácií, finančné straty a oslabenie dôvery zákazníkov v služby organizácie. Nedostupnosť OCSP služieb môže viesť k neovereniu platnosti certifikátu, čo má za následok zníženie bezpečnosti spojenia a riziko akceptácie odvolaného certifikátu. V kritických odvetviach, ako je finančný sektor či zdravotníctvo, môžu byť tieto dôsledky ešte závažnejšie, vrátane právnych postihov a sankcií [13].

Prevádzkové náklady

Prevádzka OCSP respondérov je náročná na zdroje, pretože vyžaduje rýchle spracovanie veľkého množstva požiadaviek v reálnom čase. Neefektívne overovanie certifikátov zvyšuje nároky na infraštruktúru, čo vedie k vyšším prevádzkovým nákladom [24]. Zlepšením efektivity OCSP, napríklad prostredníctvom optimalizácie veľkosti správ alebo zavedením cache mechanizmov, je možné tieto náklady výrazne znížiť a umožniť efektívnejšie využitie zdrojov.

Regulačné a právne požiadavky

V mnohých odvetviach existujú prísne regulačné a právne požiadavky týkajúce sa bezpečnosti a ochrany údajov. Napríklad finančné inštitúcie, zdravotníctvo alebo vládne systémy často podliehajú reguláciám, ktoré vyžadujú, aby bolo možné certifikáty okamžite overiť. Nesplnenie týchto požiadaviek môže mať právne dôsledky alebo viesť k pokutám a strate licencie. Zlepšením OCSP je možné splniť tieto požiadavky efektívnejšie a zabezpečiť vyššiu úroveň súladu so zákonmi a reguláciami [25].

Zhrnutie

Investícia do zlepšenia OCSP protokolu prináša strategické výhody. Zníženie nákladov, zlepšenie bezpečnosti, splnenie regulačných požiadaviek a zvýšenie dôveryhodnosti služieb predstavujú významné prínosy, ktoré môžu podporiť dlhodobý úspech organizácie v digitálnom prostredí. To môže viesť k získaniu konkurenčnej výhody na trhu, zvýšeniu trhového podielu a posilneniu značky organizácie.

2.8 Prenos OCSP komunikácie

V rámci analýzy je potrebné spomenúť protokol HTTP, ktorý slúži na transtport OCSP komunikácie [18]. HTTP je protokol aplikačnej vrstvy, ktorý sa používa na prenos dát medzi klientom a serverom. V prípade OCSP je jeho úlohou sprostredkovať požiadavky od klienta, napríklad webového prehliadača, smerom k OCSP respondéru a spätné odpovede od respondéra ku klientovi. HTTP je dôležitou súčasťou OCSP komunikácie a jeho vlastnosti významne ovplyvňujú výkon a efektivitu celého systému.

Základné vlastnosti

HTTP funguje na princípe požiadavka-odpoveď, čo znamená, že klient pošle požiadavku na server a následne čaká na odpoveď. Tento model je pomerne jednoduchý a vychádza z neho aj model pre OCSP. HTTP má pre splnenie rôznych požiadaviek klientov implementované rôzne metódy, z hľadiska OCSP sú relevantné metódy GET a POST [26].

GET

Táto metóda sa používa prevažne na načítanie obsahu stránok. Jej hlavička obsahuje identifikátor metódy, URL, prípadné nepovinné parametre a verziu HTTP protokolu. Get metóda spravidla nemá telo [26].

```
GET <URL>["?"<Parametre>] HTTP/Verzia
```

Obr. 23: Základná štruktúra GET metódy

V rámci OCSP komunikácie sa táto metóda používa pri jednoduchých požiadavkách, kde sa haš všetkých potrebných informácií zmestí do poľa pre URL. Problémom tu však je limit pre počet znakov v URL, ktorý je 2048. Preto ak klient potrebuje overiť viac certifikátov naraz, požíva sa nasledovná metóda [8].

POST

Táto metóda slúži v kontexte HTTP komunikácie pre poslanie dát od klienta na server, napríkad pri vyplnení formulára na stránke alebo pri vytvorení konta. Jej hlavička vyzerá podobne ako pri GET, s tým rozdielom, že na začiatku je samozrejme odlišný identifikátor metódy. Na rozdiel od GET, metóda POST má spravidla telo [26].

```
POST  <URL>["?"<Parametre>] HTTP/Verzia
<Telo>
```

Obr. 24: Základná štruktúra POST metódy

OCSP túto metódu využíva vtedy, keď sa požiadavka klienta nezmestí do URL poľa v metóde GET. V takom prípade sa požiadavka ukladá do tela POST metódy a OCSP respondér na druhom konci ju spracováva rovnako ako tie ktoré mu prišli cez GET [8].

HTTP cache

Okrem vyššie uvedených častí, HTTP metódy môžu obsahovať aj nepovinné hlavičky, ktoré slúžia na upresnenie informácií pre rôzne účely. Jednou z týchto hlavičiek je `Cache-Control`, ktorá umožňuje nastavenie pravidiel pre cache HTTP komunikácie [26].

Cache spočíva v ukladaní odpovedí na často opakované požiadavky a je jednou z najväčších výhod HTTP protokolu. Keď klient pošle požiadavku, HTTP cache, napríklad v prehliadači, najskôr overí, či už má odpoveď uloženú a či je táto odpoveď stále platná. Ak je odpoveď dostupná a platná, cache okamžite vráti uložené dáta, čím sa eliminuje potreba komunikácie so vzdialeným serverom. Tento proces znižuje počet priamych komunikácií medzi klientom a serverom, čím znižuje latenciu, šetrí šírku

pásma siete a znižuje záťaž na pôvodný server. V kontexte OCSP komunikácie je cache kľúčové, pretože tie isté OCSP odpovede sú často požadované viac krát.

Statický a dynamický obsah

Z hľadiska cache poznáme dva druhy obsahu, statický a dynamický. Statický obsah predstavuje dáta, ktoré sa v čase menia len zriedkavo alebo vôbec. Typicky ide o súbory, akými sú obrázky, CSS a JavaScript súbory, či rôzne dokumenty, na ktoré sa bežne používajú GET požiadavky. Tieto súbory sa pri dodržaní vhodne zvolených hlavičiek na strane servera dajú efektívne ukladať do cache na dlhšie obdobie. Klient, prípadne sprostredkujúca cache, napríklad CDN, si tak dokáže ukladať a opakovane používať nezmenené verzie týchto súborov bez nutnosti opätovného kontaktu so serverom [26].

Dynamický obsah, naopak, predstavuje dáta, ktoré sa môžu pri každej požiadavke meniť. Typickým príkladom sú odpovede z API služieb, výsledky vyhľadávania či personalizovaný obsah generovaný na strane servera, na ktoré sa zvyčajne používajú POST požiadavky. Keďže sa tieto dáta môžu meniť aj pri dvoch po sebe nasledujúcich požiadavkách, cache dynamického obsahu býva limitovaná a riadená dodatočnými mechanizmami, alebo úplne vypnutá [26].

V kontexte OCSP komunikácie poskytujú GET aj POST metódy taký istý obsah. Aj keď vzhľadom na dobu platnosti odpovede pre certifikát nie je úplne statický, stále sa dá správnym prístupom efektívne ukladať do cache.

2.9 Zhodnotenie analýzy

Analýza problematiky objasnila teoretické oblasti potrebné pre pochopenie tejto práce, uviedla súčasný stav v oblasti OCSP protokolu so zameraním na jeho slabé stránky a následne určila náš ciel pre riešenie problému zlepšenia efektivity OCSP protokolu. V kontexte OCSP sa na získavanie rovnakého typu obsahu využívajú metódy GET aj POST, pričom samotný obsah nie je úplne statický pre obmedzenú dobu platnosti odpovedí pre certifikát, no pri správnom prístupe je možné ho efektívne uložiť do cache. Problém však spočíva v tom, že POST požiadavky sa všeobecne spájajú s dynamickými operáciami, pri ktorých sa odpovede spravidla neukladajú do cache, čo nereflektuje špecifiká OCSP protokolu. V tejto práci sa preto zameriame na zlepšenie tohto nedostatku návrhom efektívneho mechanizmu ukladania GET a POST požiadaviek pre účely OCSP komunikácie do cache.

3 Špecifikácia požiadaviek

V tejto kapitole uvedieme funkčné a nie-funkčné požiadavky pre naše riešenie.

3.1 Nie-funkčné požiadavky

Riešenie bude implementované ako počítačová aplikácia, naprogramovaná v jazyku Java, ktorý je vhodnou voľbou pre tento projekt, pretože poskytuje kompatibilitu pre všetky hlavné operačné systémy, jednoduchú integráciu s existujúcimi riešeniami a umožňuje efektívne spracovanie HTTP požiadaviek. Používateľské rozhranie bude textové v príkazovom riadku.

Ďalšie kvalitatívne vlastnosti:

- **Škálovateľnosť cache** Cache musí byť schopná spravovať odpovede pre minimálne 10 000 jedinečných požiadaviek.
- **Konfigurácia systému** Aplikácia musí umožňovať konfiguráciu maximálnej veľkosti cache, maximálnej doby platnosti odpovedí a ďalších podstatných súčastí prostredníctvom konfiguračného súboru.
- **Práca s chybovými stavmi** Ak v akomkoľvek bode procesu aplikácia nie je schopná plniť svoj účel, musí v logovacom súbore poskytnúť deskriptívne informácie o tom, prečo nie je možné pokračovať a nesmie spadnúť s chybou.

3.2 Funkčné požiadavky

- **Prijímanie a spracovanie GET a POST OCSP požiadaviek**
  Aplikácia musí byť schopná prijať a spracovať požiadavky na OCSP respondéra, ktoré musia byť validované na správnosť formátu a obsahu podľa štandardu RFC 6960 [8].
- **Podpora náhodnej hodnoty v požiadavkách**
  Aplikácia musí vedieť pracovať s požiadavkami, ktoré obsahujú náhodnú hodnotu `nonce`. Odpovede pre takéto požiadavky nesmú byť brané z cache.
- **Komunikácia s OCSP respondérom**
  Ak požiadavka nemá zodpovedajúcu odpoveď v cache, systém musí kontaktovať OCSP respondér, získať aktuálnu odpoveď a uložiť ju do cache.
- **Cache odpovedí**
  Aplikácia musí byť schopná ukladať odpovede na GET aj POST požiadavky do cache tak, aby ich bolo možné opätovne využiť pri ďalších požiadavkách na tie isté certifikáty, pokiaľ nevypršala doba platnosti uvedená v OCSP odpovediach.

- **Vypršanie platnosti odpovedí v cache**
  Odpovede musia byť automaticky odstránené z cache po prekročení ich definovanej doby platnosti.

- **Logovanie stavov aplikácie**
  Systém musí zaznamenávať všetky podstatné procedúry spolu s časovými pečiatkami ich uskutočnenia do logovacieho súboru.

4 Návrh riešenia

Táto kapitola obsahuje návrh aplikácie, vrátane podrobného návrhu pre aplikáciu a príkladu konfiguračného súboru.

4.1 Návrh HTTP servera

HTTP server bude prijímať GET a POST požiadavky od klientov. Tieto požiadavky budú obsahovať OCSP požiadavky na overenie stavu certifikátov. Server bude spracovávať prichádzajúce požiadavky, vyhľadávať odpovede v cache a v prípade potreby kontaktovať OCSP respondér. Následne server vráti odpoveď klientovi.

Spracovanie a validácia OCSP požiadaviek

1. **Prijatie HTTP požiadavky**
   HTTP server príjme GET alebo POST požiadavku obsahujúcu OCSP požiadavku.

2. **Extrahovanie OCSP požiadavky**
   Zo samotnej HTTP požiadavky sa extrahuje OCSP požiadavka, ktorá obsahuje identifikátory certifikátov určených na overenie.

3. **Validácia formátu požiadavky**
   Požiadavka sa skontroluje, či má správny formát podľa ASN.1 DER kódovania a zodpovedá štandardu RFC 6960 [8]. Ak niečo nesedí, server odošle klientovi informáciu o chybnej požiadavke a proces tu končí.

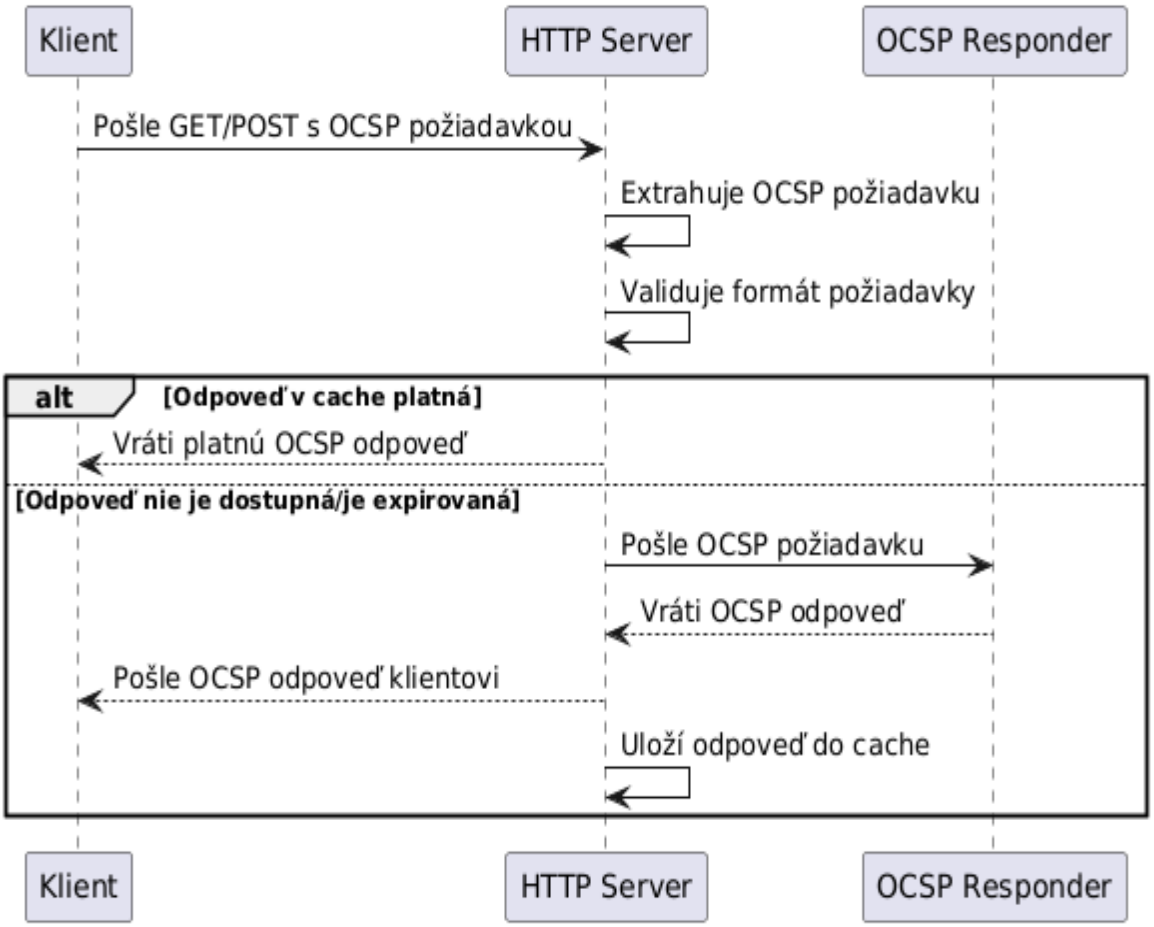
4. **Kontrola v cache**
   Extrahované certifikáty sa vyhľadajú v cache:

   - Ak je odpoveď v cache a je stále platná, vráti sa klientovi.
   - Ak požiadavka obsahuje `nonce`, alebo odpoveď nie je dostupná alebo je exspirovaná, pokračuje sa ďalšími krokmi.

5. **Odoslanie OCSP požiadavky na respondér**
   V prípade, že požiadavka nemôže byť vybraná z cache, server prepošle OCSP požiadavku OCSP respondérovi na spracovanie.

6. **Odoslanie odpovede klientovi**
   Server príjme odpoveď od OCSP respondéra a prepošle ju klientovi.

7. **Uloženie odpovede do cache**
   OCSP odpoveď je uložená do cache s príslušným časom exspirácie.



Obr. 25: Sekvenčný diagram spracovania OCSP požiadavky

4.2 Návrh cache

Cache mechanizmus bude implementovaný pomocou databázy Redis ako súčasť HTTP servera, ktorý bude spracovávať OCSP požiadavky. Hlavným cieľom cache je znížiť záťaž OCSP respondérov tým, že okrem uchovávania odpovedí na GET požiadavky bude uchovávať aj odpovede na POST požiadavky. Týmto spôsobom sa eliminuje potreba opakovaného dopytovania respondéra pri každej požiadavke na overenie sady viacerých certifikátov, čím sa zlepší výkon systému poskytovania informácií o revokácií certifikátov.

Architektúra cache

Cache bude implementovaná v Redis ako jednotná štruktúra, kde odpovede na GET a POST požiadavky budú rozlíšené pomocou prefixov kľúčov:

- **GET požiadavky**
  Odpoveď pre jednotlivý certifikát sa uloží kľúčom get:cert123.

- **POST požiadavky**
  Odpoveď pre sadu certifikátov sa uloží kľúčom post:certs123.

Pri ukladaní odpovedí sa bude zadávať doba platnosti záznamu individuálne podľa času exspirácie odpovede.

Manažment uchovávaných odpovedí

Redis poskytuje automatický mechanizmus na expiráciu záznamov pomocou doby platnosti záznamu, ktorý zabezpečí, že odpovede sa po uplynutí času platnosti odstránia z cache. Taktiež je možné nakonfigurovať LRU/LFU politiku na automatické odstraňovanie najmenej používaných odpovedí v prípade, že cache dosiahne maximálnu kapacitu.

4.3 Mechanizmus pre logovanie udalostí

Mechanizmus pre logovanie udalostí bude navrhnutý tak, aby poskytoval prehľad o fungovaní aplikácie a slúžil na diagnostiku chýb. Logovanie bude obsahovať nasledovné prvky:

- **Typy udalostí:** Logovať sa budú dôležité udalosti vrátane:
  - Prijatia HTTP požiadavky (GET/POST)
  - Interakcie s cache
  - Poslania požiadavky na OCSP respondér a prijatia odpovede
  - Chýb pri validácii OCSP požiadaviek alebo odpovedí
  - Údržby cache
- **Formát logov:** Každý záznam v logu bude obsahovať:
  - Dátum a čas udalosti,
  - Typ udalosti,
  - Popis udalosti (napr. ID certifikátu alebo sady certifikátov),
  - Úroveň závažnosti.

Logy budú zapisované do súboru, ktorého miesto uloženia bude konfigurovateľné. Takto navrhnutý systém logovania zabezpečí transparentnosť fungovania aplikácie a umožní efektívne riešenie problémov a optimalizáciu výkonu.

4.4 Príklad konfigurácie

Nižšie je uvedený príklad konfiguračného súboru, ktorý definuje pravidlá pre cache OCSP odpovedí a cestu pre ukladanie logovacích súborov.

```
{
  "cache": {
    "enabled": true,
    "max_ttl": 3600,
    "max_entries": 5000
  },
  log: {
    "path": ./logs/
  }
}
```

Popis konfiguračných parametrov:

- **cache**: Parametre pre cache.
- **enabled**: Povolenie alebo zakázanie cache mechanizmu.
- **ttl**: Doba, po ktorú je OCSP odpoveď platná v sekundách. Po uplynutí tejto doby bude odpoveď automaticky odstránená.
- **max_entries**: Maximálny počet odpovedí uložených v cache. Po prekročení tejto hodnoty budú najstaršie záznamy odstránené.
- **log**: Parametre pre logovanie, obsahujú cestu pre ukladanie logov.

Konfiguračný súbor poskytuje flexibilitu pri nastavovaní správania cache a logovania, čím umožňuje prispôsobenie aplikácie rôznym požiadavkám na výkon a údržbu.

5 Implementácia

5.1 Zmeny oproti návrhu

Počas realizácie sa vyskytlo niekoľko dôvodov, pre ktoré bolo potrebné mierne upraviť pôvodný návrh:

- **Rozšírenie konfigurácie**
  V pôvodnom návrhu sa uvažovalo so základným konfiguračným súborom so sekciami pre cache a logovanie. V implementácii sa však pridali ďalšie voliteľné parametre (napríklad `redis.port` a `redis.host`), ktoré umožňujú prispôsobenie pre rozličné prostredia bez nutnosti zmeny zdrojového kódu. Sekcia pre logovanie bola tiež odstránená, lebo na logovanie je použitý framework Logback, ktorý používa vlastný konfiguračný súbor `logback.xml`. Okrem toho sa konfiguračný súbor nepoužíva v JSON, ale vo formáte `.properties`, čo je bežný prístup pri vývoji v Jave.

- **Zmena kľúčov v cache**
  Pre jednoduchosť implementácie boli odstránené prefixy `get:` a `post:` pri tvorbe kľúča do cache. Ako kľúč sa používa len seriové číslo certifikátu, ak je v odpovedi len jeden certifikát, alebo konkatenácia sériových čísel certifikátov, ak je ich v odpovedi viac.

5.2 Opis tried

V nasledujúcej časti sú v krátkosti predstavené hlavné triedy a komponenty implementovaného riešenia. Pre lepšiu prehľadnosť sú triedy rozdelené podľa ich funkcionality a zodpovednosti v systéme. Detailný popis implementácie je v prílohe B.

Hlavné triedy

V tejto podkapitole je opis hlavných tried, ktoré implementujú kľúčové funkcionality aplikácie.

OCSPCacheServer

Táto trieda predstavuje vstupný bod aplikácie a zabezpečuje inicializáciu celého systému. V metóde `main` sa načítavajú konfiguračné parametre pomocou triedy `ConfigLoader`, inicializuje sa bazén pripojení pre `Redis` databázu a spúšťa sa servletový kontajner `Jetty`. Trieda koordinuje všetky komponenty a zabezpečuje ich správnu interakciu.

OCSPCacheServlet

Táto servletová trieda tvorí jadro aplikácie a spracováva všetky prichádzajúce OCSP požiadavky. Implementuje dve kľúčové metódy:

- Metóda **doGet** spracúva požiadavky zaslané cez HTTP GET. Extrahuje OCSP dopyt z URL cesty pomocou dekódovania base64url a následne ho transformuje na pole bajtov. Po validácii formátu požiadavky deleguje ďalšie spracovanie metóde `processOcspRequest`.
- Metóda **doPost** obsluhuje požiadavky zaslané cez HTTP POST. Načítava celé telo požiadavky, ktoré obsahuje OCSP dopyt vo formáte ASN.1 DER, a následne ho predáva metóde `processOcspRequest` na ďalšie spracovanie.

processOcspRequest

Táto metóda predstavuje centrálny bod spracovania OCSP požiadaviek. Jej funkcionalita zahŕňa:

1. Kontrolu prítomnosti náhodnej hodnoty `nonce` v požiadavke pomocou triedy OCSPRequestParser.
2. Ak požiadavka obsahuje `nonce`, je priamo preposlaná na OCSP respondér bez uloženia do cache.
3. Pre požiadavky bez `nonce` sa extrahujú sériové čísla certifikátov a vytvorí sa unikátny kľúč pre cache.
4. Následne sa vyhľadá odpoveď v `Redis` cache, a ak nie je nájdená alebo je exspirovaná, získa sa z OCSP respondéra.
5. Nová odpoveď sa uloží do cache s príslušnou dobou platnosti a odošle sa klientovi.

Ďalšie pomocné triedy a komponenty

Okrem hlavných tried, ktoré sa venujú kľúčovým funkcionalitám, boli v rámci implementácie vytvorené aj vedľajšie pomocné triedy pre menšie opakujúce sa úlohy:

ConfigLoader

Trieda zodpovedná za načítanie a správu konfiguračných parametrov aplikácie. Poskytuje metódy na získanie hodnôt z konfiguračného súboru `config.properties`, vrátane špecializovaných metód `getBoolean` a `getInt` pre konverziu na príslušné dátové typy. Táto trieda umožňuje flexibilnú konfiguráciu aplikácie bez nutnosti zmeny zdrojového kódu.

OCSPRequestParser

Špecializovaná trieda na analýzu a extrakciu dát z OCSP požiadaviek. Implementuje dve hlavné funkcionality:

- Metóda `hasNonce` detekuje prítomnosť náhodnej hodnoty v OCSP požiadavke.
- Metóda `parseOCSPRequest` na extrakciu zoznamu sériových čísel certifikátov z požiadavky, ktoré slúžia ako základ pre vytváranie kľúčov v cache.

OCSPResponseParser

Táto trieda zabezpečuje parsovanie a validáciu OCSP odpovedí. Extrahuje dôležité informácie ako čas platnosti odpovede, stav certifikátu a ďalšie relevantné údaje. Tieto informácie sú následne zaznamenané do logovacieho súboru pre účely monitorovania a diagnostiky. Okrem toho aj validuje podpis OCSP odpovedí.

Logger

Trieda zabezpečujúca jednotný systém logovania v celej aplikácii. Využíva framework Logback na zaznamenávanie rôznych typov udalostí s príslušnými úrovňami závažnosti. Táto trieda je kľúčová pre monitorovanie činnosti aplikácie a diagnostiku prípadných problémov.

6 Overenie riešenia

Táto kapitola sa venuje overeniu implementovaného riešenia pre zlepšenie efektivity OCSP komunikácie. Overenie zahŕňa kontrolu splnenia špecifikovaných požiadaviek, popis testovania funkcionality a celkové zhodnotenie dosiahnutých výsledkov. Cieľom je potvrdiť, že implementované riešenie spĺňa stanovené ciele a poskytuje očakávané prínosy v oblasti optimalizácie OCSP komunikácie.

6.1 Splnenie špecifikovaných požiadaviek

V tejto časti vyhodnocujeme, do akej miery implementované riešenie spĺňa požiadavky definované v kapitole 3. Pre každú požiadavku uvádzame spôsob jej implementácie a mieru splnenia.

Nie-funkčné požiadavky

Implementácia v jazyku Java

Požiadavka bola splnená. Celé riešenie bolo implementované v programovacom jazyku Java, čo zabezpečuje kompatibilitu s rôznymi operačnými systémami a umožňuje jednoduchú integráciu s existujúcimi riešeniami. Použitie Javy tiež poskytlo prístup k bohatým knižniciam pre prácu s HTTP a spracovanie OCSP požiadaviek.

Textové používateľské rozhranie

Požiadavka bola splnená. Aplikácia poskytuje textové rozhranie v príkazovom riadku, ktoré umožňuje jednoduché spustenie, monitorovanie činnosti servera a ukončenie.

Škálovateľnosť cache

Požiadavka bola splnená. Použitý databázový systém Redis štandardne neobmedzuje maximálny počet uložených záznamov. Jeho kapacita je limitovaná predovšetkým dostupnou operačnou pamäťou servera. Pre presnejšiu kontrolu využitia zdrojov však Redis umožňuje nastaviť maximálny pamäťový limit, ktorého nastavenie je súčasťou konfigurácie tohto riešenia. Navyše, systém podporuje politiky automatického odstraňovania záznamov, ktoré sa aktivujú pri dosiahnutí nakonfigurovanej maximálnej kapacity, čím zabezpečuje plynulú prevádzku aj pri vysokom počte ukladaných dát.

Konfigurácia systému

Požiadavka bola splnená. Aplikácia umožňuje konfiguráciu prostredníctvom súboru `config.properties`, ktorý obsahuje nastavenia pre maximálnu veľkosť cache, maximálnu dobu platnosti odpovedí a ďalšie parametre. Trieda `ConfigLoader` zabezpečuje načítanie a spracovanie týchto nastavení. Príklad obsahu konfiguračného súboru je v prílohe C.

Práca s chybovými stavmi

Požiadavka bola splnená. Riešenie obsahuje mechanizmy na detekciu a spracovanie chybových stavov. Kľúčové operácie a potenciálne chybové stavy sú ošetrené mechanizmami na zachytávanie výnimiek, aby aplikácia zostala stabilná. Chybové stavy sú zaznamenané v logovacom súbore s podrobnými informáciami o príčine.

Každý záznam v logu má konzistentnú štruktúru, ktorá začína presnou časovou pečiatkou a úrovňou závažnosti správy. Nasleduje identifikátor vlákna spracúvajúceho požiadavku, plný názov Java triedy, ktorá udalosť zaznamenala, a samotný popisný text správy o vykonanej akcii alebo udalosti.

```
2025-05-02 07:37:45 INFO  [main]
main.java.backend.OCSPCacheServer -
Spustam OCSP Cache Server na porte 8080...
```

Obr. 26: Príklad záznamu v logu.

Funkčné požiadavky

Prijímanie a spracovanie GET a POST OCSP požiadaviek

Požiadavka bola splnená implementáciou triedy OCSPCacheServlet s metódami doGet a doPost, ktoré spracúvajú príslušné HTTP požiadavky. Aplikácia úspešne validuje formát a obsah OCSP požiadaviek podľa štandardov OCSP protokolu.

Podpora náhodnej hodnoty v požiadavkách

Požiadavka bola splnená implementáciou triedy OCSPRequestParser, ktorá detekuje prítomnosť náhodnej hodnoty `nonce` v požiadavkách. Ak je táto hodnota prítomná, požiadavka je automaticky preposlaná na OCSP respondér bez použitia cache.

Komunikácia s OCSP respondérom

Požiadavka bola splnená. Aplikácia dokáže kontaktovať nakonfigurovaný OCSP respondér, získať aktuálnu odpoveď a uložiť ju do cache. Implementácia zahŕňa aj ošetrenie chybových stavov pri komunikácii s respondérom.

Cache odpovedí

Požiadavka na ukladanie odpovedí do cache bola splnená implementáciou Redis cache, ktorá umožňuje efektívne ukladanie odpovedí pre GET aj POST požiadavky. Na generovanie unikátnych kľúčov v cache systém využíva sériové čísla certifikátov, čo zaručuje korektné priradenie odpovedí k príslušným požiadavkám. Tento prístup navyše umožňuje, aby OCSP odpoveď uložená pre GET požiadavku bola použitá aj na obslúženie POST požiadavky a naopak, pokiaľ sa obe týkajú toho istého certifikátu.

Vypršanie platnosti odpovedí v cache

Požiadavka bola splnená využitím mechanizmu exspirácie záznamov v Redis databáze. Doba platnosti je nastavovaná individuálne pre každú odpoveď podľa času exspirácie definovaného v OCSP odpovedi.

Logovanie stavov aplikácie

Požiadavka bola splnená. Aplikácia využíva framework Logback na zaznamenávanie všetkých dôležitých udalostí vrátane prijatia požiadaviek, interakcií s cache, komunikácie s OCSP respondérom a prípadných chýb. Každý záznam obsahuje časovú pečiatku, triedu a úroveň závažnosti.

6.2 Testovanie priemerného času odozvy

Cieľom tohto testovania bolo overiť, či implementované riešenie prináša reálne zlepšenie efektivity OCSP komunikácie. Testovanie bolo navrhnuté tak, aby čo najviac simulovalo reálne podmienky používania a umožnilo čo najobjektívnejšie vyhodnotenie prínosu implementovaného riešenia.

Testovacie prostredie

Testovacie prostredie pozostávalo z troch hlavných komponentov:

- **Klientsky laptop** – zariadenie, ktoré posielalo OCSP požiadavky a meralo čas odozvy
- **Aplikačný server** – zariadenie, na ktorom bežal webový server Nginx a implementovaná cache aplikácia
- **OCSP respondér** – virtuálny stroj v `Oracle Cloud Infrastructure`, na ktorom bežal softvér `EJBCA` poskytujúci OCSP respondéra.

Klientsky laptop a aplikačný server boli pripojené k rovnakej lokálnej sieti prostredníctvom Wi-Fi pripojenia. OCSP respondér bol dostupný cez internet pomocou verejnej IP adresy a odhalených portov. Táto konfigurácia sa snaží napodobniť nasadenie, v ktorom klienti komunikujú s lokálnym serverom a ten následne so vzdialeným OCSP respondérom.

Metodika testovania času odozvy

Testovanie prebiehalo v troch konfiguračných režimoch:

1. **Bez cache** – vypnutá cache na Nginx serveri aj v implementovanej aplikácii
2. **Štandardná cache** – zapnutá cache GET požiadaviek na Nginx serveri, bez cache POST požiadaviek v implementovanej aplikácii

3. **Rozšírená cache** – zapnutá cache GET požiadaviek na Nginx serveri a zároveň cache POST požiadaviek v implementovanej aplikácii

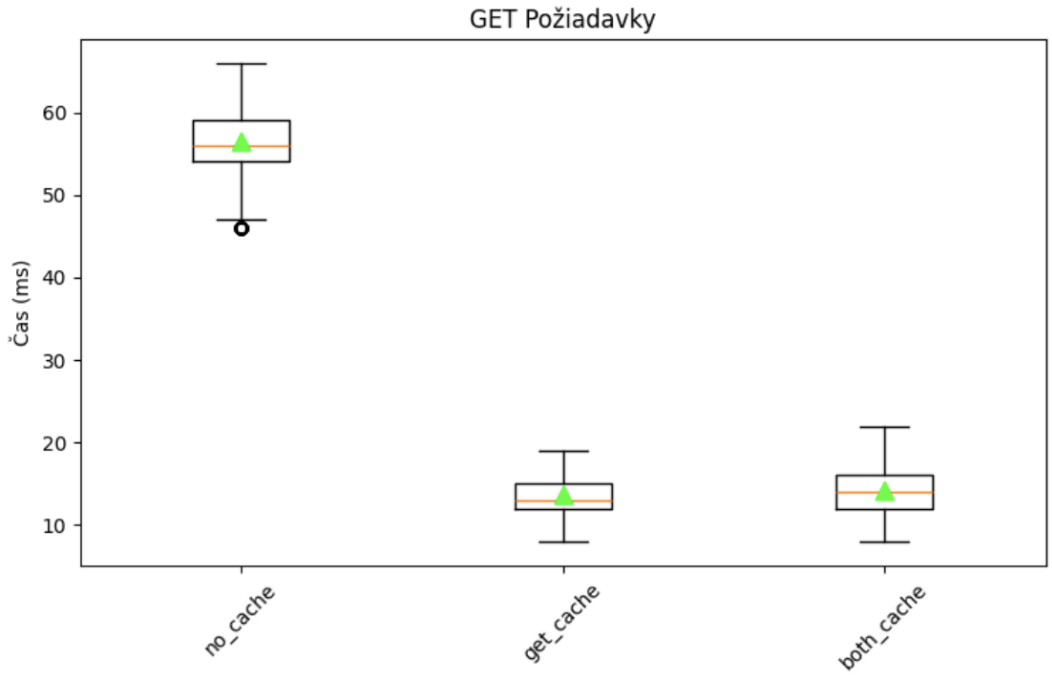
Pre každý režim bolo vykonaných 1000 požiadaviek, pričom sa striedali metódy GET a POST. Každá požiadavka bola zaznamenaná spolu s časom jej spracovania. Testy boli opakované v rôznych časoch dňa (6:00, 12:00, 18:00 a 24:00) s cieľom zahrnúť rôzne sieťové podmienky a zaťaženie infraštruktúry.

Výsledky testovania času odozvy

Pred zhodnotením výsledkov bola vykonaná prieskumná analýza dát, počas ktorej bola overená úplnosť dátových súborov a identifikované odľahlé hodnoty. Analýza potvrdila, že v dátach nechýbali žiadne hodnoty, čo zabezpečilo spoľahlivosť výsledkov. Zároveň boli identifikované niektoré výrazne odľahlé hodnoty, ktoré skreslovali celkové výsledky. Tieto hodnoty boli nahradené hraničnými hodnotami, čím sa zachovala štruktúra dát bez nežiaduceho vplyvu extrémnych hodnôt na celkovú analýzu.

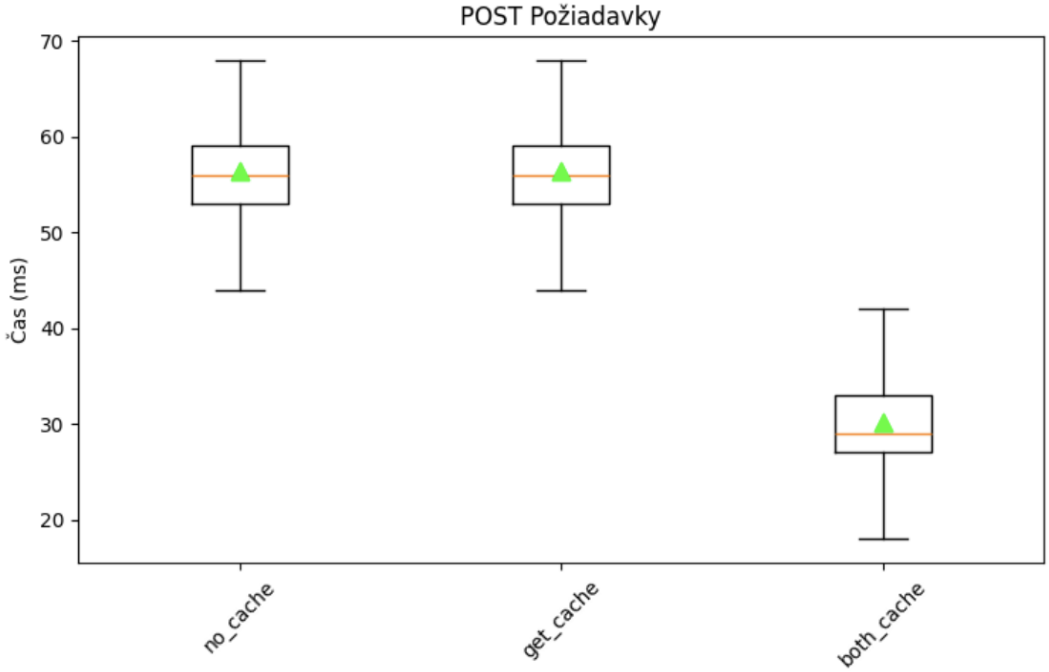
Všetky dáta zozbierané počas testovania sú uložené na priloženom digitálnom médiu, ktorého obsah je popísaný v prílohe D. Prieskumná analýza týchto dát, vrátane postupu na zabezpečenie jej reprodukovateľnosti, je detailne spracovaná a zdokumentovaná v Jupyter notebooku, ktorý je taktiež súčasťou digitálneho média.

Výsledky testovania sú vizualizované na grafoch uvedených nižšie, ktoré zobrazujú distribúciu časov odozvy pre GET a POST požiadavky v jednotlivých konfiguračných režimoch.



Obr. 27: Porovnanie časov odozvy GET požiadaviek pre rôzne konfigurácie

Z výsledkov testovania je zrejmé, že implementovanie cache výrazne znižuje priemerný čas odozvy pre GET požiadavky. V režime bez cache (no_cache) sa priemerný čas odozvy pohyboval okolo 56 ms. Po zapnutí cache pre GET požiadavky (get_cache) sa priemer znížil na približne 13.7 ms. Režim s oboma typmi cache (both_cache) vykazoval veľmi podobné výsledky pre GET požiadavky (priemer 14.2 ms), z čoho vyplýva, že cache POST požiadaviek nemá významný vplyv na priemerný čas spracovania GET požiadaviek.



Obr. 28: Porovnanie časov odozvy POST požiadaviek pre rôzne konfigurácie

Prínos implementovaného riešenia je výrazne viditeľný pri POST požiadavkách. V režime bez cache (no_cache) sa priemerný čas odozvy pohyboval okolo 56.3 ms. V režime so zapnutou cache len pre GET požiadavky (get_cache) zostal priemerný čas odozvy pre POST požiadavky takmer nezmenený, na úrovni približne 56.3 ms. Po zapnutí cache POST požiadaviek v implementovanej aplikácii (both_cache) sa priemerný čas odozvy výrazne znížil na približne 30.1 ms. To predstavuje zlepšenie o približne 46.5% oproti ostatným režimom.

6.3 Testovanie priepustnosti

Zatiaľ čo predchádzajúce testovanie sa zameriavalo na meranie času odozvy na jednotlivé požiadavky, pre komplexnejšie vyhodnotenie efektivity implementovaného riešenia bol navrhnutý aj záťažový test zameraný na priepustnosť systému. Cieľom bolo zistiť, koľko OCSP požiadaviek dokáže server spracovať za definovaný časový interval pri rôznych konfiguráciách cache.

Metodika testovania priepustnosti

Testovanie priepustnosti bolo vykonané v rovnakom prostredí ako testovanie reakčnej doby, no s odlišnou metodikou:

- Testované boli len dva konfiguračné režimy:
  - Bez cache – vypnutá cache na Nginx serveri aj v implementovanej aplikácii.
  - Rozšírená cache – vypnutá cache na Nginx serveri a zapnutá cache v implementovanej aplikácii.
- Pre každý režim bol spustený testovací skript, ktorý simuloval záťaž tým, že:
  - Paralelne posielal 4 POST požiadavky súčasne pomocou samostatných vlákien
  - Požiadavky boli zasielané kontinuálne po dobu 30 sekúnd
  - Cieľom bolo spracovanie maximálneho možného počtu požiadaviek za daný časový interval
- Skript priebežne ukladal všetky odpovede do dočasného súboru pre následnú analýzu HTTP hlavičiek a OCSP odpovedí.
- Po ukončení 30-sekundového intervalu skript analyzoval zozbierané dáta a výsledky pripojil do CSV súboru.
- Takýto test bol vykonaný v rovnakých časoch dňa ako v prvom testovacom scenári. V každom čase bol test pre daný konfiguračný režim vykonaný tri razy.

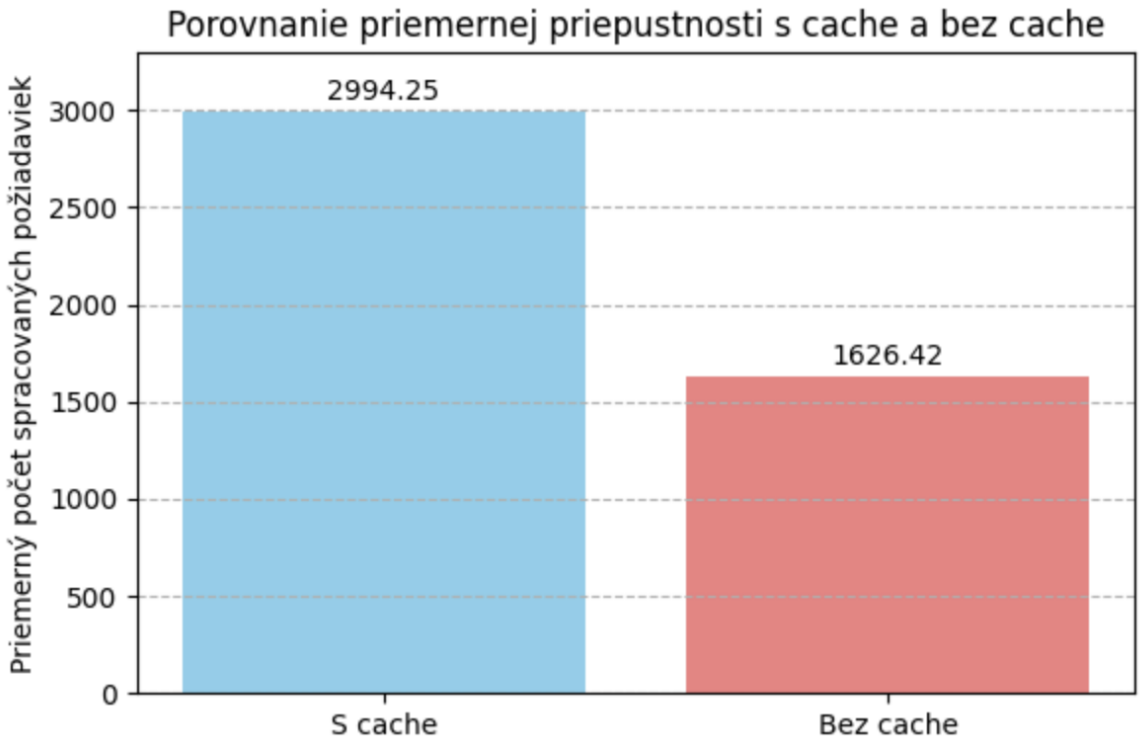
Výsledky testovania priepustnosti

Testovanie priepustnosti poskytlo detailný pohľad na schopnosť systému spracovávať väčšie množstvo požiadaviek v krátkom čase. Zozbierané dáta obsahovali informácie o:

- Celkovom počte pokusov
- Počte úspešných HTTP požiadaviek
- Počte spracovaných OCSP odpovedí
- Počte požiadaviek za minútu
- Skutočnej a cieľovej dobe trvania
- Počte vlákien a metóde požiadaviek

- Stave cache (zapnutá/vypnutá)

Kľúčovým ukazovateľom je porovnanie priemerného počtu požiadaviek, ktoré systém dokázal spracovať za jednotku času pri konfigurácii so zapnutou cache v porovnaní s konfiguráciou bez cache.



Obr. 29: Porovnanie priepustnosti systému pre rôzne konfigurácie cache

Vyššie uvedené výsledky testovania priepustnosti ukázali významný rozdiel v počte spracovaných OCSP odpovedí. V režime bez cache dokázal systém spracovať priemerne 1626 OCSP odpovedí. Pri zapnutej cache v implementovanej aplikácii systém spracoval priemerne 2994 OCSP odpovedí, čo predstavuje nárast o 84.1% viac ako bez cache. Podľa zaznamenaných údajov je teda viditeľné, že implementovaná rozšírená cache výrazne zvyšuje schopnosť systému spracovávať väčšie množstvo požiadaviek súčasne, čo potvrdzuje praktický prínos navrhnutého riešenia pre reálne nasadenie v prostredí s vysokou záťažou.

6.4 Zhodnotenie testovania

Overovanie implementovaného riešenia preukázalo, že navrhnutý systém pre cache OCSP komunikácie spĺňa všetky špecifikované požiadavky a prináša zlepšenie efektivity OCSP komunikácie. Testovanie potvrdilo nasledujúce prínosy riešenia:

- **Výrazné zníženie času odozvy**
  Implementovaná cache znížila priemerný čas odozvy POST požiadaviek o 46.5% v

porovnaní s režimom bez cache, čo predstavuje významné zlepšenie používateľskej skúsenosti a zníženie latencie pri overovaní certifikátov.

- **Zvýšenie priepustnosti**
  Ako ukázali testy priepustnosti, implementovaný cache server dokáže spracovať o približne 84.1% viac požiadaviek pri zapnutej cache, čo demonštruje jeho schopnosť efektívne obslúžiť vyššiu záťaž.

- **Konzistentné správanie v rôznych podmienkach**
  Testovanie v rôznych časoch dňa potvrdilo, že riešenie poskytuje stabilné a predvídateľné výsledky bez ohľadu na aktuálne sieťové podmienky.

Celkovo možno konštatovať, že implementované riešenie úspešne adresuje nedostatky štandardnej OCSP komunikácie a poskytuje efektívny mechanizmus na zlepšenie výkonu a spoľahlivosti OCSP komunikácie.

7 Zhodnotenie

Táto bakalárska práca sa zaoberala problematikou zlepšenia efektivity OCSP komunikácie prostredníctvom implementácie mechanizmu cache pre GET aj POST požiadavky. V rámci práce bola vykonaná podrobná analýza OCSP protokolu, jeho nedostatkov a existujúcich riešení, na základe ktorej bol navrhnutý a implementovaný vlastný systém pre optimalizáciu OCSP komunikácie.

Implementované riešenie úspešne adresuje hlavné nedostatky štandardnej OCSP komunikácie:

- **Zníženie latencie**
  Testovanie preukázalo významné zníženie času odozvy, kde pri POST požiadavkách došlo k zlepšeniu o 46.5% v porovnaní s režimom bez cache.

- **Zníženie záťaže a zvýšenie priepustnosti**
  Cache odpovedí výrazne redukuje počet požiadaviek smerovaných na OCSP respondérov, čím sa zvyšuje celková dostupnosť služby. Okrem toho testovanie ukázalo, že cache pre POST požiadavky zvyšuje priepustnosť o 84.1%

- **Flexibilita**
  Riešenie podporuje cache pre oba typy HTTP požiadaviek (GET aj POST) a umožňuje jednoduché prispôsobenie parametrov cache podľa konkrétnych potrieb nasadenia.

Významným prínosom práce je rozšírenie cache aj na POST požiadavky, čo nie je bežnou praxou v existujúcich webových serveroch. Toto vylepšenie prináša výrazné zlepšenie výkonu najmä v prípadoch, keď je potrebné overiť viacero certifikátov naraz, alebo keď klienti využívajú výhradne POST metódu pre OCSP komunikáciu.

Z pohľadu ďalšieho vývoja by bolo možné riešenie rozšíriť o:

- Integráciu s ďalšími existujúcimi vylepšeniami OCSP protokolu

- Pridanie podpory pre kompresiu odpovedí uložených v cache pre optimalizáciu využitia pamäte

- Optimalizácia tvorby kľúčov do cache tak, aby odpoveď vydaná pre viacero certifikátov mohla byť použitá aj pri požiadavke na jeden konkrétny certifikát, ak ho pôvodná odpoveď obsahuje.

- Pridanie podpory pre kontaktovanie viacerých OCSP respondérov

- Implementáciu používateľského rozhrania

Implementované riešenie prináša praktický prínos k zvýšeniu efektivity OCSP komunikácie. Overenie riešenia preukázalo, že navrhovaný optimalizačný prístup je účinný a prináša merateľné zlepšenia.

Literatúra

[1] Diana Gratiela Berbecaru and Antonio Lioy. An evaluation of x.509 certificate revocation and related privacy issues in the web pki ecosystem. *IEEE Access*, 11:79156–79170, 2023. doi: 10.1109/ACCESS.2023.3299357. URL `https://ieeexplore.ieee.org/document/10196098`.

[2] Paul Danquah and Henoch Kwabena-Adade. Public key infrastructure: An enhanced validation framework. *Journal of Information Security*, 11:241–260, 2020. doi: 10.4236/jis.2020.114016. URL `https://www.scirp.org/journal/paperinformation.aspx?paperid=103117`.

[3] Reham M. Abobeah, Mohamed M. Ezz, and Hany M. Harb. Public-key cryptography techniques evaluation. *International Journal of Computer Networks and Applications*, 2:64–75, 2015. ISSN 2395-0455.

[4] Abylay Satybaldy. *Towards Self-Sovereign Identity*. Doctoral thesis, Norwegian University of Science and Technology (NTNU), 2024.

[5] International Telecommunication Union. Itu-t x.509: Information technology - open systems interconnection - the directory: Public-key and attribute certificate frameworks. Technical Report X.509, International Telecommunication Union (ITU), 2023. URL `https://www.itu.int/rec/T-REC-X.509/en`.

[6] International Telecommunication Union. Itu-t recommendation x.690: Specification of asn.1 encoding rules – basic encoding rules (ber), canonical encoding rules (cer) and distinguished encoding rules (der). Technical Report X.690, International Telecommunication Union (ITU), 2021. URL `https://www.itu.int/rec/T-REC-X.690-202102-I/en`.

[7] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile. Technical Report 5280, Internet Engineering Task Force (IETF), May 2008. URL `https://www.rfc-editor.org/rfc/rfc5280`.

[8] S. Santesson, M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. X.509 internet public key infrastructure online certificate status protocol - ocsp. Technical Report 6960, Internet Engineering Task Force (IETF), june 2013. URL https://www.rfc-editor.org/rfc/rfc6960.

[9] Asier Atutxa, Jasone Astorga, Marc Barcelo, Aitor Urbieta, and Eduardo Jacob. Improving efficiency and security of iiot communications using in-network validation of server certificate. *Computers in Industry*, 144:103802, 2023. ISSN 0166-3615. doi: https://doi.org/10.1016/j.compind.2022.103802. URL https://www.sciencedirect.com/science/article/pii/S0166361522001981.

[10] Ahmad Samer Wazan, Romain Laborde, David W. Chadwick, Remi Venant, Abdelmalek Benzekri, Eddie Billoir, and Omar Alfandi. On the validation of web x.509 certificates by tls interception products. *IEEE Transactions on Dependable and Secure Computing*, 19(1):227–242, 2022. doi: 10.1109/TDSC.2020.3000595.

[11] T Smith, L Dickinson, and K Seamons. Let's revoke: Scalable global certificate revocation. In *Network and Distributed Systems Security (NDSS) Symposium*. NDSS, 2020. URL https://par.nsf.gov/servlets/purl/10175072.

[12] Jens Friess, Haya Schulmann, and Michael Waidner. Revocation speedrun: How the webpki copes with fraudulent certificates. *Proc. ACM Netw.*, 1(CoNEXT3), November 2023. doi: 10.1145/3629148. URL https://doi.org/10.1145/3629148.

[13] Yitayal K Ebrahim. Security analysis of website certificate validation, 2020. URL https://www.researchgate.net/publication/341509518_Security_analysis_of_website_certificate_validation.

[14] Matteo Simone. Certificate validation and tls interception. Master's thesis, Politecnico di Torino, Turin, Italy, 2021-2022.

[15] Salabat Khan, Fei Luo, Zijian Zhang, Farhan Ullah, Farhan Amin, Syed Furqan Qadri, Md Belal Bin Heyat, Rukhsana Ruby, Lu Wang, Shamsher Ullah, Meng Li, Victor C. M. Leung, and Kaishun Wu. A survey on x.509 public-key infrastructure, certificate revocation, and their modern implementation on blockchain and ledger technologies. *IEEE Communications Surveys & Tutorials*, 25(4):2529–2568, 2023. doi: 10.1109/COMST.2023.3323640.

[16] Taejoong Chung, Jay Lok, Balakrishnan Chandrasekaran, David Choffnes, Dave Levin, Bruce M. Maggs, Alan Mislove, John Rula, Nick Sullivan, and Christo Wilson. Is the web ready for ocsp must-staple? In *Proceedings of the Internet*

Measurement Conference 2018, pages 105–118. ACM, Association for Computing Machinery, 2018. ISBN 9781450356190. doi: 10.1145/3278532.3278543. URL https://doi.org/10.1145/3278532.3278543.

[17] Maryam Zulfiqar, Muhammad Umar Janjua, Muhammad Hassan, Talha Ahmad, Tania Saleem, and Jack Stokes. Tracking adoption of revocation and cryptographic features in x.509 certificates. *International Journal of Information Security*, 21: 653–668, 2021.

[18] M Svitek. Investigating certificate revocation options. Master's thesis, Masaryk University, 2023. URL https://is.muni.cz/th/fpwns/thesis_Archive.pdf.

[19] Y Pettersen. The transport layer security (tls) multiple certificate status request extension. Request for Comments 6961, June 2013. URL https://www.rfc-editor.org/rfc/rfc6961.

[20] Antonios A. Chariton, Eirini Degleri, Panagiotis Papadopoulos, Panagiotis Ilia, and Evangelos P. Markatos. Ccsp: a compressed certificate status protocol. In *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, University of Crete, Greece and FORTH-ICS, Greece, 2017. IEEE.

[21] Joel Höglund, Martin Furuhed, and Shahid Raza. Lightweight certificate revocation for low-power iot with end-to-end security. *Journal of Information Security and Applications*, 73:103424, 2023. doi: 10.1016/j.jisa.2023.103424. URL https://doi.org/10.1016/j.jisa.2023.103424.

[22] E. Stark, L.S. Huang, D. Israni, C. Jackson, and D. Boneh. The case for prefetching and prevalidating tls server certificates. In *To appear in Proceedings of Network & Distributed System Security (NDSS)*, 2012. URL http://www.truststc.org/pubs/842.html.

[23] Ibrahim Mahmood Ibrahim, Siddeeq Y. Ameen, Hajar Maseeh Yasin, Naaman Omar, Shakir Fattah Kak, Zryan Najat Rashid, Azar Abid Salih, Nareen O. M. Salim, and Dindar Mikaeel Ahmed. Web server performance improvement using dynamic load balancing techniques: A review. *Asian Journal of Research in Computer Science*, 10(1):47–62, 2021. doi: 10.9734/ajrcos/2021/v10i130234. URL https://journalajrcos.com/index.php/AJRCOS/article/view/188.

[24] Hong-Sheng Huang, Cheng-Che Chuang, Jhih-Zen Shih, Hsuan-Tung Chen, and Hung-Min Sun. An enhanced online certificate status protocol for public key infrastructure with smart grid and energy storage system, 2024. URL https://arxiv.org/abs/2409.10929.

[25] Guruprasad Nookala. The role of ssl/tls in securing api communications: Strategies for effective implementation. *Journal of Computing and Information Technology*, 4(1), 2024.

[26] R. Fielding, M. Nottingham, and J. Reschke. Http semantics. Request for Comments 9110, 2022. URL `https://datatracker.ietf.org/doc/html/rfc9110`.

Príloha A: Plán práce

V tejto prílohe sú v tabuľkách uvedené plány práce rozdelené na zimný a letný semester a je pri nich uvedené ako sa ich darilo plniť.

Zimný semester

Tabuľka A.1: Plán na zimný semester pre bakalársku prácu

Týždeň	**Plán**
1-4	Analýza problematiky OCSP protokolu
5-7	Analýza existujúcich vylepšení OCSP protokolu
8-10	Návrh vlastného vylepšenia
11-12	Dokončenie dokumentu

Plán na zimný semester bol z väčšej časti dodržaný, avšak dokončenie dokumentu trvalo dlhšie, než bolo pôvodne plánované. Dokument bol dokončený až po skončení semestra.

Letný semester

Tabuľka A.2: Plán na letný semester pre bakalársku prácu

Týždeň	**Plán**
1-6	Implementácia riešenia
7-8	Testovanie riešenia
9-10	Spísanie implementácie a overenia riešenia
11-12	Dokončenie dokumentu a elektronického média

Plán práce pre letný semester bol taktiež vo veľkej miere dodržaný, s výnimkou predĺženia fázy implementácie a testovania riešenia. Toto oneskorenie bolo spôsobené priebežným vylepšovaním programu, keďže som mal snahu dosiahnuť čo najvyššiu kvalitu výsledného riešenia. Každá úprava kódu si však vyžadovala opakované testovanie, čo predĺžilo celý proces.

Príloha B: Technická dokumentácia

V tejto prílohe sú opísané kľúčové triedy implementovaného riešenia, ich významné atribúty, metódy, parametre a návratové hodnoty. Zameriava sa na technické aspekty implementácie, ktoré dopĺňajú všeobecnejší opis riešenia uvedený v hlavnej časti práce.

OCSPCacheServer

Hlavná trieda aplikácie, ktorá zodpovedá za inicializáciu, konfiguráciu a spustenie OCSP cache servera. Koordinuje činnosť jednotlivých komponentov, ako sú HTTP server, cache mechanizmus a logika pre spracovanie OCSP požiadaviek.

Atribúty

Medzi hlavné atribúty patria:

- **CACHE_ENABLED** – `boolean` – Určuje, či je cache pre OCSP odpovede zapnutá. Načítava sa z konfiguračného súboru.
- **CACHE_TTL** – `int` – Predvolená doba platnosti pre záznamy v cache v sekundách. Použije sa, ak nie je možné určiť TTL z OCSP odpovede. Načítava sa z konfiguračného súboru.
- **CACHE_MAX_SIZE** – `int` – Maximálna veľkosť pamäte v bajtoch, ktorú môže Redis použiť pre cache. Načítava sa z konfiguračného súboru.
- **CACHE_SERVER_PORT** – `int` – Port, na ktorom OCSP cache server počúva prichádzajúce požiadavky. Načítava sa z konfiguračného súboru.
- **RESPONDER_URL** – `String` – URL adresa OCSP respondéra, na ktorý sa preposielajú požiadavky, ak nie sú nájdené v cache. Načítava sa z konfiguračného súboru.
- **logger** – Inštancia loggera pre zaznamenávanie udalostí a chýb v triede OCSPCacheServer.
- **REDIS_HOST** – `String` – Adresa Redis servera. Načítava sa z konfiguračného súboru.
- **REDIS_PORT** – `int` – Port Redis servera. Načítava sa z konfiguračného súboru.

- **SERVER_MIN_THREADS, SERVER_MAX_THREADS, SERVER_THREAD_TIMEOUT** – `int` – Konfiguračné parametre pre bazén vlákien servera, určujúce minimálny a maximálny počet vlákien a ich časový limit nečinnosti.

- **jedisPool** – Statická inštancia bazéna pripojení k Redis databáze, zdieľaná v rámci aplikácie. Takisto ako samotný server, aj tento bazén má implmentovanú konfiguráciu parametrov cez konfiguračný súbor.

Metódy

- `public static void main(String[] args)`

  – Spúšťa celú aplikáciu. Inicializuje bazén pripojení pre Redis s konfiguračnými parametrami načítanými cez `ConfigLoader`. Overuje pripojenie k Redis serveru pomocou príkazu `ping` a nastavuje maximálnu veľkosť pamäte pre Redis. Slúži aj na korektné uvoľnenie zdrojov pri ukončení aplikácie. Nakoniec volá metódu `startServer()` na spustenie HTTP servera.

- `private static void startServer()`

  – Inicializuje a spúšťa embedovaný Jetty HTTP server. Konfiguruje bazén vlákien pre server s parametrami načítanými z konfiguračného súboru. Vytvára `ServerConnector` na definovanom porte `CACHE_SERVER_PORT` a registruje `OCSPCacheServlet` na obsluhu požiadaviek prichádzajúcich na cestu `/ocsp/*`.

OCSPCacheServer.OCSPCacheServlet

Vnútorná statická trieda, ktorá tvorí jadro logiky pre spracovanie HTTP GET a POST požiadaviek obsahujúcich OCSP dopyty.

Metódy

- `protected void doPost(HttpServletRequest request, HttpServletResponse response)`

  – Spracováva HTTP POST požiadavky. Načíta telo požiadavky a následne deleguje spracovanie na metódu `processOcspRequest`.

- `protected void doGet(HttpServletRequest request, HttpServletResponse response)`

  - Spracováva HTTP GET požiadavky. OCSP request dáta sú očakávané v URL ceste za `/ocsp/`, zakódované pomocou Base64 kódovania. Po extrakcii a dekódovaní dát deleguje spracovanie na metódu `processOcspRequest`. V prípade chýbajúceho alebo nesprávne zakódovaného requestu odošle klientovi chybovú HTTP odpoveď.

- `private void processOcspRequest(byte[] ocspRequestData, HttpServletResponse response)`

  - Centrálna metóda pre spracovanie OCSP požiadavky. Najprv pomocou skontroluje prítomnosť náhodnej hodnoty `nonce` v požiadavke. Ak je prítomná a cacheje zapnutá, požiadavka sa priamo prepošle na OCSP respondér bez kontroly cache. V opačnom prípade sa extrahujú sériové čísla certifikátov z požiadavky, z ktorých sa vytvorí kľúč pre cache. Ak je cache zapnutá, pokúsi sa načítať odpoveď z cache. Ak odpoveď v cache nie je nájdená, alebo je cachovanie vypnuté, zavolá sa metóda na získanie odpovede od OCSP respondéra. Získaná odpoveď (buď z cache alebo od respondéra) sa následne analyzuje pre určenie jej doby platnosti (`TTL`). Nakoniec sa OCSP odpoveď odošle klientovi s príslušnými HTTP hlavičkami.

- `private byte[] fetchFromResponder(byte[] ocspRequestData)`

  - Zabezpečuje komunikáciu s OCSP respondérom definovaným v `RESPONDER_URL`. Vytvorí s ním HTTP spojenie, nastaví metódu na POST, pridá potrebné HTTP hlavičky a odošle požiadavku od klienta. Načíta odpoveď od respondéra a vráti ju ako pole bajtov. Loguje HTTP status kód odpovede a prípadné chyby.

ConfigLoader

Pomocná trieda zodpovedná za načítavanie konfiguračných parametrov aplikácie zo súboru `config.properties`. Ak súbor neexistuje alebo sa nepodarí načítať, použijú sa predvolené hodnoty.

Atribúty

- `CONFIG_FILE` – `String` – Statická konštanta definujúca cestu ku konfiguračnému súboru.

- `properties` – Statická inštancia triedy `Properties`, ktorá uchováva načítané konfiguračné hodnoty.

Metódy

- `static {}` (Statický inicializačný blok)
  - Pri prvom načítaní triedy sa pokúsi načítať obsah konfiguračného súboru do inštancie `properties`. V prípade chyby pri načítaní vypíše chybovú hlášku na štandardný chybový výstup, ale aplikácia pokračuje s predvolenými hodnotami.
- `public static String get(String key, String defaultValue)`
  - Vráti hodnotu konfiguračného parametra ako reťazec.
- `public static int getInt(String key, int defaultValue)`
  - Vráti hodnotu konfiguračného parametra ako celé číslo.
- `public static boolean getBoolean(String key, boolean defaultValue)`
  - Vráti hodnotu konfiguračného parametra ako boolovskú hodnotu.

OCSPRequestParser

Pomocná trieda zodpovedná za parsovanie binárnych OCSP požiadaviek zakódovaných v ASN.1 DER formáte. Používa knižnicu `BouncyCastle` na dekódovanie štruktúry OCSP požiadavky.

Metódy

- `public static List<String> parseOCSPRequest(byte[] ocspRequestData)`
  - Parsuje OCSP požiadavku a extrahuje sériové čísla všetkých certifikátov, pre ktoré sa žiada status.
- `public static boolean hasNonce(byte[] ocspRequestData)`
  - Kontroluje, či daná OCSP požiadavka obsahuje rozšírenie `nonce`.

OCSPResponseParser

Pomocná trieda zodpovedná za parsovanie binárnych OCSP odpovedí a overovanie ich digitálneho podpisu. Využíva knižnicu `BouncyCastle` na prácu s OCSP štruktúrami a `Java Cryptography API` na prácu s podpismi a certifikátmi.

Metódy

- `public static void parseOCSPResponse(byte[] ocspResponseBytes)`
  - Spracuje OCSP odpoveď. Dekóduje základnú OCSP odpoveď a ak je jej status úspešný, iteruje cez jednotlivé `SingleResp` objekty. Pre každý z nich loguje sériové číslo certifikátu, jeho stav, čas aktualizácie a čas nasledujúcej aktualizácie. Následne načíta certifikát OCSP respondéra a pomocou jeho verejného kľúča overí platnosť digitálneho podpisu OCSP odpovede volaním metódy `verifyOCSPSignature`.
- `private static boolean verifyOCSPSignature(BasicOCSPResp response, PublicKey publicKey)`
  - Overuje digitálny podpis OCSP odpovede pomocou poskytnutého verejného kľúča. Používa podpisový algoritmus `SHA256withRSA`.
- `private static String getResponseStatusText(int status)`
  - Konvertuje číselný status OCSP odpovede na čitateľný textový reťazec.
- `private static String getCertStatusText(Object certStatus)`
  - Konvertuje objekt reprezentujúci stav certifikátu na čitateľný textový reťazec.
- `public static Date getNextUpdate(byte[] ocspResponseBytes)`
  - Parsuje OCSP odpoveď a extrahuje z nej hodnotu nasledujúcej aktualizácie. Táto hodnota indikuje, dokedy je informácia v odpovedi považovaná za aktuálnu.

PathLoader

Pomocná trieda poskytujúca statické metódy na načítavanie kryptografických kľúčov a X.509 certifikátov zo súborov v PEM alebo DER formáte.

Metódy

- `public static X509Certificate loadCertificate(String certPath)`
  - Načíta X.509 certifikát zo súboru v DER alebo PEM formáte.
- `private static String readKeyFile(String path)`
  - Pomocná metóda na načítanie celého obsahu textového súboru do jedného reťazca.

Príloha C: Používateľská príručka

Táto príloha poskytuje základné informácie pre nasadenie a konfiguráciu OCSP Cache Servera. Aplikácia je navrhnutá tak, aby bežala ako serverový proces na pozadí a jej primárna interakcia s používateľom prebieha prostredníctvom konfiguračného súboru.

Pred spustením

1. **Java Runtime Environment:** Uistite sa, že na systéme, kde bude server bežať, je nainštalovaná Java verzia 17 alebo novšia.

2. **Redis Server:** Aplikácia vyžaduje bežiaci Redis server pre ukladanie OCSP odpovedí do cache. Uistite sa, že Redis server je dostupný a správne nakonfigurovaný.

3. **Konfiguračný súbor:** Pripravte si konfiguračný súbor s názvom `config.properties` v rovnakom adresári, z ktorého budete spúšťať aplikáciu.

Konfigurácia systému

Správanie OCSP Cache Servera sa riadi prostredníctvom konfiguračného súboru `config.properties`. Tento súbor obsahuje páry kľúč-hodnota, ktoré definujú rôzne aspekty servera. Nižšie sú uvedené kľúčové konfiguračné parametre:

- `server.port`: Číselná hodnota portu, na ktorom bude OCSP Cache Server počúvať prichádzajúce HTTP požiadavky. Predvolená hodnota je `8080`.

- `ocsp.responder.url`: Plná URL adresa OCSP respondéra, na ktorý budú preposielané požiadavky, ktoré sa nenájdu v cache.

- `server.cache.enabled`: Booleovská hodnota (`true` alebo `false`), ktorá určuje, či je mechanizmus cache aktívny. Predvolená hodnota je `true`.

- `server.cache.ttl`: Celé číslo reprezentujúce predvolenú dobu platnosti záznamov v cache v sekundách. Táto hodnota sa použije, ak nie je možné určiť TTL priamo z OCSP odpovede. Predvolená hodnota je `3600`.

- `redis.host`: Názov hostiteľa alebo IP adresa Redis servera. Predvolená hodnota je `localhost`.

- `redis.port`: Číselná hodnota portu, na ktorom Redis server počúva. Predvolená hodnota je `6379`.

- `redis.max.size`: Celé číslo reprezentujúce maximálnu veľkosť pamäte v bajtoch, ktorú môže Redis alokovať pre dáta cache. Napríklad `10000000` pre 10MB.
- `redis.threads.max.total`: Maximálny celkový počet pripojení v bazéne pripojení pre Redis.
- `redis.threads.max.idle`: Maximálny počet nečinných pripojení v bazéne pripojení pre Redis.
- `redis.threads.min.idle`: Minimálny počet nečinných pripojení v bazéne pripojení pre Redis.
- `server.threads.min`: Minimálny počet vlákien v bazéne pre server.
- `server.threads.max`: Maximálny počet vlákien v bazéne pre server.
- `server.threads.idleTimeout`: Čas v milisekundách, po ktorom bude nečinné vlákno v bazéne pre server ukončené.
- `ca.cert`: Cesta k súboru s certifikátom CA, ktorý sa použije na overenie podpisu OCSP odpovedí. Tento parameter je dôležitý pre zabezpečenie integrity odpovedí.

Ukážka obsahu súboru `config.properties` použitého pri testovaní riešenia:

```
    #######   Konfiguracia OCSPCacheServer   #######

server.port=8080
server.threads.min=50
server.threads.max=1024
server.cache.enabled=true
server.cache.ttl=3600
server.threads.idleTimeout=10000

         #   Konfiguracia ostatnych sluzieb   #

redis.host=host.docker.internal
redis.port=6379
redis.threads.max.total=1024
redis.threads.max.idle=20
redis.threads.min.idle=4
ca.cert=/app/ca.pem
ocsp.responder.url=http://89.168.80.177:9000/ocsp
```

Obr. 30: Príklad konfigurácie config.properties.

Upravte hodnoty podľa potrieb vášho prostredia.

Konfigurácia logovania

Aplikácia využíva framework Logback pre pokročilé logovanie. Konfigurácia Logbacku sa nachádza v osobitnom súbore `logback.xml`, ktorý musí byť umiestnený v koreňovom adresári aplikácie. Tento súbor umožňuje detailne nastaviť, ako a kam sa budú logovacie správy ukladať.

Nižšie je príklad obsahu súboru `logback.xml`, ktorý bol použitý pri testovaní riešenia. Zdôvodu prehľadnosti zobrazenia sú niektoré konkrétne hodnoty nahradené tromi bodkami.

```
<configuration>
  <appender name="FILE" class="...RollingFileAppender">
    <file>logs/server.log</file>
    <rollingPolicy class="...TimeBasedRollingPolicy">
      <fileNamePattern>...</fileNamePattern>
      <maxHistory>30</maxHistory>
    </rollingPolicy>

    <encoder>
      <pattern>%d{yyyy-MM-dd HH:mm:ss}...</pattern>
    </encoder>
  </appender>

  <appender name="CONSOLE" class="...ConsoleAppender">
    <encoder>
       <pattern>%d{yyyy-MM-dd HH:mm:ss}.. </pattern>
    </encoder>
  </appender>

  <logger name="org.eclipse.jetty" level="WARN" />
  <root level="INFO">
    <appender-ref ref="FILE" />
    <appender-ref ref="CONSOLE" />
  </root>
</configuration>
```

Obr. 31: Príklad konfigurácie logback.xml.

Kľúčové časti konfigurácie `logback.xml`:

- **Appender**: Definuje cieľ pre logovacie správy. V predvolenom nastavení sú definované dva:
  - `FILE`: Zapisuje logy do súboru. Používa `TimeBasedRollingPolicy`, čo znamená, že logy sa budú automaticky archivovať (rotovať) na dennej báze a staré logy budú uchovávané po dobu definovanú v `maxHistory` (napr. 30 dní).
  - `CONSOLE`: Zapisuje logy na štandardný konzolový výstup.
- **Encoder**: Pre každý appender definuje formát logovacej správy. V predvolenom nastavení obsahuje každý logovací záznam dátum a čas, úroveň logovania, názov vlákna, názov triedy a samotnú správu.
- **Logger**: Umožňuje nastaviť špecifickú úroveň logovania pre konkrétne balíky alebo triedy. V predvolenom nastavení je pre balík `org.eclipse.jetty` nastavená úroveň `WARN`, aby sa znížil objem menej dôležitých logov z Jetty servera.
- **Root**: Predstavuje koreňový logger, ktorý zachytáva všetky logovacie správy, pre ktoré nie je definovaný špecifickejší logger. V predvolenom nastavení je nastavený na úroveň `INFO` a odosiela správy do appenderov `FILE` aj `CONSOLE`.

Používateľ môže tento súbor upraviť podľa svojich potrieb, napríklad zmeniť cestu k logovacím súborom, formát logov, politiku rotácie alebo úrovne logovania pre rôzne časti aplikácie.

Monitorovanie a ukončenie

- **Monitorovanie:** Činnosť servera je možné sledovať prostredníctvom logovacích výstupov. Logy obsahujú informácie o prijatých požiadavkách, interakciách s cache, komunikácii s OCSP respondérom a prípadných chybových stavoch.
- **Ukončenie:** Server je možné štandardne ukončiť signálom prerušenia. Aplikácia automaticky zabezpečí korektné uvoľnenie zdrojov.

Príloha D: Obsah digitálneho média

Evidenčné číslo práce: FIIT-16768-120762

Spolu s týmto dokumentom sa odovzdáva aj digitálne médium vo forme zip archívu. Táto príloha popisuje jeho obsah.

Adresáre:

- **Document** – adresár s latex artefaktmi
- **Implementation** – projektový adresár
- **Testing** – adresár s dátami z testovania a testovacími skriptami

Súbory:

- **OCSPCacheServer.jar** – spustiteľný súbor s aplikáciou
- **bp.ipynb** – Súbor s prieskumnou analýzou testovacích dát vo formáte `Jupyter notebook`
- **BP_PeterBrenkus.pdf** – bakalárska práca aj s textovými prílohami