Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky

Zabezpečená komunikácia klient server s využitím post-kvantových algoritmov

Diplomová práca
(revízia 2, 13.05.2024)

2024 **Bc. Jozef Šimko**

Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky

Zabezpečená komunikácia klient server s využitím post-kvantových algoritmov

Diplomová práca
(revízia 2, 13.05.2024)

Študijný program:	Počítačové siete
Študijný odbor:	Informatika
Školiace pracovisko:	Katedra elektroniky a multimediálnych telekomunikácií (KEMT)
Školiteľ:	prof. Ing. Miloš Drutarovský, CSc.
Konzultant:	-

Košice 2024 **Bc. Jozef Šimko**

Abstrakt v SJ

Táto diplomová práca sa venuje použitiu post-kvantovej kryptografie na zabezpečenie štandardnej TCP/IP komunikácie na báze architektúry klient-server prostredníctvom TLS 1.3 protokolu. Cieľom teoretickej časti je oboznámiť čitateľa s bezpečnostným rizikom, ktoré prináša rozvoj kvantových počítačov, informovať ho o význame, matematických smeroch a celom procese NIST štandardizácie post-kvantovej kryptografie, opísať zmeny TLS 1.3 spojené s ich integráciou do protokolu a priblížiť základné princípy predbežne štandardizovaných algoritmov. Praktická časť obsahuje postup riešenia zadania, použité softvérové nástroje a zaoberá sa knižnicami a aplikáciami, ktoré umožňujú experimentálne testovať post-kvantové kryptografické algoritmy pri TCP/IP komunikácii klient-server na báze TLS. Zároveň opisuje implementáciu post-kvantových algoritmov do aplikácie klienta v jazyku C/C++ s využitím prekladača GCC, ktorá je kompatibilná s OS Windows aj OS Linux. V závere práce sú opísané realizované pripojenia k testovaciemu serveru a výsledky experimentálnych meraní časov generovania kľúčov a overovania podpisov vybraných algoritmov post-kvantovej kryptografie.

Kľúčové slová v SJ

architektúra klient-server, autentizácia, certifikačná autorita, CRYSTALS-Dilithium, CRYSTALS-Kyber, enkapsulácia kľúčov, jazyk C, kvantové počítače, liboqs, ML-DSA, ML-KEM, NIST, OpenSSL, oqs-provider, post-kvantová kryptografia, prekladač gcc, TCP/IP, TLS, výmena kľúčov

Abstrakt v AJ

This Master's thesis describes the use of post-quantum cryptography to secure standard TCP/IP communication based on a client-server architecture using the TLS 1.3 protocol. The aim of the theoretical part of the thesis is to introduce quantum computers and their impact on security, to inform about meaning, mathematical directions and the whole process of NIST standardization of post-quantum cryptography, to describe the TLS 1.3 changes associated with their integration into the protocol and to present the basic principles of the pre-standardized algorithms. The practical part analyses the solution procedure, used software tools and includes libraries and applications that allow experimental testing of post-quantum cryptographic algorithms in TLS-based TCP/IP client-server communication. Also, it takes a look on the implementation of the post-quantum algorithms in a C/C++ client application using the GCC compiler, which is compatible with both OS Windows and OS Linux. Last but not least, this part describes connection to the public test server and the results of experimental time measurements of key generation and signature verification of selected post-quantum cryptography algorithms.

Kľúčové slová v AJ

authentication, certificate authority, client-server, CRYSTALS-Dilithium, CRYSTALS-Kyber, GCC compiler, key encapsulation, key exchange, language C, liboqs, ML-DSA, ML-KEM, NIST, OpenSSL, oqs-provider, post-quantum cryptography, quantum computers, TCP/IP, TLS

Bibliografická citácia

ŠIMKO, Bc. Jozef. *Zabezpečená komunikácia klient server s využitím post-kvantových algoritmov*. Košice: Technická univerzita v Košiciach, Fakulta elektrotechniky a informatiky, 2024. 102s. Vedúci práce: prof. Ing. Miloš Drutarovský, CSc.

71317

TECHNICKÁ UNIVERZITA V KOŠICIACH

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Katedra elektroniky a multimediálnych telekomunikácií

ZADANIE DIPLOMOVEJ PRÁCE

Študijný odbor: **Informatika**

Študijný program: **Počítačové siete**

Názov práce:

Zabezpečená komunikácia klient server s využitím post-kvantových algoritmov

Secure Client Server Communication Based on Post-Quantum Cryptographic Algorithms

Študent: **Bc. Jozef Šimko**

Školiteľ: **prof. Ing. Miloš Drutarovský, CSc.**

Školiace pracovisko: **Katedra elektroniky a multimediálnych telekomunikácií**

Konzultant práce:

Pracovisko konzultanta:

Pokyny na vypracovanie diplomovej práce:

Analyzujte možnosť využitia zabezpečenej TCP/IP komunikácie klient server s využitím post-kvantových algoritmov a knižnice OpenSSL. V teoretickej časti opíšte aktuálny stav štandardizácie post-kvantových algoritmov v rámci NIST súťaže a z pohľadu súťaže perspektívne post-kvantové algoritmy. S využitím vhodných existujúcich kryptografických knižníc vytvorte kompaktnú zabezpečenú demonštračnú implementáciu senzorového uzla pre štandardnú TCP/IP komunikáciu klient-server s využitím protokolu TLS a certifikátov chránených post-kvantovými algoritmami. Implementáciu vytvorte v jazyku C a použite GCC prekladač. Experimentálne overte konektivitu vytvoreného klienta s vlastným serverom na PC platforme. Analyzujte a vyhodnoťte výpočtové nároky jednotlivých častí vytvoreného riešenia a porovnajte ich s riešením na báze štandardného protokolu TLS 1.3.

Jazyk, v ktorom sa práca vypracuje: slovenský

Termín pre odovzdanie práce: 19.04.2024

Dátum zadania diplomovej práce: 31.10.2023

TECHNICKÁ UNIVERZITA V KOŠICIACH FAKULTA ELEKTROTECHNIKY A INFORMATIKY -11-

prof. Ing. Liberios Vokorokos, PhD.

dekan fakulty

Čestné vyhlásenie

Vyhlasujem, že som záverečnú prácu vypracoval(a) samostatne s použitím uvedenej odbornej literatúry.

Košice, 19.4.2024 .....

Vlastnoručný podpis

Poďakovanie

Na tomto mieste by som rád poďakoval svojmu vedúcemu práce, prof. Ing. Milošovi Drutarovskému, CSc. za jeho čas, ochotu a odborné vedenie počas riešenia mojej záverečnej práce.

Zároveň by som sa rád poďakoval svojej rodine a priateľom za ich podporu a povzbudzovanie počas celého môjho štúdia.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Zoznam zdrojových kódov

Zoznam skratiek

AES Advanced Encryption Standard.

API Application Programming Interface.

BIKE Bit Flipping Key Encapsulation.

BKZ Block-Korkine-Zolotarev.

BPS Bezpečnosť počítačových systémov.

CA Certificate Authority.

CPU Central Processing Unit.

CVP Closest Vector Problem.

DEMA Differential Electromagnetic Analysis.

DH Diffie-Hellman.

DNSSEC Domain Name System Security Extensions.

DPA Differential Power Analysis.

DSA Digital Signature Algorithm.

ECC Elliptic Curve Cryptography.

ECDH Elliptic Curve Diffie-Hellman.

ECDSA Elliptic Curve Digital Signature Algorithm.

EUF-CMA Existential Unforgeability under Chosen Message Attack.

FALCON Fast-Fourier Lattice-base Compact Signatures over NTRU.

FIPS Federal Information Processing Standards.

GCC GNU Compiler Collection.

GNU GNU's Not Unix.

GPU Graphics Processing Unit.

HQC Hamming Quasi-Cyclic.

IDs Identifiers.

IKE Internet Key Exchange.

IND-CCA Indistinguishability under Chosen Ciphertext Attack.

IND-CPA Indistinguishability Under Chosen Plaintext Attack.

IoT Internet of Things.

IPsec Internet Protocol Security.

KEM Key-Encapsulation Mechanism.

KEX Key Exchange.

LMS Leighton-Micali Signatures.

LWE Learning with Errors.

ML-DSA Module Lattice Digital Signature Algorithm.

ML-KEM Module Lattice Key Encapsulation Mechanism.

MLWE Module Learning with Errors.

MPC Multi-Party Computation.

MSVC Microsoft Visual C++.

MSYS Minimal System.

NIST National Institute of Standards and Technology.

NIST SP National Institute of Standards and Technology Special Publication.

NP Non-deterministic Polynomial.

NTRU Number Theory Research Unit.

NTT Number Theoretic Transform.

OIDs Object Identifiers.

OQS Open Quantum Safe.

OS Operating System.

PKE Public Key Encryption.

RAM Random-Access Memory.

RBG Random Bit Generator.

RDTSC Read Time-Stamp Counter.

RSA Rivest-Shamir-Adleman.

SHA Secure Hash Algorithm.

SIKE Supersingular Isogeny Key Encapsulation.

SIS Short Integer Solution.

SSH Secure Shell.

SSL Secure Sockets Layer.

SUF-CMA Strong Existential Unforgeability under Chosen-Message Attack.

SVP Shortest Vector Problem.

TCP/IP Transmission Control Protocol/Internet Protocol.

TLS Transport Layer Security.

UOV Unbalanced Oil and Vinegar.

WSL2 Windows Subsystem for Linux 2.

XMSS eXtended Merkle Signature Scheme.

Úvod

Post-kvantová kryptografia (PQC – Post Quantum Cryptography) patrí medzi aktuálne najviac skúmané a rozvíjané oblasti kryptografie, predovšetkým kvôli bezpečnostným rizikám, ktoré predstavujú kvantové počítače pre aktuálne používané kryptografické algoritmy ako algoritmus RSA (Rivest-Shamir-Adleman) alebo algoritmus ECDH (Elliptic Curve Diffie-Hellman). Napriek tomu, že kvantové počítače momentálne nedosahujú požadovaný výpočtový výkon, tak teoretické a experimentálne analýzy dokazujú, že dostatočne robustný a stabilný kvantový počítač bude schopný vypočítať matematické problémy, na ktorých je založená bezpečnosť aktuálne používaných kryptografických algoritmov. Z tohto dôvodu vznikla nová oblasť kryptografie, teda PQC, zaoberajúca sa algoritmami, ktorých bezpečnosť bude založená na matematických problémoch, ktoré budú náročné na výpočet aj pri použití kvantových počítačov.

V roku 2016 vyhlásil americký NIST (National Institute of Standards and Technology) verejnú súťaž s cieľom nájsť a štandardizovať nové PQC algoritmy na výmenu kľúčov a digitálne podpisy podobným spôsobom, ako bola štandardizovaná symetrická šifra AES (Advanced Encryption Standard) [1] alebo symetrická šifrovacia schéma ASCON [2] pre ľahkú kryptografiu[1]. V priebehu súťaže bolo mnoho PQC algoritmov vyradených kvôli prelomeniu ich bezpečnosti alebo na základe nepriaznivých výsledkov analýzy výkonu. Aktuálne je súťaž v štvrtom kole, pričom boli vyhlásení finálni a záložní kandidáti na štandardizáciu a dostupné sú už tiež prvé verzie štandardov. Vydanie oficiálnych verzií NIST štandardov sa očakáva v priebehu tohto roku. Na základe výsledkov prvej súťaže vyhlásil NIST v roku 2023 dodatočne aj druhú súťaž venovanú iba PQC algoritmom pre digitálny podpis, s cieľom nájsť a štandardizovať PQC podpisové schémy s inými výkonovými vlastnosťami ako víťazné schémy prvej súťaže. Druhá súťaž sa v čase písania tejto práce nachádza ešte len v prvom kole a jej finálne výsledky sa očakávajú v roku 2027.

Napriek tomu, že nové PQC algoritmy by mali byť bezpečné aj pri použití

[1] `zang.lightweightcryptography`

kvantových počítačov, tak už počas súťaže bolo mnoho z nich prelomených aj pri použití klasických počítačov. Všetky nové PQC algoritmy sú skúmané z pohľadu bezpečnosti, pričom sa analyzujú jednotlivé prvky šifrovacích schém a ich odolnosť voči kryptoanalýze. Aj z toho dôvodu sa odporúča prvotne používať hybridné formy kryptografických algoritmov, ktoré sú zložené z PQC a klasických algoritmov.

Okrem bezpečnosti je dôsledne skúmaná a testovaná optimalizácia nových PQC algoritmov a ladenie výpočtových parametrov, pre zachovanie požadovanej bezpečnostnej úrovne a dosiahnutie optimálneho výkonu. Kvôli výraznemu zvýšeniu veľkostí verejných aj súkromných kľúčov, zväčšeniu certifikátov a podpisov sa tiež dôkladne skúmajú možnosti implementácie PQC algoritmov pre vstavané systémy. Okrem toho sa PQC algoritmy experimentálne implementujú aj do bežne používaných protokolov ako je TLS (Transport Layer Security), SSH (Secure Shell) alebo IPsec (Internet Protocol Security) a analyzujú sa implementačné, časové a výkonové zmeny procesov týchto protokolov v dôsledku nových algoritmov.

Prvá kapitola uvádza rozdiely medzi klasickou, kvantovou a post-kvantovou kryptografiou. Opisuje tiež koncept kvantových počítačov, ich postupný vývoj a experimentálne výsledky kvantových algoritmov, čím približuje význam a stále zvyšujúcu sa potrebu štandardizovať kvantovo-odolné kryptografické algoritmy.

Druhá kapitola sa venuje prehľadu základných matematických smerov, ktoré sa využívajú v PQC algoritmoch. Okrem toho je v kapitole priblížená špecifikácia, priebeh a predbežné výsledky NIST štandardizácie PQC algoritmov.

Cieľom tretej kapitoly je oboznámiť čitateľa s nutnými zmenami v TLS protokole a dôsledkami týchto zmien, ktoré súvisia s novými PQC algoritmami.

Štvrtá kapitola obsahuje zjednodušené opisy, matematické princípy a bezpečnostné vlastnosti predbežne štandardizovaných PQC algoritmov ML-KEM a ML-DSA.

Piata kapitola sa venuje použitým nástrojom a knižniciam s podporou PQC algoritmov, vrátane stručného návodu k ich inštalácii. Okrem toho kapitola vysvetľuje zvolený postup pri práci na úpravách zdrojových kódov vybraných knižníc a zaoberá sa vykonanými úpravami, zaujímavým doplnkom, zhodnoteniu výsledkov a testovaniu na rôznych platformách.

Posledná šiesta kapitola opisuje pripojenie vytvoreného klienta k verejnému testovaciemu serveru a tiež výsledky experimentálnych meraní časov generovania kľúčov pri procese výmeny kľúčov a overovanie podpisov zo strany klienta.

1 Význam post-kvantovej kryptografie v informačnej bezpečnosti

Šifrovacie a autentizačné algoritmy sú bežnou súčasťou dnešného života. Používané kryptosystémy sa však musia stále vyvíjať a prispôsobovať nie len rafinovanejším pokusom o prelomenie, ale tiež stále rýchlejším a výkonnejším počítačom. Modernú kryptografiu môžeme rozdeliť na celkovo 3 podkategórie:

- pre-kvantovú kryptografiu,
- kvantovú kryptografiu,
- post-kvantovú kryptografiu.

Medzi pre-kvantovú kryptografiu patria aktuálne symetrické šifry ako AES a asymetrické šifry ako RSA alebo ECC (Elliptic Curve Cryptography). Tieto šifry sú založené na pomerne jednoduchých matematických princípoch, ktoré pri dostatočne veľkých číslach nie je možné dostatočne rýchlo prelomiť ani na najmodernejších zariadeniach, tak ako ich dnes poznáme. Niektoré z nich však môžu byť zraniteľné pri použití vhodných techník na kvantových počítačoch. Tejto problematike sa venujeme v nasledujúcej podkapitole.

Kvantová a post-kvantová kryptografia sa niekedy spoločne označujú aj jedným názvom ako kvantovo-odolná kryptografia. Napriek tomu, že obe sú vyvíjané s cieľom byť odolné voči kvantovým počítačom, tak fungujú na rozdielnych princípoch, a teda je potrebné rozlišovať ich od seba.

Kvantová kryptografia sa zaoberá kryptografiou, ktorá na rozdiel od tej klasickej nevyužíva princípy matematiky, ale kvantovej fyziky, konkrétne kvantovej mechaniky [3]. Hlavnou oblasťou vývoja je kvantová distribúcia kľúča [3] prostredníctvom fotónov cez kvantové prostredie, typicky optické vlákno alebo vákuum [4]. Pre tento typ kryptografie je teda potrebné špeciálne vybavenie [5]. Inak tiež povedané, momentálne nie je možné používať kvantovú kryptografiu na klasických počítačoch. Problematike kvantovej kryptografie sa detailne venujú napríklad publikácie [4],[6].

Post-kvantová kryptografia sa venuje algoritmom, ktoré sú založené na matematických princípoch, je možné ich využívať na klasických počítačoch a zároveň budú bezpečné aj pri použití kvantových počítačov. Nemusí ísť nutne o úplne nové algoritmy, pretože kvantové počítače nepredstavujú bezpečnostné riziko pre všetky pre-kvantové šifry. Ktoré algoritmy sú zraniteľné a prečo si priblížime v nasledujúcej podkapitole.

1.1 Kvantové počítače z pohľadu post-kvantovej kryptografie

V tejto práci sa nebudeme detailne venovať fyzikálnym princípom a funkciám kvantových počítačov. Avšak pokladáme za dôležité priblížiť aspoň ich základnú charakteristiku a stručný prehľad vývoja v posledných rokoch, aby sme predstavili riziká, ktoré kvantové počítače predstavujú pre oblasť kryptografie, no zároveň poukázali na fakt, že stále majú niekoľko nedostatkov a technologických obmedzení, na ktorých sa bude v priebehu najbližších rokov pracovať.

Zjednodušene môžeme povedať, že kvantové počítače pracujú s kvantovými bitmi, inak nazývanými aj qubity, ktoré na základe princípu superpozície môžu nadobúdať aj iné stavy ako 0 a 1 pri klasických bitoch [7]. Viacero zároveň existujúcich stavov qubitov umožňuje realizáciu paralelných výpočtov [7] pri riešení matematických problémy, ktoré aktuálne počítače nedokážu riešiť v reálnom čase.

Medzi tieto matematické problémy patria, okrem iných, aj faktorizácia čísel, ktorú môžeme použiť na prelomenie algoritmu RSA, a tiež výpočet diskrétneho logaritmu, ktorým môžeme napadnúť DH (Diffie-Hellman), resp. ECDH algoritmus na výmenu kľúčov. Akým spôsobom sú tieto výpočty realizované pomocou kvantových algoritmov si priblížime v podkapitolách 1.2 a 1.3. Prehľad vybraných matematických problémov, ktoré je možné zrýchlene riešiť prostredníctvom kvantového počítania, je možné nájsť v článku [8].

Samotné kvantové algoritmy však nestačia a na prelomenie aktuálne používaných šifier potrebujeme kvantový počítač, ktorý bude univerzálny, škálovateľný a spoľahlivý [9].

Pre realizáciu výpočtov konkrétnych matematických problémov musíme splniť niekoľko podmienok, medzi ktoré patria [10]:

- dostatočné množstvo qubitov, ideálne bez interferencie, aby sme znížili celkovú chybovosť,

- stabilný systém, ktorý dokáže udržať kvantový stav všetkých qubitov čo najdlhšie,
- vhodný kvantový algoritmus s univerzálnymi kvantovými operáciami na realizáciu daného problému.

To, že vývoj kvantových počítačov napreduje, môžeme vidieť na postupnom zvyšovaní množstva qubitov v priebehu posledných rokov. Zatiaľ čo v roku 1998 boli pri testovaní kvantového algoritmu použité iba dva qubity [11], v roku 2001 sa ich počet zvýšil na 7 qubitov [12], v roku 2006 zase na 12 qubitov [13], v roku 2017 ich bolo už 50 [14] a v roku 2018 až 72 qubitov [15].

Avšak zvyšovanie qubitov v systéme neznamená aj zvýšenie celkového výkonu. Dôležité je zabezpečiť čo najväčšie kvantové previazanie a znižovať zašumenie systému [10]. Vhodným príkladom sú počítače od spoločnosti D-Wave, konkrétne počítač 2000Q z roku 2017, ktorý používa 2000 qubitov a jeho nástupca Advantage[1] z roku 2020, ktorý používa cez 5000 kvantových bitov. Počet používaných qubitov v týchto systémoch je určite obdivuhodný, no funkcie oboch počítačov nie sú univerzálne – sú navrhnuté na riešenie optimalizačných problémov [10],[16],[17]. Aj preto sa tieto zariadenia označujú ako kvantové optimalizátory [18]. Počítače od D-Wave vyvolali veľkú vlnu kontroverzie a kritiky [16]. Napriek tomu však spoločnosť niekoľko z nich predala iným spoločnostiam [19] a pracuje na ich postupnej optimalizácii a zvyšovaní množstva qubitov [17]. D-Wave zároveň predstavila prístup k počítaču 2000Q prostredníctvom cloudu[2] [20].

Čo sa týka univerzálnych kvantových počítačov, tak v roku 2022 predstavila spoločnosť IBM kvantový procesor, ktorý používa 433 qubitov [21] a koncom roka 2023 jeho nástupcu, ktorý používa celkovo 1121 qubitov [22], čo momentálne predstavuje najvyšší počet použitých qubitov v univerzálnom procesore[3]. Okrem toho IBM predstavila a sprístupnila testovacie počítače a kvantové simulátory aj verejnosti prostredníctvom cloudovej technológie a Python frameworku[4].

Pri qubitoch musíme riešiť aj otázku chybovosti. Nízku chybovosť, a teda vysoký výpočtový výkon systému [23], môžeme docieliť viacerými metódami znižovania interferencie, ktorá je spôsobená prostredím alebo vplyvom ďalších qubitov v systéme. Pochopenie týchto metód vyžaduje hlbšie znalosti v oblasti kvantovej mechaniky, a preto prikladáme doplňujúce články k tejto problematike [23], [24],[25],[26]. Pre účely tejto práce nám stačí povedať, že znižovanie chybovosti

[1] `https://www.dwavesys.com/solutions-and-products/systems/`
[2] `https://cloud.dwavesys.com/leap/`
[3] stav ku dňu 09.03.2024
[4] `https://quantum.ibm.com/services/resources`

patrí medzi hlavné výzvy pri konštrukcii škálovateľných a spoľahlivých kvantových počítačov.

Ďalšou výzvou je stabilita systému na udržanie kvantového stavu qubitov čo najdlhšie. Aktuálne sa životnosť qubitov pri použití vhodných techník pohybuje v desiatkach až stovkách milisekúnd [25],[27], pričom vedci sa snažia tento čas stále zvyšovať [28],[29]. Aj keď to tak na prvý pohľad nevyzerá, aj pri takto krátkom čase môžeme pri použití dostatočného množstva qubitov získať výsledok daného problému v priebehu niekoľkých sekúnd, pričom pri využití klasického počítača by nám to trvalo niekoľko hodín alebo dní.

Približný vznik kryptoanalyticky relevantného kvantového počítača sa odhaduje už v roku 2026 [9], zatiaľ čo iné odhady sa pohybujú okolo roku 2030 alebo 2031 [9]. Napriek množstvu predpokladaných vymožeností, ktoré kvantové počítače môžu priniesť, je nutné využiť dostupný čas aj na prípravu ochrany voči rizikám, ktoré sú tiež očakávané.

1.2 Shorov kvantový algoritmus

Za jeden z najvýznamnejších kvantových algoritmov je považovaný Shorov algoritmus, ktorý v roku 1994 predstavil Peter Shor. Tento algoritmus umožňuje realizovať výpočet faktorizačného problému pomocou kvantového paralelizmu [10]. Nejde o priamy výpočet prvočíselných súčiniteľov, ale o premenu problému na hľadanie periodickej funkcie [7]. Ide o pravdepodobnostný algoritmus, ktorý funguje v polynomiálnom čase a využíva vlastnosť superpozície kvantových stavov [10].

Shorov algoritmus sa skladá z dvoch častí [30]:

1. premena faktorizačného problému (rozklad na činitele) na problém hľadania periódy,

2. kvantový algoritmus pre hľadanie periódy.

Samotná existencia periodickej funkcie úzko súvisí s vlastnosťami modulárnej aritmetiky nad konečným poľom, ktoré sa v kryptografii často využívajú. Napríklad, ak na všetky hodnoty $\mathbb{GF}(7)$, teda $\{0, 1, 2, 3, 4, 5, 6\}$, použijeme operáciu modulo 3, tak nám vyjdu hodnoty $\{0, 1, 2, 0, 1, 2\}$, ktoré sa opakujú s hodnotou periódy 3.

Celý princíp transformácie problému faktorizácie čísla na problém hľadania periódy a jeho riešenie môžeme ukázať na príklade [30],[31]:

1. Majme číslo N, ktoré nie je prvočíslo a je nepárne. Napríklad $N = 15$.

2. Náhodne vyberieme číslo x medzi 1 a N, $a = 7$.

3. Vypočítame $gcd(a, N)$ použitím Euklidovho algoritmu alebo Malej Fermatovej vety. Ak $gcd(a, N) \neq 1$, tak spoločný deliteľ týchto čísel je zároveň jeden zo súčiniteľov čísla N. V našom prípade $gcd(7, 15) = 1$, takže postupujeme na ďalší krok.

4. Hľadáme periódu funkcie $f(x) = a^x \pmod N$, teda najmenšie celé číslo, pre ktoré platí $f(x) = f(x+r)$. Majme premennú $q = 1$, pomocou ktorej budeme počítať $qx \pmod N$. Ak bude výsledok modulárneho delenia rovný 1, pokračujeme na ďalší krok, inak nahradíme *q* zvyškom po delení a opakujeme až kým nám nevyjde zvyšok 1.

$$\begin{aligned} 1 * 7 \mod 15 &\equiv 7 \\ 7 * 7 \mod 15 &\equiv 4 \\ 4 * 7 \mod 15 &\equiv 13 \\ 13 * 7 \mod 15 &\equiv 1 \end{aligned}$$

   Realizovali sme 4 cykly výpočtov, preto $r = 4$.

5. Ak *r* je nepárne, vrátime sa k bodu č. 2 a zvolíme si iné číslo.

6. Majme číslo *p* rovné zvyšku v $\frac{r}{2}$ iterácii, teda $p = 4$. Ak $p + 1 = N$, vrátime sa k bodu č. 2 a zvolíme si iné číslo. V našom prípade $4 + 1 \neq 15$.

7. Výsledné súčinitele vypočítame ako $gcd(p \pm 1, N)$, teda

$$\begin{aligned} gcd(3, 15) &= 3, \\ gcd(5, 15) &= 5. \end{aligned}$$

Transformovanie faktorizačného problému na problém hľadania periodickej funkcie bol známy už pred samotným vznikom Shorovho algoritmu. Avšak práve hľadanie periódy, ktoré v našom opise algoritmu predstavuje bod 4, je pri použití veľkých čísel extrémne náročné pre klasické počítače. Niekedy sa môžeme stretnúť aj so zápisom, v ktorom bod 4 rozpíšeme ako $x^r \equiv 1 \pmod N$ [30]. Po úpravách získame

$$a^r - 1 = (a^{\frac{r}{2}} - 1)(a^{\frac{r}{2}} + 1) \equiv 0 \mod N).$$

Takáto formulácia daného problému je následne transformovaná na kvantové inštrukcie [32], ktoré sú vykonávané kvantovým počítačom.

Shor pri riešení tohto problému použil Simonov kvantový algoritmus [33] pre hľadanie periódy binárnych boolovských funkcií, pričom do pôvodného algoritmu doplnil kvantovú Fourierovú transformáciu [10]. Algoritmus pre n-bitový vstup potrebuje minimálne $2n + 3$ qubitov v kvantovej superpozícii stavov, ktoré sa vzájomne vyrušia deštruktívnou interferenciou, ak sa ich stavové hodnoty výrazne líšia [10]. Hodnota finálneho stavu je s vysokou pravdepodobnosť hodnota výslednej periódy [10]. Zníženie chybovosti je možné docieliť opakovaním celého algoritmu alebo využitím viacerých qubitov. Detailný opis kvantovej časti algoritmu, spolu s definíciami kvantových stavov a kvantovej Fourierovej transformácie môže čitateľ nájsť napríklad v článkoch [32],[34],[35],[36].

Shor na základe svojho algoritmu definoval postup na výpočet problému faktorizácie čísel, problému diskrétneho logaritmu aj problému diskrétneho logaritmu prvku eliptickej krivky [10].

Shorov algoritmus bol prvýkrát experimentálne overený tímom výskumníkov z IBM v roku 2001, ktorým sa na kvantovom počítači založenom na metóde nukleárnej magnetickej rezonancie podarilo nájsť prostredníctvom siedmych qubitov faktorizáciu čísla 15 [12]. Na prvý pohľad sa môže zdať, že toto číslo je príliš malé a nájsť jeho prvočíselný rozklad je jednoduché, no z pohľadu výskumu kvantových počítačov šlo o zásadný experiment, ktorý demonštroval schopnosť udržať dostatočnú kontrolu, aspoň v tej dobe, nad jedným z najväčších a najkomplexnejších kvantových systémov a spustiť kvantový algoritmus na výpočet matematického problému [12].

Za dôležitý sa tiež považuje experiment z roku 2016, kedy sa odborníkom z MIT a Univerzity v Innsbrucku podarilo vytvoriť malý kvantový počítač z piatich atómov v iónovej pasci [37]. Prostredníctvom tohto počítača dokázali overiť Shorov algoritmus a nájsť prvočíselný rozklad pre číslo 15. V tomto prípade však bolo na výpočet faktorizácie čísla 15 využitých iba 5 qubitov, z ktorých každý bol prezentovaný jedným atómom [37], pričom štyri atómy boli využité na spustenie Shorovho algoritmu, zatiaľ čo piaty atóm bol využitý na extrahovanie a ukladanie výsledkov. Stabilita systému bola zaručená iónovou pascou. Zníženie počtu využívaných qubitov bolo možné docieliť metódou recyklácie qubitov [37]. Okrem použitej technológie je hlavným rozdielom medzi týmto experimentom a experimentom IBM z roku 2001 tiež možnosť škálovateľnosti celého systému. To znamená, že celý systém bol navrhnutý tak, aby bolo možné do neho pridávať ďalšie atómy a lasery [37], s čím ďalej súvisí aj jeho výpočtový výkon, a teda schopnosť určiť prvočíselný rozklad aj pre oveľa väčšie čísla. Výsledok tohto experimentu zároveň ukázal, že momentálne využívané kryptografické algoritmy budú s po-

stupným vývojom kvantových počítačov prelomené a je nutné pracovať na kryptografii, ktorá bude bezpečná aj pri používaní takýchto počítačov.

Okrem týchto experimentov existuje ďalšie množstvo odborných publikácií, v ktorých autori uvádzajú, že dokázali rozložiť veľké číslo na súčin dvoch menších čísel [38],[39]. Niektoré z týchto prác sú označované za zavádzajúce, pretože algoritmy výpočtov sú prispôsobené pre vopred zvolené číslo, prípadne obsahujú minimum kvantových výpočtov. Za najnovšie a pomerne zaujímavé publikácie považujeme [40],[41].

1.3 Groverov algoritmus

Ďalším významným kvantovým algoritmom je Groverov algoritmus. Ten predstavil Lov Grover v roku 1996. Ide o algoritmus , ktorý dokáže nájsť prvok v neusporiadanom zozname obsahujúcom N prvkov a to s vysokou pravdepodobnosť v $\sqrt{N}$ krokoch [10], čo znamená, že nie je potrebné prehľadávať celý zoznam. Oproti klasickému výpočtu poskytuje tento algoritmus kvadratické zrýchlenie. Algoritmus je zároveň možné aplikovať aj na určenie priemeru a mediánu neusporiadaného zoznamu, na riešenie problému kolízie funkcií alebo na niektoré optimalizačné problémy ako sú nedeterministické polynomiálne (NP – Non-deterministic Polynomial) problémy [8],[36].

Zjednodušene Groverov algoritmus najprv vytvorí kvantové orákulum, maticu U_ω, ktorá definuje predpokladaný stav. Orákulum je následne aplikované na zoznam prvkov, ktorý je reprezentovaný kvantovým zásobníkom vo forme qubitov v superpozícii [10]. Orákulum spôsobí, že sa zmení amplitúda hľadaného stavu [42],[43]. Následne je na kvantový zásobník aplikovaná operácia Groverovej difúzie (tiež Groverova operácia či iterácia), ktorá nájde zmenený stav a vráti výsledok [10]. Pravdepodobnosť správnosti výslednej hodnoty je vysoká, pričom zníženie chybovosti je možné docieliť opakovaním celého algoritmu [43].

Matematické, fyzikálne a geometrické dôkazy platnosti Groverovho algoritmu môžeme nájsť napríklad v publikáciach [36],[43],[44].

2 Post-kvantová kryptografia

V predošlej kapitole sme priblížili, akým spôsobom je možné aplikovať Shorov algoritmus na riešenie problému faktorizácie čísel a riešenie problému diskrétneho logaritmu, čím vzniká predpoklad, že postupným pokrokom vo vývoji kvantových počítačov bude ohrozená bezpečnosť systémov, ktoré využívajú asymetrickú kryptografiu. Predpokladá sa, že algoritmy ako RSA, DSA (Digital Signature Algorithm) , ECDSA (Elliptic Curve Digital Signature Algorithm), DH alebo ECDH, ktoré sa momentálne bežne využívajú, budú s postupom času prelomené [7],[9]. Tieto algoritmy je potrebné nahradiť post-kvantovými algoritmami. Naopak Groverov algoritmus môžeme aplikovať na hľadanie kľúča symetrických šifier útokom hrubou silou [10] a na kolízne útoky na hašovacie funkcie. Na dosiahnutie rovnakej úrovne bezpečnosti symetrických algoritmov musíme použiť kľúč s dvakrát väčšou dĺžkou [7],[9].

Prehľad vybraných algoritmov s porovnaním ich aktuálnej a post-kvantovej bezpečnosti zobrazuje tabuľka 2.1.

Tabuľka 2.1: Porovnanie bežne používaných pre-kvantových algoritmov a ich aktuálnej aj post-kvantovej bezpečnosti [45]

Algoritmus	**Dĺžka kľúča [b]**	**Aktuálna bezpečnosť [b]**	**Post-kvantová bezpečnosť [b]**
AES-128	128	128	64
AES-256	256	256	128
Salsa20	256	256	128
SHA-256	256	256	128
SHA3-256	256	256	128
RSA-2048	2048	112	0
RSA-3072	3072	128	0
DH-3072	3072	128	0
DSA-3072	3072	128	0
256-bit ECDH	256	128	0
256-bit ECDSA	256	128	0

Post-kvantová kryptografia využíva rôzne matematické problémy, na základe

ktorých môžeme jednotlivé PQC algoritmy rozdeliť na [7],[9],[10]:

- algoritmy na báze teórie kódovania,
- algoritmy na báze hašovacích funkcií,
- algoritmy využívajúce mriežky.

V odborných publikáciach sa môžeme stretnúť aj s iným delením, v ktorom sú okrem vyššie uvedených aj [7],[9]:

- algoritmy na báze izogénie nad supersingulárnymi eliptickými krivkami [9],
- multivariačné algoritmy založené na polynomiálnych rovniciach [9],
- MPC-in-the-Head algoritmy [46].

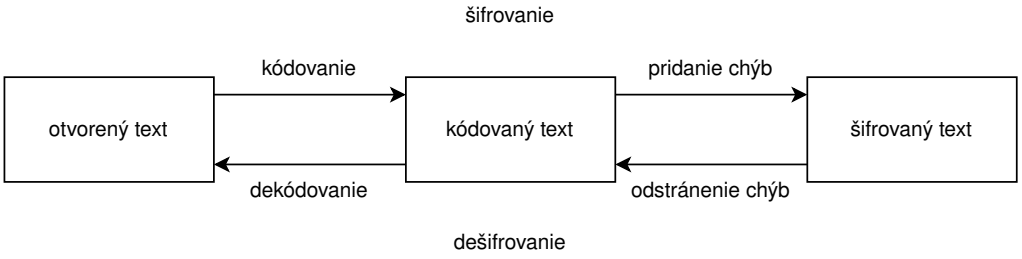
Týmto algoritmom sa však v našej práci nebudeme bližšie venovať, pretože aj keď sú založené na potenciálne kvantovo-odolných matematických základoch, tak medzi očakávanými štandardmi sa aktuálne nenachádza žiadny algoritmus založený na týchto problémoch.

Kryptografia na báze teórie kódovania je založená na bezpečnosti samoopravných kódov. Prvý algoritmus na báze kódov predstavil McEliece už v roku 1978 [47]. Algoritmy sa opierajú o dva výpočtové problémy, za predpokladu, že sú použité dostatočne veľké dĺžky kľúčov [45],[48]:

- problém všeobecného dekódovania, ktorý patrí medzi nedeterministické polynomiálne problémy,
- náročnosť odlíšiť verejný kľúč vo forme matice od náhodnej matice, čo závisí od konkrétneho druhu použitých samoopravných kódov.

Generovanie kľúčov predstavuje vytvorenie verejného kľúča, maticu generátora, a súkromného kľúča, teda dekodéru s definovanou váhou chýb. Šifrovanie je založené na násobení správy vo forme vektora s maticou, ktorá predstavuje samoopravný lineárny kód, pričom k výsledku násobenia je následne pridaná chyba s definovanou Hammingovou váhou [48]. Pri dešifrovaní je zo šifrovaného textu odstránená chyba dekodérom a následne je reťazec vynásobený multiplikatívnou inverziou pôvodnej matice lineárneho kódu [48],[49]. Proces šifrovania a dešifrovania môžeme vidieť na obrázku 2.1.

Operácie generovania kľúčov, šifrovania a dešifrovanie sú veľmi rýchle, avšak minimálna veľkosť kľúčov sa pohybuje v tisíckach až miliónoch bajtov [48], čo neumožňuje využívať tieto algoritmy na všetkých zariadeniach. Novšie algoritmy



Obr. 2.1: Proces šifrovania a dešifrovanie algoritmom na báze samoopravných kódov [48]

používajú rôzne štruktúry na kompresiu kľúčov, ktoré však vo väčšine prípadov spôsobili prelomenie celého algoritmu [45]. Medzi záložnými kandidátmi NIST štandardizácie PQC algoritmov sa nachádzajú celkovo tri algoritmy na báze kódovania, medzi nimi aj upravená verzia pôvodného algoritmu McEliece.

Kryptografia využívajúca hašovacie funkcie je označenie pre algoritmy, ktoré sú založené na bezpečnosti konkrétnych hašovacích funkcií, ktoré by mali byť odolné voči nájdeniu vzoru a zároveň spĺňali podmienky slabej a silnej bezkolíznosti. Keďže sa na hašovacie funkcie vzťahuje len Groverov kvantový algoritmus, tak pri dostatočnom zvýšení dĺžky kľúča by mali hašovacie funkcie ako SHA3 (Secure Hash Algorithm) alebo BLAKE2 ostať kvantovo-bezpečné [7],[9].

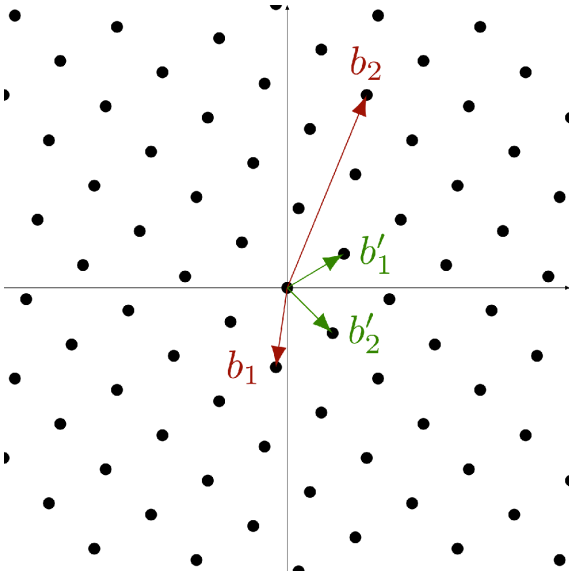
Súkromný kľúč predstavuje vstupný reťazec hašovacej funkcie a verejný kľúč predstavuje výsledný haš. Takéto algoritmy nie sú vhodné na výmenu kľúčov [49], ale sú efektívne pri podpisovaní certifikátov, hlavne vďaka svojej rýchlosti.

Lamportova schéma pre jednorázový podpis z roku 1979 [50] bola prvým algoritmom založenom na hašovacích funkciách [7]. Nedostatky tejto schémy vo forme obmedzeného množstva použiteľných kľúčov a ich veľkosti boli vyriešené aplikáciou Merklovej stromovej štruktúry [51],[10]. Na tomto princípe, samozrejme s doplnenými optimalizačnými úpravami, aktuálne fungujú algoritmy LMS (Leighton-Micali Signatures) [52] a XMSS (eXtended Merkle Signature Scheme) [53] [54], ktoré už sú pomerne známe a štandardizované. Nevýhodou týchto algoritmov je udržiavanie stavu, čo znamená, že podpisovateľ musí viesť záznamy o súkromných kľúčoch v stromovej štruktúre a o presnom počte podpísaných správ [55], pričom každá chyba v záznamoch predstavuje bezpečnostné riziko. Zároveň je možné produkovať obmedzený počet súkromných kľúčov, ktorý je možné zvýšiť na úkor veľkosti podpisu [54],[55].

Problémy stavových algoritmov na základe hašovacích funkcií riešia bezstavové algoritmy [7], akým je napríklad SPHINCS [56], jeden z víťazných algoritmov NIST štandardizácie 2. Bezstavové algoritmy nevyžadujú udržiavať záznamy

o súkromných kľúčoch a vykonaných podpisoch, avšak dôsledkom toho sa výrazne zväčšila veľkosť podpisov [7].

Kryptografia na báze mriežky je založená na problémoch hľadania najkratšieho vektora k báze (SVP – Shortest Vector Problem), hľadania najbližšieho vektora k báze alebo bodu (CVP – Closest Vector Problem), učenia sa s chybami (LWE – Learning with Errors) a hľadania nenulového vektoru (SIS – Short Integer Solution) v mriežke [54],[57]. Mriežka je tvorená usporiadanými bodmi v n-rozmernom vektorovom priestore, pričom báza je označenie pre skupinu vektorov, ktorých kombinácie umožňujú vygenerovať všetky body mriežky [10]. Z pohľadu vytvárania bezpečnostných algoritmov je zaujímavý fakt, že dve úplne rozdielne bázy môžu generovať rovnakú mriežku, čo môžeme prirovnať k existencii súkromného a verejného kľúča v asymetrickej kryptografii. Mriežka s dvomi odlišnými bázami je zobrazená na obrázku 2.2. Algoritmy založené na problémoch mriežky sa vyznačujú svojou jednoduchosťou, rýchlosťou a efektívnosťou [55]. Tieto algoritmy sa používajú pri šifrovaní, homomorfnom šifrovaní, výmene kľúčov aj v podpisových schémach [7],[49]. Proces generovania kľúčov, šifrovania a dešifrovania sa líši od konkrétneho algoritmu.



Obr. 2.2: Príklad dvojrozmernej mriežky, ktorá môže byť generovaná pomocou dvoch rozdielnych báz (párov vektorov) [57]

V roku 1996 predstavil Miklós Ajtai prvý algoritmus [58] založený na mriežke [7]. Šlo o hašovaciu funkciu, ktorá bola založená na SIS probléme. Následne v roku 1998 predstavili Hoffstein, Pipher a Silverman šifrovací algoritmus NTRU (Number Theory Research Unit) [59], založený na SVP probléme [7]. NTRU nebol doteraz prelomený a bol súčasťou štandardizácie post-kvantových algoritmov. Pred-

bežným víťazom štandardizácie je CRYSTALS-Kyber [60], algoritmus na výmenu kľúčov založený na LWE probléme nad konečným poľom vo vektorovom priestore (MLWE – Module Learning with Errors), ktorému sa venujeme v podkapitole 4.1. Medzi víťaznými podpisovými schémami sa nachádza aj CRYSTALS-Dilitihum [61], tiež založený na problémoch mriežky, ktorému sa venujeme v podkapitole 4.2.

NIST štandardizácia PQC algoritmov

S postupným rozvojom kvantových počítačov v priebehu rokov pribúdali práce o dôležitosti a potrebe štandardizácie post-kvantových algoritmov. Vznikali preto rôzne šifrovacie algoritmy, ktoré boli kryptoanalyticky testované. Niektoré z nich sa testovali aj v reálnych aplikáciách. Napríklad v roku 2015 pridal Google podporu[1] pre post-kvantový algoritmus New Hope na výmenu kľúčov na vybrané servery, v roku 2016 zase Microsoft vydal vývojársku knižnicu[2] na experimentovanie s hybridným post-kvantovým algoritmom.

Za významný považujeme rok 2016, kedy americký Národný inštitút pre štandardy a technológie NIST vyhlásil verejnú súťaž pod názvom Štandardizácia post-kvantovej kryptografie[3] s cieľom vytvoriť, testovať a štandardizovať šifrovacie algoritmy, ktoré budú bezpečné aj pri použití kvantových počítačov. Očakáva sa, že nové PQC algoritmy s verejným kľúčom by mali nahradiť 3 kryptografické štandardy [62] – FIPS (Federal Information Processing Standards) 186 [63], NIST SP (National Institute of Standards and Technology Special Publication) 800-56A [64] a NIST SP 800-56B [65]. Ešte pred vyhlásením samotnej súťaže bola zverejnená požiadavka na návrh minimálnych podmienok na predloženie, prijatie a hodnotenie kandidátskych algoritmov. Všetky prijaté návrhy sú spolu s vytvorenými kritériami verejne prístupné [62]. Následne bola 20. decembra 2016 oficiálne vyhlásená samotná súťaž, ktorá bola rozdelená do dvoch kategórií:

- šifrovanie s verejným kľúčom (PKE – Public Key Encryption) a výmena kľúčov (KEX – Key Exchange), resp. enkapsulácia kľúčov (KEM – Key Encapsulation Mechanism)
- schémy digitálnych podpisov.

[1] `https://security.googleblog.com/2016/07/experimenting-with-post-quantum.html`

[2] `https://www.microsoft.com/en-us/download/details.aspx?id=52438&751be11f-ede8-5a0c-058c-2ee190a24fa6=True`

[3] z ang. Post-Quantum Cryptography Standardization

Celá súťaž bola rozdelená na štyri kolá, počas ktorých boli jednotlivé algoritmy prijímané a posudzované výskumníkmi z NIST-u a odbornej verejnosti. Po každom kole prebiehali workshopy, kde každý tím, ktorý splnil nutné podmienky, predstavil svoju implementáciu. Následne prebiehali kryptoanalytické a implementa-čno-analytické hodnotenia, ktoré zúžili množstvo postupujúcich kandidátov. Jednotlivé hodnotenia boli realizované v troch oblastiach:

- bezpečnosť,
- metriky výkonnosti,
- charakteristiky implementácie.

Bezpečnosť bola hlavným a najdôležitejším kritériom hodnotenia. NIST oznámil integráciu všetkých štandardizovaných post-kvantových algoritmov do bežne používaných protokolov ako sú TLS, SSH, IKE (Internet Key Exchange), IPsec alebo DNSSEC (Domain Name System Security Extensions) [62]. Každý algoritmus bol posudzovaný z pohľadu bezpečnosti v týchto protokoloch. V prípade šifrovacích algoritmov a algoritmov na výmenu kľúčov bola definovaná IND-CCA2 (Indistinguishability under Chosen Ciphertext Attack) ako minimálna úroveň bezpečnosti a to s predpokladom, že útočník nemá vo väčšine prípadov prístup k viac než 2^{64} šifrovaným správam [62]. Pre dočasné/jednorázové[4] šifrovanie bola určená IND-CPA (Indistinguishability Under Chosen Plaintext Attack) ako minimálna úroveň zabezpečenia, pričom všetky možné riziká so znovu-použitím rovnakého kľúča museli byť vysvetlené v priloženej dokumentácii. Pre schémy na digitálne podpisy NIST určil EUF-CMA (Existential Unforgeability under Chosen Message Attack) ako bezpečnostnú úroveň a podobne ako pri algoritmoch na výmenu kľúčov definoval, že útočník môže mať prístup k najviac 2^{64} správam, avšak použitie vyššieho počtu bude tiež analyzované [62]. Ako doplnkové bezpečnostné kritérié pridal NIST možnosť využívať dokonalú doprednú bezpečnosť[5], bezpečnosť pri útokoch s použitím postranných kanálov, odolnosť proti útokom viacerými kľúčmi[6], bezpečnosť použitých matematických štruktúr a reakcie na chyby pri znovu-použití rovnakej hodnoty nonce, rovnakých párov kľúčov alebo zlej inicializácii náhodného generátoru čísel [62]. NIST zároveň uviedol, že kvôli možnému vzniku nových kvantových kryptoanalytických schém a neurčitej výpočtovej kapacite kvantových počítačov nie je možné jasne definovať

[4] z ang. ephemeral
[5] z ang. perfect forward secrecy
[6] z ang. multi-key attacks

bezpečnosť post-kvantových algoritmov a vyjadriť ju v klasických bitových jednotkách. Z týchto dôvodov boli vytvorené kategórie s definovanými referenčnými primitívami na približné rozdelenie širokej škály bezpečnosti algoritmov, aby bolo možné porovnávať parametre jednotlivých kryptosystémov, pochopiť následky zmeny parameterov a rýchlejšie reagovať na budúce riziká [62]. Celkovo vzniklo 5 bezpečnostných úrovní[7], označované aj ako levely bezpečnosti L1-L5 [66], pričom každá kategória je definovaná symetrickou šifrou a minimálnym útokom s definovanou hranicou použitých kvantových alebo klasických logických hradiel. Tieto úrovne sú zobrazené v tabuľke 2.2.

Tabuľka 2.2: Bezpečnostné úrovne definované na začiatku štandardizácie post-kvantových algoritmov pre lepšiu analýzu a kategorizáciu nových algoritmov [62]

Úroveň bezpečnosti	**Referenčná symetrická šifra**	**Počet (kvantových) logických hradiel**
L1	AES 128	$(2^{170}/\text{MAXDEPTH})$ 2^{143}
L2	SHA-256	2^{146}
L3	AES 192	$(2^{233}/\text{MAXDEPTH})$ 2^{207}
L4	SHA3-384	2^{210}
L5	AES 256	$(2^{298}/\text{MAXDEPTH})$ 2^{272}

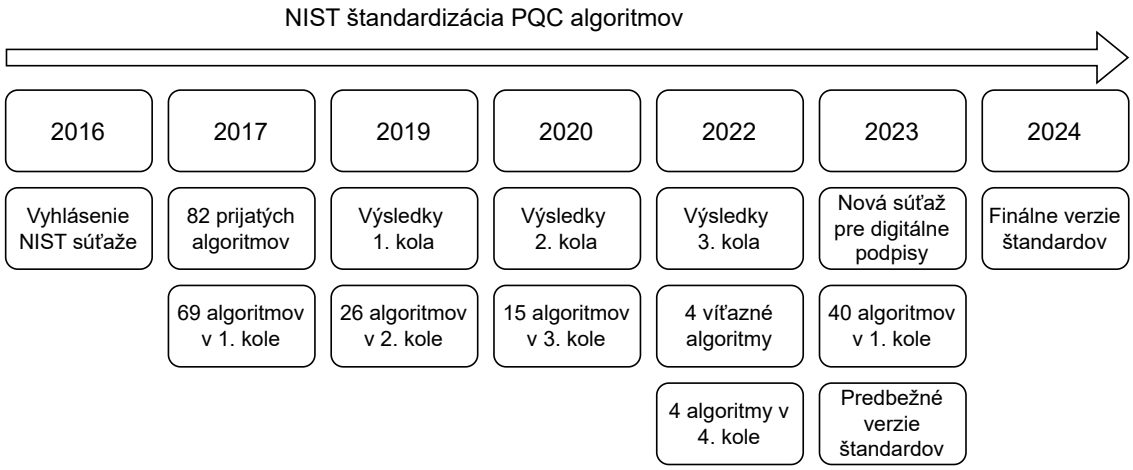
Metriky výkonnosti je označenie pre hodnotenie algoritmov na základe ich výpočtových nákladov ako sú požiadavky na pamäť (veľkosti kľúčov, šifrovaného textu alebo podpisov), efektívnosť spracovania súkromných a verejných kľúčov, efektívnosť generovania kľúčov alebo chybovosť šifrovania [62]. Výrazné zväčšenie verejných kľúčov alebo podpisov môže spôsobovať problémy v aplikáciách s obmedzenou šírkou pásma, v protokoloch kvôli limitovaným rozmerom paketov alebo v zariadeniach ako sú napríklad IoT (Internet of Things) senzory, ktoré sa vyznačujú malou internou pamäťou. Efektívnosť šifrovania, dešifrovania, enkapsulácia, dekapsulácie, podpisovania a overovania podpisu sú dôležité ako pre malé zariadenia s nízkym výkonom ako napríklad čipové karty, tak aj pre veľké servery, ktoré v jednom momente musia spracovávať obrovské množstvo požiadaviek. Aj keď posudzovanie výpočtových nárokov na generovanie kľúčov bolo definované pre šifrovacie a enkapsulačné algoritmy hlavne pri použití doprednej bezpečnosti, tak výrazné zvýšenie času generovania kľúčov môže všeobecne spôsobiť problémy v rôznych aplikáciách [62]. Chybovosť algoritmov by mala byť tiež

[7] v prvej verzii špecifikácie je možné nájsť aj 6. úroveň

čo najnižšia. Pri vzniku chýb šifrovania a dešifrovania museli autori uviesť predpokladaný pomer chybovosti algoritmu spolu s bezpečnostnými rizikami [62]. Napriek tomu, že rôzne protokoly majú funkcie, ktoré riešia problémy s chybami šifrovania, tak sú často spojené s ďalšími časovými aj výpočtovými nákladmi.

Charakteristiky implementácie boli zamerané na flexibilitu, jednoduchosť a licenčné podmienky štandardizácie [62]. V prípade flexibility boli brané do úvahy vlastnosti ako možnosť modifikácie algoritmu pre doplnenie nových funkcií, zme-ny parametrov na úpravu bezpečnosti a výpočtových nárokov, nezávislosť od použitého hardvéru vrátané zariadení s obmedzenými zdrojmi, paralelizovateľnosť výpočtov alebo jednoduchá integrácia do už existujúcich protokolov s čo najnižším množstvom nutných úprav samotného protokolu [62]. Pri jednoduchosti sa riešili detaily spojené s dizajnom a použitými štruktúrami, ktoré umožňujú lepšie pochopenie jadra celého algoritmu, a teda aj rýchlejší a efektívnejší prístup ku kryptoanalýze [62]. Licenčné podmienky boli zamerané na právnu stránku celej štandardizácie.

Priebeh NIST štandardizácie je znázornený na obrázku 2.3.



Obr. 2.3: Časová línia NIST štandardizácie PQC algoritmov

Prvé kolo bolo zamerané na zhromaždenie kandidátskych algoritmov. Toto kolo skončilo 30. novembra 2017 a celkovo bolo predložených 82 algoritmov [67]. Po overení minimálnych podmienok, ktoré zahŕňali kritéria pre referenčné a optimalizované implementácie v jazyku C, vrátane testovacích vektorov, požiadavky na obsah dokumentáciu a podmienku nezávislosti implementácie od použitého hardvéru, postúpilo na hodnotenie 69 kandidátov [67]. Všetky prelomené algoritmy boli vyradené zo súťaže.

Do druhého kola následne prešlo len 26 kandidátov [67], z toho 17 algoritmov na výmenu kľúča a 9 pre digitálny podpis. V tomto kole mal každý tím možnosť

vykonať vo svojich implementáciách menšie úpravy. Napriek tomu však počet kandidátov výrazne klesol kvôli nepriaznivým krypto-analytickým výsledkom. Zostávajúce algoritmy sa výrazne podobali v bezpečnosti, výkone aj dizajne. Silní kandidáti museli byť v tomto kole vyradení s cieľom zamerať všetku pozornosť na tie najperspektívnejšie algoritmy [68]. Pri výbere postupujúcich algoritmov boli do úvahy brané predovšetkým nároky na výpočet a efektívnosť na rôznych platformách. Z 15 postupujúcich kandidátov bolo sedem vybraných a označených za finalistov a ďalších osem za náhradníkov [68].

V treťom kole mohli tímy opäť vykonať menšie úpravy svojich implementácií a odoslať ich na hodnotenie. Vzhľadom na nízky počet postupujúcich algoritmov z predošlého kola bolo záverečné hodnotenie tretieho kola venované detailnému porovnaniu jednotlivých algoritmov vo všetkých troch základných kritériách [69].

Výsledkom tretieho kola bolo predbežné rozhodnutie o vybratí niekoľkých algoritmov pre štandardizáciu. Ako hlavný algoritmus na výmenu kľúča bol zvolený CRYSTALS-Kyber. Zároveň do štvrtého doplňujúceho kola postúpili ďalší 4 kandidáti, BIKE (Bit Flipping Key Encapsulation) [70], Classic McEliece [71], HQC (Hamming Quasi-Cyclic) [72] a SIKE (Supersingular Isogeny Key Encapsulation) [73]. Krátko po začatí štvrtého kola bol SIKE prelomený [74]. Štvrté kolo skončilo označením zvyšných troch algoritmov za záložné.

Na štandardizáciu boli tiež vybrané tri algoritmy na digitálny podpis. Aj keď je CRYSTALS-Dilithium označovaný ako primárny algoritmus a hlavný víťaz súťaže, tak algoritmy FALCON (Fast-Fourier Lattice-base Compact Signatures over NTRU) [75] a SPHINCS+ [76] boli vybrané kvôli svojim vlastnostiam, pretože každý z nich poskytuje iné výhody a nevýhody, ktoré súvisia s výpočtami, detailmi v implementácii, dĺžkou používaného kľúča, ich rýchlosťou alebo záťažou pre konkrétny hardvér [69].

Prehľad všetkých algoritmov spolu s ich typom, použitím a celkovým počtom dostupných variantov zobrazuje tabuľka 2.3.

Po oznámení víťazných algoritmov vyhlásil NIST novú súťaž pod názvom „Signature onramp“ so zámerom zhromaždiť doplnkové post-kvantové schémy na digitálne podpisy, ktoré budú založené na iných matematických základoch ako sú problémy na mriežke, ktorých odolnosť voči kryptoanalýze je stále pomerne nejasná, a zároveň budú dosahovať lepšie výkonnostné výsledky ako SPHINCS+ [77],[78]. Okrem toho si NIST od tejto súťaže sľubuje nové algoritmy s lepšími výsledkami v konkrétnych aplikáciách, v ktorých štandardizované schémy nie sú úplne optimálne. V júli 2023 bolo uzavreté prvé kolo, do ktorého postúpilo 40 kan-

Tabuľka 2.3: Prehľad postupujúcich post-kvantových algoritmov po treťom kole NIST súťaže [69]

Algoritmus	**Typ algoritmu**	**Použitie algoritmu**	**Počet variantov**
CRYSTALS-Kyber	na báze mriežky	KEM	3
HQC	na báze teórie kódovania	KEM	3
BIKE	na báze teórie kódovania	KEM	3
Classic McEliece	na báze teórie kódovania	KEM	10
CRYSTALS-Dilithium	na báze mriežky	DSA	3
FALCON	na báze mriežky	DSA	2
SPHINCS+	na báze bezstavových hašovacích funkcií	DSA	12

didátov [78]. V čase písania tejto práce bolo prelomených už 20 kandidátskych schém, no pri mnohých z nich autori uvádzajú, že dokážu nájdené chyby opraviť, takže na predbežné výsledky je nutné počkať až do oficiálneho zverejnenia hodnotenia prvého kola. Hodnotiace kritéria sú zhodné s kritériami pôvodnej súťaže, s tým rozdielom, že algoritmy na báze mriežky musia vykazovať aspoň jednu veľkú výhodu vo výkone oproti algoritmom CRYSTALS-Kyber a FALCON. V prípade schém nezaložených na mriežkach by mali mať aspoň jednu veľkú výhodu vo výkone oproti SPHINCS+ [78]. Za perspektívne sa aktuálne javia multivariačné algoritmy [77] ako UOV (Unbalanced Oil and Vinegar) [79] alebo MAYO [80]. Nepredpokladá sa, že výsledky tejto súťaže vo forme štandardov, by mali byť dostupné skôr ako v roku 2027.

Koncom augusta 2023 vydal NIST tri predbežné verzie FIPS štandardov a poskytol tak odbornej verejnosti prístup pre dodatočné komentáre a pripomienky. Tieto koncepty štandardov sú pomenované ako FIPS 203 [81], FIPS 204 [82] a FIPS 205 [83], pričom každý z nich je odvodený z algoritmov, ktoré boli označené za finalistov súťaže [84].

FIPS 203 špecifikuje kryptografickú schému nazvanú štandard mechanizmu zapuzdrenia kľúča využívajúceho moduly a mriežku [8], ktorá je odvodená od predloženého návrhu algoritmu CRYSTALS-KYBER [84].

FIPS 204 opisuje normu algoritmu pre digitálny podpis, ktorý bol odvodený od predloženého algoritmu CRYSTALS-Dilithium [84]. Predbežná verzia tohto štandardu nesie názov Štandard digitálneho podpisu založenom na moduloch a mriežke [9].

FIPS 205, štandard digitálneho podpisu založenom na bezstavovom hašovaní[10],

[8] z ang. Module-Lattice-Based Key Encapsulation Mechanism Standard
[9] z ang. Module-Lattice-Based Digital Signature Standard
[10] z ang. Stateless Hash-Based Digital Signature Standard

tiež špecifikuje algoritmus pre digitálny podpis, avšak v tomto prípade je navrhovaná schéma odvodená od algoritmu SPHINCS+ [84].

V novembri 2023 bola uzavretá verejná diskusia k predbežným štandardom. Zozbierané komentáre a pripomienky boli začiatkom decembra 2023 verejne publikované [85]. NIST uviedol, že všetky komentáre a pripomienky budú analyzované a v prípade potreby budú na ich základe upravené alebo doplnené jednotlivé koncepty štandardov. Predpokladané zverejnenie oficiálnych verzií štandardov sa odhaduje na rok 2024 [84],[86], pričom v čase písania tejto práce stále neboli zverejnené (stav ku dňu 12.4.2024). Za spomenutie tiež stojí, že NIST v roku 2024 plánuje zverejniť ešte jeden koncept štandardu, ktorý bude špecifikovať schému algoritmu pre digitálny podpis odvodený od algoritmu FALCON [84],[86].

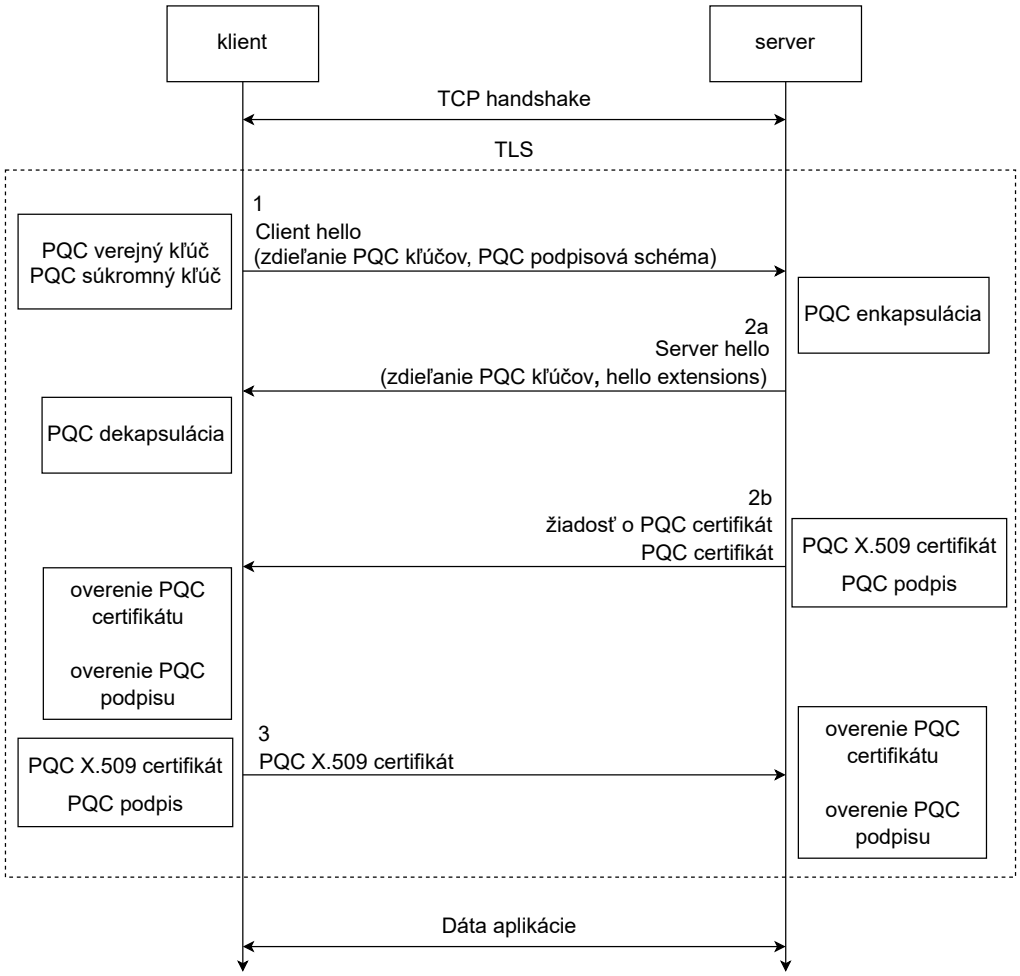
3 Post-kvantová kryptografia v protokole TLS

Všetky štandardizované algoritmy boli vybrané aj na základe toho, aby bola ich integrácia do protokolov a aplikácií čo najjednoduchšia a s čo najnižším množstvom nutných úprav [62]. TLS protokol je nezávislý od použitých algoritmov [66], ktoré sa volia pri nadviazení spojenia[1] medzi klientom a serverom vo forme parametrov. Zjednodušený proces nadviazania TLS spojenia s použitím PQC algoritmov je znázornený na obrázku 3.1. Na prvý pohľad sa môže zdať, že hlavnou zmenou je nahradenie KEX algoritmu KEM algoritmom. Aj keď enkapsulácia prebieha inak ako klasický DH, resp. ECDH algoritmus na výmenu kľúčov, tak táto zmena sa neodrazí na samotnom priebehu TLS handshaku [89]. Detailom enkapsulačných algoritmov a ich spôsobe integrácie do TLS sa venujeme v nasledujúcej podkapitole. Dôležitou zmenou je predovšetkým nutnosť definovať nové identifikačné čísla (IDs – Identifiers) enkapsulačných algoritmov, IDs algoritmov pre digitálne podpisy a identifikačné čísla X.509 objektov[2] (OIDs – Object Identifiers) [88]. Zároveň je potrebné prispôsobiť veľkosti niektorých štruktúr kvôli niektorým algoritmom a ich výrazne väčším veľkostiam kľúčov, certifikátov či podpisov. To sa samozrejme môže odraziť na množstve prenesených paketov a tiež celkovom čase TLS handshaku [66].

Za dôležité považujeme spomenúť odporúčanie používať hybridné algoritmy [66],[89],[90], zložené z pre-kvantových a post-kvantových algoritmov, pomocou ktorých sa očakáva plynulejší prechod na plne post-kvantové algoritmy, počas ktorého bude možné aktívne testovať vlastnosti a bezpečnosť post-kvantových algoritmov.

[1] z ang. handshake

[2] je nutné upraviť aj X.509 štandard pre certifikáty



Obr. 3.1: Schéma priebehu nadviazania spojenia v TLS protokole po integrácii post-kvantových algoritmov [87],[88]

3.1 Enkapsulácia kľúčov

NIST dlhodobo označoval proces výmeny kľúčov prostredníctvom DH a ECDH algoritmu ako KEX. Na začiatku štandardizácie však určil, že výmena kľúčov bude ďalej označovaná ako KEM [62]. Táto zmena však nemá žiadny vplyv na samotný proces nadviazania spojenia v TLS protokole, pretože KEM algoritmy je možné transformovať na KEX algoritmy rovnako ako niektoré KEX algoritmy sa dajú upraviť do podoby KEM algoritmov [91]. Tento fakt umožňuje pomerne jednoduchú implementáciu KEM mechanizmu do TLS protokolu.

Enkapsulačný mechanizmus je zložený z troch algoritmov/funkcií [62],[92]:

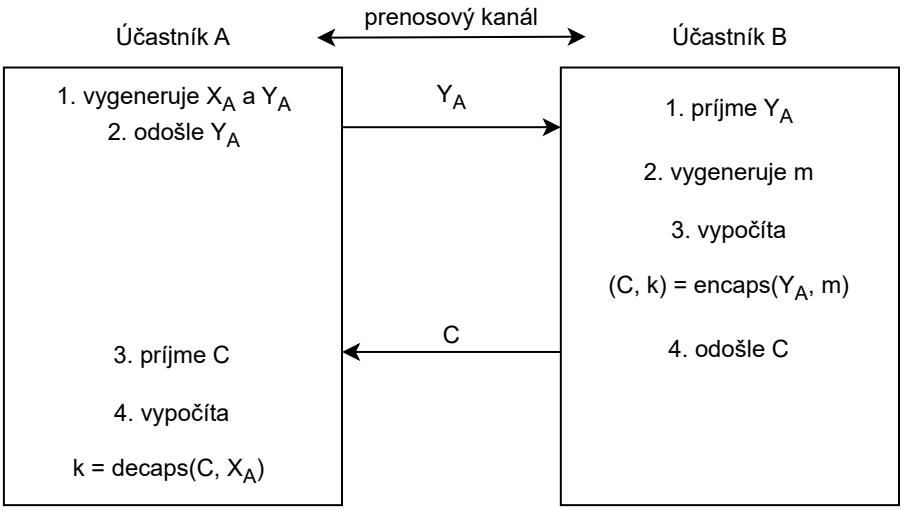
- algoritmus na generovanie súkromného a verejného kľúča,
- enkapsulačný algoritmus na zašifrovanie tajného kľúča,
- dekapsulačný algoritmus na dešifrovanie tajného kľúča.

Samozrejme, za samotným procesom generovania kľúčov, enkapsulácie a dekapsulácie je ďalšie množstvo rôznych matematických výpočtov, ktoré sa líšia v závislosti od použitého algoritmu.

Predpokladajme, že účastník A chce realizovať výmenu symetrického tajného kľúča s účastníkom B. Obaja účastníci vygenerujú svoje súkromné a verejné kľúče, ktoré si vymenia. Účastník A potom zvolí náhodný tajný kľúč a zašifruje ho svojim verejným kľúčom, tento proces je označený ako enkapsulácia. Šifrovaný text následne účastník B dešifruje (dekapsuluje) svojim súkromným kľúčom a získa tajný kľúč [92].

Implementácie KEM algoritmov v TLS protokole však používajú mierne iný proces enkapsulácie a dekapsulácie, ktorý pripomína klasický DH algoritmus. Tento proces je principiálne znázornený na obrázku 3.2.

Predpokladajme výmenu tajného kľúča medzi účastníkmi A a B. Účastník A vygeneruje súkromný kľúč X_A a verejný kľúč Y_A. Účastník A následne odošle svoj verejný kľúč Y_A účastníkovi B, analogicky k odoslaniu hodnôt p a q pri DH algoritme. Účastník B použije verejný kľúč Y_A na enkapsuláciu, ktorej výsledkom je tajný kľúč k a šifrovaný text C, ktorý odošle účastníkovi A, podobne ako sa vymieňajú parametre kľúča pri DH algoritme. Účastník A potom použije svoj súkromný kľúč X_A na dekapsuláciu šifrovaného textu C, čím získa hodnotu tajného kľúča k [92],[81].



Obr. 3.2: Proces výmeny symetrického kľúča pri použití enkapsulačného mechanizmu v TLS

Integrácia do TLS

Ako sme v úvode kapitoly spomenuli, tak TLS protokol je nezávislý od použitých algoritmov. Po nadviazaní spojenia, napríklad prostredníctvom TCP/IP (Transmission Control Protocol/Internet Protocol), klient odošle *ClientHello* správu, ktorej štruktúra je znázornená v kóde 3.1 a na ktorú server reaguje odoslaním správy *ServerHello* [93]. Súčasťou oboch správ je premenná *extensions*, teda zoznam rozšírení, ktoré sa používajú na dodefinovanie ďalších parametrov komunikácie. Pomocou tohto zoznamu je možné do TLS jednoducho integrovať post-kvantové algoritmy prostredníctvom rozšírení *supported_groups* [93].

Zdrojový kód 3.1: Štruktúra *ClientHello*, ktorá sa používa na definovanie základných parametrov komunikácie zo strany klienta

```
1     struct {
2           ProtocolVersion legacy_version = 0x0303;
3           Random random;
4           opaque legacy_session_id<0..32>;
5           CipherSuite cipher_suites<2..2^16-2>;
6           opaque legacy_compression_methods<1..2^8-1>;
7           Extension extensions<8..2^16-1>;
8       } ClientHello;
```

Do TLS teda treba doplniť identifikačné čísla pre jednotlivé post-kvantové algoritmy, aby ich protokol dokázal rozpoznať a podľa toho použiť ďalšie funkcie pre konkrétny algoritmus.

Zároveň treba spomenúť aj možnú úpravu niektorých štruktúr a premenných. Veľkosti kľúčov pri použití ECDH algoritmu X25519 sú pomerne malé a odvíjajú sa od definície použitej eliptickej krivky [94], čo znamená, že súkromný kľúč má veľkosť 32 bajtov a verejný kľúč definovaný ako výsledok násobenia súkromného kľúča s náhodným bodom eliptickej krivky nad konečným poľom má tiež 32 bajtov. PQC algoritmy používajú pri generovaní kľúčov viacero parametrov, od ktorých potom závisia výsledné veľkosti kľúčov. Napríklad CRYSTALS-Kyber pri generovaní kľúčov vytvorí $(384k+32)$-bitový verejný kľúč a $(768k+96)$-bitový súkromný kľúč, kde k definuje dimenziu vektorového priestoru [82].

TLS aktuálne podporuje maximálnu veľkosť kľúča $2^{16}-1$ bajtov [93, str. 126]. Predbežný štandard ML-KEM založený na algoritme CRYSTALS-Kyber sa do tohto limitu zmestí, ale napríklad záložný algoritmus Classic McEliece prekračuje limit takmer 21-násobne. Porovnanie všetkých parametrov post-kvantových enkapsulačných algoritmov s klasickými ECDH algoritmami zobrazuje tabuľka 3.1.

[3] veľkosti kľúčov a šifrovaného textu sú rovnaké aj pre „f“ varianty algoritmov

Tabuľka 3.1: Porovnanie veľkostí kľúčov a šifrovaného textu bežne používaných algoritmov na výmenu kľúčov a vybraných post-kvantových enkapsulčných algoritmov [93],[95]

Algoritmus	**PQ**	**Veľkosť verejného kľúča [B]**	**Veľkosť súkromného kľúča [B]**	**Veľkosť šifrovaného textu [B]**
x25519	Nie	32	32	32
x448	Nie	56	56	56
secp256r1	Nie	64	32	64
secp384r1	Nie	97	48	97
Kyber512	Áno	800	1632	768
Kyber768	Áno	1184	2400	1088
Kyber1024	Áno	1568	3168	1568
BIKE-L1	Áno	1541	5223	1573
BIKE-L3	Áno	3083	10105	3115
BIKE-L5	Áno	5122	16494	5154
HQC-128	Áno	2249	2289	4481
HQC-192	Áno	44522	4562	9026
HQC-256	Áno	7245	7285	14469
Classic-McEliece 348864^3	Áno	261120	6492	96
Classic-McEliece 460896^3	Áno	524160	13608	156
Classic-McEliece 6688128^3	Áno	1044992	13932	208
Classic-McEliece 6960119^3	Áno	1047319	13948	194
Classic-McEliece 8192128^3	Áno	1357824	14120	208

3.2 Digitálne podpisy a ich overenie

Proces autentizácie prostredníctvom post-kvantových algoritmov pre digitálny podpis prebieha rovnako ako pri pre-kvantových algoritmoch ako je RSA alebo ECDSA [88],[66]. Už spomínaný zoznamom rozšírení *extensions* v hello paketoch obsahuje tiež rozšírenia označené ako *signature_algorithms_cert* a *signature_algorithms* [93, str. 41, 126]. Aj keď obe rozšírenia predstavujú zoznamy podporovaných schém pre digitálne podpisy, tak *Signature_algorithms_cert* definuje podpisovú schému pre konkrétny certifikát, zatiaľ čo *signature_algorithms* je zoznam všetkých podporovaných algoritmov. Ak *Signature_algorithms_cert* nie je definovaná, tak sa certifikát overuje na základe algoritmov v *signature_algorithms* [93]. Pre dosiahnutie podpory post-kvantových algoritmov pre digitálny podpis v TLS

je teda znovu potrebné doplniť ich identifikačné čísla.

Rozdielom oproti klasickým algoritmom sú znovu výrazne väčšie veľkosti verejných a súkromných kľúčov aj veľkosť podpisov, čo súvisí s celým procesom generovania kľúčov a vytvárania podpisu. Napríklad schémy pre digitálny podpis ed25519 a ed448 sú pri výpočtoch definované celkovo 11 parametrami, medzi ktorými je celočíselná hodnota b, ktorá musí spĺňať podmienku $2^{b-1} > p$, kde p je prvočíslo. Verejný kľúč má potom veľkosť b bitov a podpis $2b$ bitov. Podľa štandardu je b pevne definovaná ako $b = 256$ pre ed25519 a $b = 456$ pre ed448 [63, str. 28]. Pri PQC algoritmoch súvisia veľkosti kľúčov a podpisu s viacerými parametrami, napríklad CRYSTALS-Dilitihum generuje verejný kľúč s veľkosťou $32+32k(bitlen(q-1)-d)$ bitov a podpis s veľkosťou $32+32l(1+bitlen(\gamma_1-1))+\omega+k$ bitov, kde jednotlivé premenné závisia od použitého variantu algoritmu [82]. Na druhej strane algoritmus SPHINCS+ vďaka kompresii dosahuje porovnateľné veľkosti kľúčov s EdDSA, ale použitie stromovej štruktúry má za dôsledok výrazné zväčšenie podpisu [76]. Prehľad všetkých veľkostí kľúčov a podpisov sa nachádza v tabuľke 3.2. Napriek niekoľkonásobnému nárastu súkromných aj verejných kľúčov nie je pri PQC podpisových schémach potrebné upravovať pre-

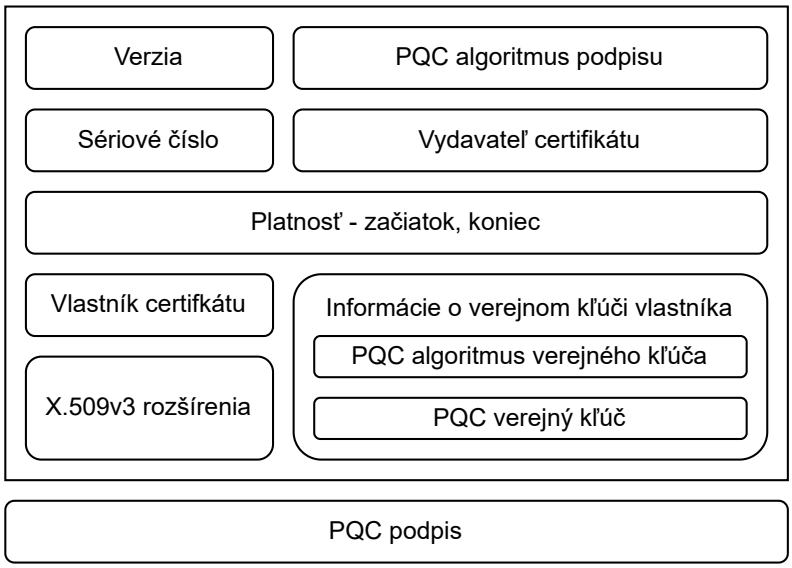
Tabuľka 3.2: Porovnanie veľkostí kľúčov a podpisov schém pre digitálne podpisy bežne používaných pre-kvantových algoritmov a predbežne štandardizovaných post-kvantových algoritmov [93],[96],[95]

Algoritmus	**PQ**	**Veľkosť verejného kľúča [B]**	**Veľkosť súkromného kľúča [B]**	**Veľkosť podpisu [B]**
RSA-2048	Nie	256	256	256
ecdsa_secp256r1	Nie	64	32	64
ed25519	Nie	32	32	64
ed448	Nie	57	57	114
Dilithium2	Áno	1312	2528	2420
Dilithium3	Áno	1952	4000	3293
Dilithium5	Áno	2592	4864	4595
FALCON-512	Áno	897	1281	752
FALCON-1024	Áno	1793	2305	1462
SPHINCS+-128f[4]	Áno	32	64	17088
SPHINCS+-128s[4]	Áno	32	64	7856
SPHINCS+-192f[4]	Áno	48	96	35664
SPHINCS+-192s[4]	Áno	48	96	16224
SPHINCS+-256f[4]	Áno	64	128	49856
SPHINCS+-256s[4]	Áno	64	128	29792

[4]veľkosti kľúčov a podpisov sú rovnaké pre SHA2 aj SHAKE varianty algoritmu

menné a štruktúry TLS, pretože TLS má definovanú maximálnu veľkosť podpisu 2^{16}-1 bajtov [88],[93, str. 132]. Rozdiely sa však môžu prejaviť na časoch podpisovania a overovania podpisu, ktoré súvisia s veľkosťami kľúčov a výpočtovou náročnosťou algoritmov [89].

S digitálnymi podpismi úzko súvisia aj certifikáty, ktoré majú v TLS štandardizovaný formát X.509 [93]. Štruktúra X.509 certifikátov je nezávislá od použitého algoritmu, takže doplniť do nej podporu pre post-kvantové algoritmy je pomerne jednoduché [88]. Štruktúra PQC X.509 certifikátu je znázornená na obrázku 3.3. Podobne ako pri TLS, tak aj pre X.509 štandard je dôležité zadefinovať identifikačné čísla X.509 objektov pre jednotlivé algoritmy. Maximálna veľkosť X.509 certifikátu, ktorú TLS definuje, je 2^{24}-1 bajtov [93, str. 132]. Napriek jednoduchej integrácii PQ algoritmov do TLS netreba zabúdať na to, že certifikáty sú vydávané certifikačnými autoritami, ktoré budú tiež musieť uskutočniť interné zmeny kvôli podpore post-kvantových algorimov.



Obr. 3.3: Štruktúra X.509 certifikátu s podporou pre PQC algoritmy [88]

4 Predbežné NIST štandardy pre post-kvantovú kryptografiu

Táto kapitola sa venuje detailom predbežných verzií vybraných NIST štandardov algoritmov post-kvantovej kryptografie. Aj keď nepredpokladáme výrazné zmeny v základných princípoch jednotlivých algoritmoch po oficiálnom vydaní štandardov, ktoré sa očakáva v priebehu roku 2024[1], tak čitateľ by mal brať do úvahy možné zmeny v detailoch implementácií a niektorých parametroch týchto algoritmov.

4.1 CRYSTALS-Kyber (ML-KEM)

Algoritmus ML-KEM (Module Lattice Key Encapsulation Mechanism) je post-kvantový algoritmus na enkapsuláciu kľúčov založený na algoritme CRYSTALS-Kyber, od ktorého sa líši len v niektorých detailoch, ktorým sa venujeme v ďalšej časti tejto podkapitoly. Názov ML-KEM bol definovaný zo strany NIST-u v predbežnej verzii štandardu FIPS 203 [81]. Pre jednoduchosť budeme v tejto podkapitole označovať ML-KEM a CRYSTALS-Kyber ako jeden algoritmus. Zároveň symboly použité v rámci tejto podkapitoly sme pre lepšiu orientáciu prebrali priamo zo štandardu.

CRYSTALS-Kyber dosahoval výborné výsledky v priebehu celej súťaže a spomedzi všetkých kandidátov, vrátane pomerne známeho algoritmu NTRU, bol vybraný na základe efektívneho generovania kľúčov, enkapsulácie a dekapsulácie, na základe rýchlej a optimalizovanej implementácie z pohľadu výpočtov a nárokov na pamäť a tiež kvôli možnosti škálovania celkovej úrovne bezpečnosti [69]. Kyber je prvotne konštruovaný ako schéma pre šifrovanie s verejným kľúčom, pričom je upravená na enkapsulačný algoritmus s IND-CPA bezpečnosťou aplikáciou Fujisaki-Okomoto transformácie [97],[81].

Bezpečnosť algoritmu ML-KEM je založená na MLWE probléme [81]. Ide o roz-

[1] ku dňu 12.04.2024 stále nevyšli

šírenú verziu LWE problému, ktorý si môžeme predstaviť ako sústavu rovníc s viacerými premennými, pričom ku každej rovnici je pripočítaná chyba, ktorá spôsobí, že sústavu nie je možné vypočítať konvenčnými spôsobmi, napríklad Gaussovou eliminačnou metódou [57]. Tento proces môžeme výrazne sťažiť použitím modulárnej aritmetiky nad štruktúrou cyklotomického polynomiálneho okruhu vo vektorom priestore [81]. Takto získame mriežku s definovanou dimenziou, rádom poľa a bodmi reprezentovanými formou vektorov ktorých súradnice sú tvorené polynómami. S dimenziou vektorového priestoru úzko súvisí celkové množstvo polynómov vo vektore na definovanie konkrétneho bodu v priestore.

4.1.1 Opis algoritmus CRYSTALS-Kyber

Cyklotomický polynomiálny okruh R môžeme definovať ako

$$R = \mathbb{Z}[X]/(X^n + 1),$$

kde n je mocnina čísla 2. V štandarde je táto hodnota fixne definovaná číslom 256 [81].

Polynomiálny okruh nad konečným poľom získame ako

$$R_q = \mathbb{Z}_q[X]/(X^n + 1)$$

alebo ako

$$\mathbb{GF}(q)[X]/(X^n + 1),$$

kde q je prvočíslo. V štandarde je táto hodnota fixne definovaná prvočíslom 3329 [81]. R_q je teda zložený z polynómov s koeficientami v $\mathbb{GF}(q)$ a najvyšším rádom $n - 1$ [57].

Vektorový priestor následne definujeme vzťahom

$$R_q^k = (\mathbb{Z}_q[X]/(X^n + 1))^k,$$

kde k predstavuje dimenziu vektorového priestoru [81].

Kľúče

Súkromný kľúč je zložený z vektoru polynómov $\mathbf{s} \in R_q^k$ obsahujúcich náhodné malé koeficienty a definovanou chybou $\mathbf{e} \in R_q^k$, ktorá má tiež formu vektora ktorého súradnice sú tvorené polynómami. Náhodné malé koeficienty sú generované na základe distribučnej funkcie λ_1 [69]. V literatúre sa môžeme stretnúť s definíciou, že súkromný kľúč je tvorený iba vektorom $\mathbf{s}$, každopádne aj chybový vektor

$\mathbf{e}$ musí ostať tajný [69], a preto môžeme povedať, že spoločne tvoria súkromný kľúč

$$SK = \{\mathbf{s}, \mathbf{e}\}.$$

Verejný kľúč je tvorený dvoma prvkami [69], maticou $\mathbf{A} \in R_q^{k \times k}$ a vektorom polynómov $\mathbf{t}$. Matica $\mathbf{A}$ je náhodne vygenerovaná na základe 256-bitovej počiatočnej hodnoty. Vektor $\mathbf{t}$ je daný vzťahom

$$\mathbf{t} = \mathbf{A}\mathbf{s} + \mathbf{e},$$

čo znamená, že verejný kľúč môžeme zapísať ako

$$VK = \{\mathbf{A}, \mathbf{t}\} = \{\mathbf{A}, \mathbf{A}\mathbf{s} + \mathbf{e}\}.$$

Šifrovanie

Pri každom šifrovaní sú náhodne vygenerované nové vektory polynómov $\mathbf{r}, \mathbf{e_1} \in R_q^k$ a polynóm $e_2 \in R_q$ [69]. Náhodné koeficienty sú generované na základe distribúcie λ_2 pseudonáhodnej funkcie [69]. Otvorený text M transformujeme do binárnej podoby, ktorú použijeme na vytvorenie binárneho polynómu m_b, kde koeficienty môžu nadobúdať hodnoty buď 0 alebo 1. Binárny polynóm následne vynásobíme hodnotou $\lceil \frac{q}{2} \rfloor$[2],

$$m = \left\lceil \frac{q}{2} \right\rfloor * m_b$$

teda celočíselnou hodnotou, ktorá je najbližšia k $\frac{q}{2}$. Táto hodnota slúži na zväčšenie koeficientov polynómu, čo je dôležité pri dešifrovaní správy [98]. Zároveň tým dosiahneme čiastočnú kompresiu správy [69]. Šifrovaný text C je prezentovaný hodnotami $(\mathbf{c_1}, c_2) \in R_q^k \times R_q$, teda vektorom polynómov $\mathbf{c_1}$ a polynómom c_2, ktoré vypočítame podľa vzťahu

$$C = (\mathbf{c_1}, c_2) = (\mathbf{A}^T\mathbf{r} + \mathbf{e_1}, \mathbf{t}^T\mathbf{r} + e_2 + m)$$

kde $\mathbf{A}^T$ predstavuje transponovanú maticu $\mathbf{A}$ a $\mathbf{t}^T$ transponovaný vektor $\mathbf{t}$.

Dešifrovanie

Proces dešifrovania šifrovaného textu C v tvare $(\mathbf{c_1}, c_2)$ môžeme vyjadriť v tvare

$$m_n = c_2 - \mathbf{c_1}\mathbf{s},$$

[2]symbol $\lceil \rfloor$ sa používa na označenie celočíselného zaokrúhlenia

kde m_n je označenie pre zašumený[3] polynóm [98],[99]. Aby sme pochopili prečo je tento polynóm zašumený, tak tento vzťah môžeme ďalej rozpísať podľa predchádzajúcich vzťahov do tvaru [99]

$$\begin{aligned} m_n &= c_2 - \mathbf{c_1 s} \\ &= \mathbf{t}^T\mathbf{r} + e_2 + m - \mathbf{s}(\mathbf{A}^T\mathbf{r} + \mathbf{e_1}) \\ &= (\mathbf{As} + \mathbf{e})^T\mathbf{r} + e_2 + \left\lceil \frac{q}{2} \right\rfloor m_b - \mathbf{s}(\mathbf{A}^T\mathbf{r} + \mathbf{e_1}) \\ &= \mathbf{e}^T\mathbf{r} + e_2 + m - \mathbf{se_1} \end{aligned}$$

Hodnoty $\mathbf{e}^T\mathbf{r}$, e_2 a $\mathbf{se_1}$ sú relatívne malé, pretože všetky chybové vektory boli generované s malými koeficientami. Naopak koeficienty v polynóme m sú pomerne veľké. Každý koeficient polynómu m_n následne „zaokrúhlime“ na číslo $\frac{q}{2}$ alebo 0, podľa toho ktoré sa nachádza bližšie [98],[99]. Koeficienty rovné číslu $\frac{q}{2}$ môžeme potom nahradiť jednotkou, nulové koeficienty ostanú bez zmeny. Týmto spôsobom získame pôvodný binárny polynóm m_b [98],[99].

Tento postup výpočtov môžu pripomínať skôr algoritmus na šifrovanie s verejným kľúčom než enkapsulačný algoritmus. To je spôsobené tým, že enkapsulačný algoritmus využíva pri niektorých výpočtoch funkcie šifrovacej schémy s verejným kľúčom [69],[81]. Tieto funkcie sa však neodporúča používať priamo na šifrovanie [81]. Hlavný rozdiel medzi šifrovacou schémou a enkapsulačným algoritmom je v tom, že enkapsulačný algoritmus nedovoľuje šifrovať konkrétny tajný kľúč, ale generuje ho počas enkapsulácie s využitím Fujisaki-Okomoto transformácie. S tým súvisia aj rozdiely v postupnosti výpočtov jednotlivých strán počas komunikácie.

Ako príklad šifrovania a dešifrovania môžeme použiť takzvaný Baby Kyber [98],[99], ktorý demonštruje všetky výpočty, ktoré sú realizované pri normálnej verzii algoritmu, ale s použitím malých čísel. Na výpočet použijeme $q = 17$ a $n = 4$, čo znamená, že na každý koeficient aplikujeme operáciu $(\mod 17)$ a na každý polynóm použijeme $(\mod(X^4 + 1))$.

Generovanie kľúčov

Zvolíme súkromný kľúč $\mathbf{s}$ a chybový vektor $\mathbf{e}$, pričom všetky polynómy budú obsahovať náhodne malé koeficienty

$$\begin{aligned} \mathbf{s} &= (-x^3 - x^2 + x, -x^3 - x) \\ \mathbf{e} &= (x^2, x^2 - x) \end{aligned}$$

[3] z ang. noisy

Vygenerujeme náhodnú maticu $\mathbf{A}$ zloženú z polynómov, ktorých koeficienty budú patriť do $\mathbb{GF}(q)$, teda $\mathbb{GF}(17)$. Rád všetkých polynómov bude maximálne $n-1$, teda 3.

$$\mathbf{A} = \begin{pmatrix} 6x^3 + 16x^2 + 16x + 11 & 9x^3 + 4x^2 + 6x + 3 \\ 5x^3 + 3x^2 + 10x + 1 & 6x^3 + x^2 + 9x + 15 \end{pmatrix}$$

Vypočítame $\mathbf{t} = \mathbf{As} + \mathbf{e}$

$$\mathbf{As} = \begin{pmatrix} 6x^3 + 16x^2 + 16x + 11 & 9x^3 + 4x^2 + 6x + 3 \\ 5x^3 + 3x^2 + 10x + 1 & 6x^3 + x^2 + 9x + 15 \end{pmatrix} * \begin{pmatrix} -x^3 - x^2 + x \\ -x^3 - x \end{pmatrix}$$

$$= \begin{pmatrix} -15x^6 - 26x^5 - 41x^4 - 18x^3 - x^2 + 8x \\ -11x^6 - 9x^5 - 23x^4 - 24x^3 - 14x \end{pmatrix}$$

$$\mathbf{t} = \mathbf{As} + \mathbf{e} = \begin{pmatrix} -15x^6 - 26x^5 - 41x^4 - 18x^3 - x^2 + 8x \\ -11x^6 - 9x^5 - 23x^4 - 24x^3 - 14x \end{pmatrix} + \begin{pmatrix} x^2 \\ x^2 - x \end{pmatrix}$$

$$= \begin{pmatrix} -15x^6 - 26x^5 - 41x^4 - 18x^3 + 8x \\ -11x^6 - 9x^5 - 23x^4 - 24x^3 + x^2 - 15x \end{pmatrix}$$

Na všetky koeficienty aplikujeme operáciu *mod 17* a na každú mocninu použijeme operáciu *mod* $(X^4 + 1)$, vyjde nám

$$\mathbf{t} = \begin{pmatrix} 16x^3 + 15x^2 + 7 \\ 10x^3 + 12x^2 + 11x + 6 \end{pmatrix}$$

Takže naše kľúče sú

$$\text{Verejný kľúč VK } = (\mathbf{A}, \mathbf{t})$$

$$\text{Súkromný kľúč SK } = (\mathbf{s}, \mathbf{e})$$

Šifrovanie

Zvolíme náhodné vektory polynómov $\mathbf{r}, \mathbf{e_1}$ a polynóm e_2 s malými koeficientami. Tieto parametre generujeme pri každom šifrovaní, takže pri opakovanom šifrovaní rovnakého textu získame rozdielne výsledky.

$$\mathbf{r} = (-x^3 + x^2, x^3 + x^2 - 1)$$
$$\mathbf{e_1} = (x^2 + x, x^2)$$
$$e_2 = (-x^3 - x^2)$$

Pretože sme na začiatku určili, že $n = 4$, tak môžeme šifrovať maximálne 4-bitové slovo. V tomto príklade budeme šifrovať číslo $11_{(10)}$, ktoré má v dvojkovej

číselnej sústave zápis $1011_{(2)}$. Binárny zápis čísla použijeme na vytvorenie binárneho polynómu m_b, ktorý bude reprezentovať naše slovo.

$$m_b = 1x^3 + 0x^2 + 1x + 1 = x^3 + x + 1$$

Na šifrovanie však použijeme rozšírený polynóm m, ktorý získame vzťahom

$$\begin{aligned} m &= \lceil \frac{q}{2} \rfloor m_b = \lceil \frac{17}{2} \rfloor (x^3 + x + 1) \\ &= 9x^3 + 9x + 9 \end{aligned}$$

Šifrovanie realizujeme výpočtom

$$\begin{aligned} C &= (\mathbf{c_1}, c_2) \\ \mathbf{c_1} &= \mathbf{A}^T\mathbf{r} + \mathbf{e_1} \\ c_2 &= \mathbf{t}^T\mathbf{r} + e_2 + m \end{aligned}$$

$$\begin{aligned} \mathbf{A}^T\mathbf{r} &= \begin{pmatrix} 6x^3 + 16x^2 + 16x + 11 & 5x^3 + 3x^2 + 10x + 1 \\ 9x^3 + 4x^2 + 6x + 3 & 6x^3 + x^2 + 9x + 15 \end{pmatrix} * \begin{pmatrix} -x^3 + x^2 \\ x^3 + x^2 - 1 \end{pmatrix} \\ &= \begin{pmatrix} -x^6 - 2x^5 + 13x^4 + 11x^3 + 9x^2 - 10x - 1 \\ -3x^6 + 12x^5 + 8x^4 + 21x^3 + 17x^2 - 9x - 15 \end{pmatrix} \end{aligned}$$

$$\begin{aligned} \mathbf{c_1} &= \mathbf{A}^T\mathbf{r} + \mathbf{e_1} \\ &= \begin{pmatrix} -x^6 - 2x^5 + 13x^4 + 11x^3 + 9x^2 - 10x - 1 \\ -3x^6 + 12x^5 + 8x^4 + 21x^3 + 17x^2 - 9x - 15 \end{pmatrix} + \begin{pmatrix} x^2 + x \\ x^2 \end{pmatrix} \\ &= \begin{pmatrix} -x^6 - 2x^5 + 13x^4 + 11x^3 + 10x^2 - 9x - 1 \\ -3x^6 + 12x^5 + 8x^4 + 21x^3 + 18x^2 - 9x - 15 \end{pmatrix} \end{aligned}$$

Vypočítame

$$\begin{aligned} \mathbf{c_1} &= \mathbf{c_1} \bmod (x^4 + 1) \\ &= \begin{pmatrix} -x^6 - 2x^5 + 13x^4 + 11x^3 + 10x^2 - 9x - 1 \\ -3x^6 + 12x^5 + 8x^4 + 21x^3 + 18x^2 - 9x - 15 \end{pmatrix} \bmod (x^4 + 1) \\ &= \begin{pmatrix} x^2 + 2x - 13 + 11x^3 + 10x^2 - 9x - 1 \\ 3x^2 - 12x - 8 + 21x^3 + 18x^2 - 9x - 15 \end{pmatrix} \\ &= \begin{pmatrix} 11x^3 + 11^2 - 7x - 14 \\ 21x^3 + 21x^2 - 21x - 23 \end{pmatrix} \end{aligned}$$

Ďalej vypočítame

$$\mathbf{c_1} = \mathbf{c_1} \bmod (17)$$
$$= \begin{pmatrix} 11x^3 + 11^2 - 7x - 14 \\ 21x^3 + 21x^2 - 21x - 23 \end{pmatrix} \bmod (17)$$
$$= \begin{pmatrix} 11x^3 + 11^2 + 10x + 3 \\ 4x^3 + 4x^2 + 13x + 11 \end{pmatrix}$$

Následne postupne vypočítame hodnotu c_2

$$\mathbf{t}^T\mathbf{r} = \begin{pmatrix} 16x^3 + 15x^2 + 7 & 10x^3 + 12x^2 + 11x + 6 \end{pmatrix} * \begin{pmatrix} -x^3 + x^2 \\ x^3 + x^2 - 1 \end{pmatrix}$$
$$= -6x^6 + 23x^5 + 38x^4 + x^2 - 11x - 6$$

$$c_2 = \mathbf{t}^T\mathbf{r} + e_2 + m$$
$$= (-6x^6 + 23x^5 + 38x^4 + x^2 - 11x - 6) + (-x^3 - x^2) + (9x^3 + 9)$$
$$= (-6x^6 + 23x^5 + 38x^4 + 8x^3 - 11x + 3)$$

Nakoniec vypočítame

$$c_2 = [(-6x^6 + 23x^5 + 38x^4 + 8x^3 - 11x + 3) \bmod (x^4 + 1)] \bmod 17$$
$$= 8x^3 + 6x^2 + 12x + 16$$

C bude vyzerať

$$C = (\mathbf{c_1}, c_2)$$
$$(C = (\begin{pmatrix} 11x^3 + 11^2 + 10x + 3 \\ 4x^3 + 4x^2 + 13x + 11 \end{pmatrix},\ 8x^3 + 6x^2 + 12x + 16)$$

Dešifrovanie

Pri dešifrovaní použijeme súkromný kľúč a šifrovaný text na výpočet „zašumenej" správy m_n

$$m_n = c_2 - \mathbf{s}^T\mathbf{c_1}$$

$$\mathbf{s}^T\mathbf{c_1} = \begin{pmatrix} -x^3 - x^2 + x & -x^{-}x \end{pmatrix} * \begin{pmatrix} 11x^3 + 11x^2 + 10x + 3 \\ 4x^3 + 4x^2 + 13x + 11 \end{pmatrix}$$
$$= -15x^6 - 26x^5 - 27x^4 - 17x^3 - 6x^2 - 8x$$

Zredukujeme mocniny a koeficienty pomocou modulárnej aritmetiky operáciami *mod 17* a *mod* $(X^4 + 1)$

$$\mathbf{s}^T\mathbf{c_1} = 9x^2 + x + 10$$

$$m_n = (8x^3 + 6x^2 + 9x + 16) - (9x^2 + x + 10)$$

$$= 8x^3 - 3x^2 + 8x + 6$$

Upravíme do finálnej podoby

$$m_n = 8x^3 + 14x^2 + 8x + 6$$

Koeficienty našej zašumenej správy zaokrúhlime podľa

$$round(a_n) = \begin{cases} 0 & \text{ak } a_n \text{ je bližšie k číslu 0 alebo 17} \\ 9 & \text{ak } a_n \text{ je bližšie k číslu } \lceil \frac{17}{2} \rfloor, \text{ teda k 9} \end{cases}$$

$$m = 9x^3 + 0x^2 + 9x + 9$$

Vypíšeme koeficienty a transformujeme ich na bity

$$9099 \rightarrow 1011$$

Získali sme našu pôvodnú správu 1011 v binárnom zápise.

Ako sme na začiatku uviedli, na názornú ukážku základných matematických operácii algoritmu sme použili vzorový príklad zo zdroja [98], ktorý sme pre lepšie pochopenie pomerne detailne rozpísali. Napriek tomu, že nám vyšla správna dešifrovaná správa, tak čitateľ si môže všimnúť rozdielne výsledky niektorých medzivýpočtov. Tieto rozdiely sme si všimli aj my a to aj napriek tomu, že naše ručné výpočty potvrdzujú výpočty online nástrojov ako Wolfram Alpha[4] a Matrix calculator[5].

Na konci príkladu musíme tiež upozorniť na to, že pri použití iného otvoreného textu pri šifrovaní s takto nastavenými vstupnými parametrami by sme nemuseli na konci výpočtu získať korektný dešifrovaný text [99]. Tento fakt je spôsobený pomerom chybovosti algoritmu, ktorý sa odvíja od nastavených parametrov, ktorým sa ďalej venujeme v nasledujúcej podkapitole. Zároveň nami zvolené parametre umožňovali realizovať pomerne jednoduché výpočty. Avšak s nárastom celkového množstva vektorov a dĺžkou polynómov s maximálnym rádom 255 sa výrazne zvyšuje náročnosť výpočtov. Aj keď sčítanie môže byť ešte

[4] `https://www.wolframalpha.com/`

[5] `https://matrixcalc.org/`

relatívne jednoduché, tak násobenie matíc a polynómov je výpočtovo veľmi náročné. Pre zvýšenie efektívnosti týchto výpočtov využíva algoritmus Kyber NTT (Number Theoretic Transform) transformáciu, čo je špecifická verzia diskrétnej Fourierovej transformácie. Polynómy a vektory sú transformované do NTT domény nad konečným poľom, kde je realizovaná veľká časť výpočtov a následne je výsledok spätne transformovaný [81]. Okrem toho sú polynómy veľmi dlhé, takže je na nich pri šiforvaní aplikovaná kompresia odstránením menej významných bitov. Pri dešifrovaní sú následne menej významné bity pridané pri procese dekompresie na získanie pôvodného polynómu [81].

4.1.2 Špecifikácia algoritmu

Algoritmus ML-KEM je špecifikovaný celkovo v troch verziách, pričom ich označenie ostalo rovnaké ako pri pôvodnom algoritme CRYSTALS-Kyber [81]:

- ML-KEM-512,
- ML-KEM-768,
- ML-KEM-1024.

Na prvý pohľad môžeme vidieť rozdiely medzi jednotlivými verziami algoritmu vo forme veľkostí kľúčov a šifrovaného textu, ktoré sú zobrazené v tabuľke 4.1. Tie však závisia od premenných, ktoré sa používajú pri výpočtoch a ktoré musíme primerane definovať s cieľom dosiahnuť požadovanú úroveň bezpečnosti. Tabuľka 4.2 zobrazuje špecifikácie parametrov pre všetky verzie algoritmu.

NIST odporúča používať ML-KEM-768, kvôli vhodnému pomeru bezpečnosti a výpočtovej náročnosti [81]. Ostatné verzie sú vhodné pre špecifické zariadenia a aplikácie v závislosti od dostupného výpočtového výkonu a od požiadavok na úroveň bezpečnosti. Konštanty n a q definujú mriežku nad konečným

Tabuľka 4.1: Porovnanie veľkostí kľúčov a šifrovaného textu rôznych verzií algoritmu CRYSTALS-Kyber, resp. ML-KEM [81]

Algoritmus	**Verejný kľúč [B]**	**Súkromný kľúč [B]**	**Šifrovaný text [B]**	**Zdieľaný tajný kľúč [B]**
ML-KEM-512	800	1632	768	32
ML-KEM-768	1184	2400	1088	32
ML-KEM-1024	1568	3168	1568	32

poľom. Parameter k predstavuje dimenziu vektorového priestoru. Premenná λ_1,

Tabuľka 4.2: Parametre používané pri výpočtoch generovania kľúčov, enkapsulácii a dekapsulácii rôznych verzií algoritmu CRYSTALS-Kyber, resp. ML-KEM [81]

Algoritmus	**Úroveň bezpečnosti**	n	q	k	λ_1	λ_2	d_u	d_v	RBG
ML-KEM-512	L1	256	3329	2	3	2	10	4	128
ML-KEM-768	L3	256	3329	3	2	2	10	4	192
ML-KEM-1024	L5	256	3329	4	2	2	11	5	256

resp. λ_2 špecifikuje distribúciu náhodných koeficientov pri generovaní vektorov s a e, resp. $\mathbf{e_1}$ a $\mathbf{e_2}$. Premenné d_u a d_v sa používajú pri kompresii a dekompresii [81]. RGB definuje požadovanú bitovú bezpečnosť použitého náhodného generátora bitov (RBG – Random Bit Generator). Pri porovnaní parametrov si môžeme všimnúť, že niektoré premenné sú konštantné, zatiaľ čo niektoré sa menia len minimálne. Toto je jeden z dôvodov, prečo je algoritmus Kyber relatívne jednoduché implementovať a zároveň meniť úroveň bezpečnosti alebo inak prispôsobovať algoritmus v závislosti od konkrétneho zariadenia alebo aplikácie prostredníctvom premenných.

Za dôležité tiež považujeme spomenúť chybovosť algoritmu, ktorú sme naznačili na konci príkladu v predošlej podkapitole. Zatiaľ čo proces enkapsulácie a dekapsulácie vždy vráti správny formát výstupných dát, tak sa môže stať, že účastník A získa iný tajný kľúč ako účastník B. Tento proces môžeme opísať algoritmom 1.

Algoritmus 1 Zjednodušený pohľad na proces enkapsulácie a dekapsulácie s chybným výsledkom [81]

1: (ek, dk) ← ML-KEM.KeyGen()
2: (c, K) ← ML-KEM.Encaps(ek)
3: K′ ← ML-KEM.Decaps(c, dk)
4: K $\neq$ K′

Chybovosť procesu enkapsulácie a dekapsulácie závisí od zvolených parametrov [81], predovšetkým od distribúcie náhodných koeficientov a tiež od veľkosti prvočísla q, s ktorou pracujeme pri zašumenom polynóme reprezentujúcom našu správu. Tabuľka 4.3 obsahuje chybovosť pre jednotlivé verzie algoritmu. Vysoká chybovosť algoritmu s použitím malých čísel bola demonštrovaná v článku [99], kde autor použil tri rôzne hodnoty q, pričom zvyšovaním hodnoty sa znižoval pomer chybovosti.

Tabuľka 4.3: Chybovosť dekapsulácie algoritmu ML-KEM [81]

Algoritmus	Pomer chybovosti
ML-KEM-512	2^{-139}
ML-KEM-768	2^{-164}
ML-KEM-1024	2^{-174}

4.1.3 Bezpečnosť algoritmu

Bezpečnosť algoritmu je založená na MLWE probléme [81], spojením LWE problému a SVP problému v mriežke. Predpokladá sa, že MLWE problém by mal byť rovnako náročný na výpočet ako riešenie SVP problému vo vhodne definovanom vektorovom priestore.

Kryptoanalýza celého algoritmu je pomerne náročná a skrýva sa za ňou veľké množstvo matematických výpočtov a analýz. Každopádne autori tvrdia, že samotný MLWE problém sa dá pri kryptoanalýze rozdeliť na samostatné problémy LWE a SVP, na ktoré vieme aplikovať rôzne postupy a pokúsiť sa tak o nalomenie celého algoritmu. Na LWE problém je možné použiť takzvaný primal útok a duálny útok[6] [60], ktoré využívajú rôzne triediace algoritmy, matematické štruktúry, redukcie a optimalizačné výpočty, pričom ich úspešnosť je podmienená špecifickými kritériami. Na SVP problém sa vzťahuje BKZ (Block-Korkine-Zolotarev) algoritmus [100], ktorý však dosahuje exponenciálnu časovú zložitosť aj na kvantových počítačoch [61]. Týmto útokom sa nebudeme detailne zaoberať, záujemcovia o túto problematiku sa môžu dočítať viac v odborných publikáciach [101],[102],[103],[104],[105].

Okrem toho algoritmus používa hašovacie funkcie SHAKE-128, SHA3-256 a SHA3-512, u ktorých autori predpokladajú ich bezpečnosť a odolnosť voči kolízii [60].

Pri bezpečnostnej analýze sa tiež spomínajú vybrané útoky s využitím postranných kanálov. Konkrétne ide o časový útok, diferenčný útok a šablónový útok[7]. Detailné zhrnutie a analýzu všetkých útokov je možné nájsť v [106].

Referenčné implementácie Kyberu[8] neobsahujú žiadne formy vyhľadávania v tabuľkách, ktoré sú často používané pri časových útokoch [60]. Autori tiež brali do úva-hy aj násobičky s nekonštantným časom, ktoré však nepokladajú za bezpečnostné riziko, keďže algoritmus využíva relatívne malé čísla [60]. Na operácie

[6] z ang. primal attack
[7] z ang. template attack
[8] `https://github.com/pq-crystals/kyber`

modulárnej aritmetiky je zase aplikovaná Montgomeryho a Barrettova redukcia [60]. Koncom roka 2023 však niekoľko výskumníkov[9][10], vrátane D. J. Bernsteina, upozornilo na bezpečnostnú chybu v mnohých implementáciách algoritmu, ktorú označili menom KyberSlash a ktorá umožňovala získať tajný kľúč z časovej analýzy enkapsulácie [107]. Chyby sa nachádzali v zdrojových kódoch rôznych knižníc, vrátane tých, ktoré experimentálne používali spoločnosti ako Amazon alebo Signal Messenger LLC. Po zverejnení informácii o bezpečnostnom riziku začali správcovia jednotlivých knižníc s úpravami problémových častí kódov. Profesor Bernstein vytvoril webovú stránku[11], ktorá obsahuje stručné zhrnutie problémových častí zdrojových kódov spolu so zoznamom knižníc, ktoré označili staršiu verziu za rizikovú a do novej verzie doplnili bezpečnostnú záplatu. V zozname sa tiež nachádzajú knižnice, ktoré stále obsahujú chyby a predstavujú tak bezpečnostné riziko.

V prípade diferenčných útokov na základe merania spotreby (DPA – Differential Power Analysis) alebo elektromagnetického žiarenia (DEMA - Differential Electromagnetic Analysis) autori predpokladajú vstavanú ochranu zariadenia vo forme maskovania vyššieho rádu [60]. Začiatkom roka 2023 sa začali šíriť správy o tom, že švédski výskumníci prelomili Kyber použitím diferenčného útoku a umelej inteligencie, ktorá bola trénovaná na analýzu maskovaných dát. Prelomená však bola konkrétna implementácia algoritmu s maskovaním piateho rádu, nie celý algoritmus ako taký [108],[109],[110]. Detaily k podmienkam, priebehu a výsledkom útoku je možné nájsť v publikácii [111].

Šablónový útok Kyberu sa spomína v súvislosti s modelom útoku [112], zloženom z časovej analýzy a DPA, ktorý bol použitý na prelomenie šifrovacej schémy založenej na probléme v mriežke. Autori tohto útoku označili metódy maskovania a realizáciu výpočtov v konštantnom čase za hlavné formy protiopatrení [112], ktoré implementácia Kyberu obsahuje, avšak jeho tvorcovia zdôrazňujú potrebu ďalšej analýzy celého modelu útoku a jeho možné bezpečnostné dôsledky na celý algoritmus [60].

4.2 CRYSTALS-Dilithium (ML-DSA)

ML-DSA (Module Lattice Digital Signature Algorithm) je jedným z PQC algoritmov pre digitálny podpis, založený na algoritme CRYSTALS-Dilithium. Názov

[9] https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/ldX0ThYJuBo

[10] https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/hWqFJCucuj4/m/-Z-jm_k9AAAJ

[11] https://kyberslash.cr.yp.to/

ML-DSA zaviedol NIST v predbežnej verzii štandardu FIPS 204 [82]. ML-DSA sa od pôvodného algoritmu líši len špecifikácii niektorých parametrov, ktorým sa venujeme pri analýze algoritmu, pričom pre lepšiu orientáciu používame symboly priamo zo štandardu. Okrem toho zámerne neudávame pseudokódy pre jednotlivé funkcie, pretože sú súčasťou štandardu [82]. V rámci tejto podkapitoly preto budeme označovať ML-DSA a Dilithium ako jeden algoritmus.

CRYSTALS-Dilithium je algoritmus pre digitálne podpisy, ktorý poskytuje SUF-CMA (Strong Existential Unforgeability under Chosen-Message Attack) úroveň bezpečnosti [82]. Je založený na MLWE probléme, podobne ako enkapsulačný algoritmus CRYSTALS-Kyber. To zjednodušene znamená, že pri výpočtoch budeme využívať polynómy, ktoré budú predstavovať body vo vektorovom priestore obmedzenom definovaným konečným poľom a budeme k nim pripočítavať generované chyby. Overovanie podpisov využíva heuristickú Fiat-Shamirovu schému [82].

4.2.1 Opis algoritmu CRYSTALS-Dilithium

Polynomiálny okruh pre $n, q \in \mathbb{N}$ definujeme ako

$$R_q = \mathbb{Z}_q[X]/(X^n + 1)$$

alebo ako

$$GF(q)[X]/(X^n + 1)$$

kde *q* je prvočíslo a *n* je väčšinou mocnina čísla 2. Algoritmus používa $q = 2^{23} - 2^{13} + 1 = 8380417$ a $n = 256$ [82]. Najvyšší rád polynómov bude $n - 1$ s koeficientami v $\mathbb{GF}(q)$. Vektorový priestor potom môžeme ohraničiť vzťahom [113]

$$R_q^l = \mathbb{Z}_q[X]/(X^n + 1),$$

kde $l \in \mathbb{N}$ definuje dimenziu vektorového priestoru.

Generovanie kľúčov

Proces generovania súkromného a verejného kľúča je veľmi podobný ako pri algoritme CRYSTALS-Kyber. Zjednodušene môžeme povedať, že súkromný kľúč je zložený z vektorov polynómov $\mathbf{s_1} \in R_q^l$, $\mathbf{s_2} \in R_q^k$[12] [82]. Koeficienty polynómov vo vektoroch sú malé, v rozmedzí $\langle -\eta, \eta \rangle$. Hodnota η závisí od verzie algoritmu, ktorým sa venujeme v ďalej podkapitole. Súkromný kľúč má potom tvar

$$SK = \{s_1, s_2\}.$$

[12] V literatúre môžeme nájsť túto hodnotu označenú ako *e*, čím to výrazne pripomína generovanie kľúčov pri algoritme CRYSTALS-Kyber [113].

V skutočnosti je súkromný kľúč tvorený viacerými hodnotami, ktoré sú na konci výpočtov kódované do výsledného súkromného kľúča s celkovou dĺžkou $(32 + 32+64+32)((l+k)\lceil \log_2(2\eta)\rceil + 13k)$ bajtov [82]. Súčasťou súkromného kľúča je verejná hodnota random seedu ρ, z ktorého je odvodená matica A verejného kľúča, 256-bitová súkromná počiatočná hodnota K, 512-bitový haš verejného kľúča označený ako tr, vektory $\mathbf{s_1}, \mathbf{s_2}$ a vektor polynómov $\mathbf{t_0}$, ktorý obsahuje $d = 13$ najmenej významných bitov verejného kľúča pred kompresiou [82]. Napriek tomu, že niektoré hodnoty sú verejné, tak súkromný kľúč môžeme zapísať aj v tvare

$$SK = \{\rho, K, tr, \mathbf{s_1}, \mathbf{s_2}, \mathbf{t_0}\}$$

Verejný kľúč môžeme v jednoduchosti opísať maticou polynómov $\mathbf{A} \in R_q^{k \times l}$ a vektorom $\mathbf{t}$ [82], ktorý vypočítame vzťahom

$$\mathbf{t} = \mathbf{A}\mathbf{s_1} + \mathbf{s_2}$$

Potom môžeme verejný kľúč vyjadriť ako

$$VK = \{\mathbf{A}, \mathbf{t}\} = \{\mathbf{A}, \mathbf{A}\mathbf{s_1} + \mathbf{s_2}\}.$$

Takýto zápis je označený ako rozšírená forma verejného kľúču [82], pretože reálne je verejný kľúč $(32 + 320k)$-bajtová hodnota zložená z verejnej počiatočnej hodnoty ρ, na základe ktorej je pseudonáhodne generovaná matica $\mathbf{A}$, a komprimovaného vektoru polynómov $\mathbf{t_1}$, ktorý získame odstránením $d = 13$ najmenej významných bitov zo všetkých koeficientov polynómov vektora $\mathbf{t}$ [82]. Verejný kľúč má potom tvar

$$VK = \{\rho, \mathbf{t_1}\}$$

Generovanie podpisu

Pri procese podpisovania je správa *M* spojená s hodnotou hašu tr zo súkromného kľúča a použitá ako vstup pre hašovaciu funkciu SHAKE256 [82]. Výsledkom je 512-bitový haš správy označený ako μ. Štandard však počíta aj s možnosťou, že zariadenie hardvérovo nepodporuje SHAKE256, a preto je možné podpisovať otlačok správy, ktorý vytvoríme inou hašovacou funkciou s minimálnou λ[13]-bitovou bezpečnosťou [82].

Hlavná časť generovania digitálneho podpisu je pravdepodobnostná, založená na cykle odmietnutia vzorkovania[14], podľa Fiat-Shamirovej schémy s prerušeniami [82],[113]. Algoritmus je tak svojou štruktúrou dosť podobný Schnorrovým podpisom, napríklad EdDSA [63]. Cyklus obsahuje výpočet na generovanie

[13] presná hodnota závisí od verzie algoritmu, viď tabuľka 4.4

[14] z ang. rejection sampling loop

podpisu, pričom časť vytvoreného podpisu môže obsahovať štatistickú závislosť od súkromného kľúča, čo by viedlo k úniku informácií a bezpečnostnému riziku. Takýto podpis je preto označený za neplatný a cyklus pokračuje až dokým nie je vytvorený platný podpis [82]. Predpokladaný počet opakovaní cyklu závisí od vstupných parametrov definovaných pri generovaní podpisu.

Na začiatku každého cyklu náhodne vygenerujeme maskovací vektor $\mathbf{y} \in R_q^l$ s pomerne malými koeficientami, v rozmedzí $\langle -\gamma_1 + 1, \gamma_1 \rangle$ [82],[113]. Veľkosť hodnoty γ_1 závisí od verzie použitého algoritmu, ktorým sa venujeme v nasledujúcej podkapitole. Následne vypočítame hodnotu $\mathbf{w}$, ktorá je označovaná ako „záväzok[15]“ [82], podľa vzťahu

$$\mathbf{w} = \mathbf{A}\mathbf{y},$$

pričom koeficienty výsledného polynómu zaokrúhlime k najbližšiemu násobku čísla $2\gamma_2$, ktorá je definovaná podľa zvolenej verzie algoritmu. Výsledok po zaokrúhlení môžeme označiť ako $\mathbf{w_1}$ [82].
Hodnoty $\mathbf{w_1}$ a μ spojíme a použijeme ako vstup pre hašovaciu funkciu, ktorej výstupom je haš $\tilde{c}$. Potom premenná $\tilde{c}_1$ predstavuje prvých 256 bitov hašu $\tilde{c}$, ktoré použijeme na vygenerovanie náhodného polynómu, označovaného ako „výzva[16]“ [82], $\mathbf{c} \in R_q$ s koeficientmi -1, 0, 1 a Hammingovou váhou τ [82]. Následne vypočítame podľa vzťahu

$$\mathbf{z} = \mathbf{y} + c\mathbf{s_1}.$$

Ak je niektorý koeficient polynómu $\mathbf{z}$ väčší ako $\gamma_1 - \beta$ alebo ak je niektorý koeficient bitov nízkeho rádu polynómu $\mathbf{w} - c\mathbf{s_2}$ väčší ako $\gamma_2 - \beta$, tak je podpis označený za neplatný a cyklus začne odznovu [82]. Po splnení podmienok môžeme vytvoriť vektor $\mathbf{h} \in R_2^k$, označovaný ako „nápoveda[17]“, ktorý bude obsahovať najviac ω nenulových koeficientov [82]. Ten nám pri overovaní podpisu umožní vytvoriť $\mathbf{w_1}$ na základe verejného kľúča.
Finálny podpis bude mať tvar $(\tilde{c}, \mathbf{z}, \mathbf{h})$

Overenie podpisu

Pri overovaní podpisu vypočítame polynóm $\mathbf{c}$ z hašu $\tilde{c}$, podobne ako pri tvorbe podpisu. Potom vypočítame približnú hodnotu $\mathbf{w}'$ vzťahom

$$\mathbf{w}' = \mathbf{A}\mathbf{z} - c\mathbf{t_1}2^d.$$

[15] z ang. commitment
[16] z ang. challenge
[17] z ang. hint

Ak je podpis správny, tak platí

$$\mathbf{w} = \mathbf{Ay} = \mathbf{Az} - c\mathbf{t} + c\mathbf{s_2} \approx \mathbf{w'} = \mathbf{Az} - c\mathbf{t_1}2^d,$$

pretože c a $\mathbf{s_2}$ obsahujú malé koeficienty a $\mathbf{t_1}2^d \approx \mathbf{t}$ [82]. Na základe hodnôt $\mathbf{h}$ a $\mathbf{w'}$ získame $\mathbf{w'_1}$. Na záver overíme, že koeficienty v $\mathbf{z}$ sa nachádzajú v rozmedzí $\langle -\gamma_1 - \beta, \gamma_1 - \beta \rangle$, či vektor $\mathbf{h}$ obsahuje najviac ω nenulových koeficientov a tiež či pôvodný haš $\tilde{c}$ je zhodný s hodnotou hašu $\tilde{c}'$ vytvorenou spojením μ a $\mathbf{w'_1}$ [82].

4.2.2 Špecifikácia algoritmu

ML-DSA je dostupný v troch variantoch, pričom okrem samotného názvu bol zmenený aj význam číselnej prípony v názve algoritmu. Zatiaľ čo pôvodne názvy Dilithium2, Dilithium3 a Dilithium5 odzrkadľovali predpokladanú úroveň bezpečnosti [61], teda L2, L3 a L5, tak čísla x a y v názve ML-DSA-xy predstavujú dimenziu matice A vo verejnom kľúči [82]. Štandard FIPS 204 definuje tri verzie algoritmu ML-DSA [82]:

- ML-DSA-44,
- ML-DSA-65,
- ML-DSA-87.

Okrem dimenzie matice sa verzie algoritmu líšia aj v ďalších parametroch, ktoré sa využívajú pri výpočtoch [82]. Definované parametre sú vhodne zvolené tak, aby algoritmus dosahoval požadovanú bezpečnosť a zároveň bol dostatočné optimalizovaný z pohľadu výpočtov, veľkostí kľúčov a veľkosti podpisu. Porovnanie parametrov jednotlivých verzií algoritmu zobrazuje tabuľka 4.4.

Z pohľadu implementácie môžeme dodatočne rozdeliť proces generovania podpisov na uzavretý[18], resp. náhodný a deterministický [82],[61]. Celý proces generovania podpisu ostáva rovnaký, rozdiel je iba na spôsobene odvodenia počiatočnej hodnoty, na základe ktorého sa generuje maskovací vektor $\mathbf{y}$ [82]. V prípade náhodnej verzie sa počiatočná hodnota počíta, okrem iných hodnôt, aj na základe 256-bitovového vstupu vygenerovaného náhodným generátorom čísel, zatiaľ čo deterministická verzia používa 256 bitovú hodnotu zloženú z núl [82]. Odporúča sa používať náhodnú verziu, pretože aj keď je deterministická verzia vhodná pre zariadenia, ktoré neobsahujú náhodné generátory čísel, tak jej použitie predstavuje bezpečnostné riziko spojené s útokmi cez postranné kanály [82], ktorým sa venujeme v nasledujúcej podkapitole.

[18] označené ang. hedged

Tabuľka 4.4: Prehľad parametrov používaných procesoch generovania kľúčov, generovania digitálnych podpisov a overovania podpisov pre rôzne verzie algoritmu CRYSTALS-Dilitihum, resp. ML-DSA [82]

Algoritmus / Parameter	ML-DSA-44	ML-DSA-65	ML-DSA-87
Úroveň bezpečnosti	L2	L3	L5
q	8380417	8380417	8380417
n	256	256	256
η	2	4	2
$k; l$	4; 4	6; 5	8; 7
d	13	13	13
γ_1	2^{17}	2^{19}	2^{19}
γ_2	$(q\text{-}1)/88$	$(q\text{-}1)/32$	$(q\text{-}1)/32$
τ	39	49	60
λ	128	192	256
β	78	196	120
ω	80	55	75
n	4,25	5,1	3,85

Konštanty n a q definujú vektorový priestor mriežky. Premenná η určuje rozsah koeficientov pri generovaní vektorov $\mathbf{s_1}$ a $\mathbf{s_2}$ súkromného kľúča. Čísla k, l opisujú dimenziu matice $\mathbf{A}$. Parameter d označuje počet najmenej významných bitov, ktoré sú odstránené z koeficientov polynómov pri kompresii. Hodnota γ_1 udáva rozsah koeficientov polynómov vo vektore y, γ_2 je zase referenčná hodnota, na základe ktorej sa zaokrúhľujú koeficienty $\mathbf{w_1}$ [82]. Parameter τ je Hammingova váha, ktorá určuje počet nenulových koeficientov v polynóme $\mathbf{c}$. λ predstavuje minimálny počet bitov, resp. odolnosť voči kolízii, na dosiahnutie potrebnej bezpečnosti, ak chceme pri generovaní podpisu kvôli podpore alebo lepšej optimalizácii použiť inú hašovaciu funkciu ako SHAKE256 [82]. Premenná β udáva rozsah koeficientov polynómu pri kontrole podpisu, zatiaľ čo ω definuje počet nenulových koeficientov v polynóme h. Posledná hodnota označená ako n predstavuje predpokladaný počet opakovaní cyklu, kým bude vygenerovaný platný podpis [82].

Veľkosti kľúčov a podpisov sa nachádzajú v tabuľke 4.5. Napriek tomu, že ML-DSA používa oproti klasickým algoritmom výrazne väčšie kľúče a generovaný podpis je tiež oveľa väčší, tak jeho autori prispôsobili vstupné parametre tak, aby bolo možné algoritmus čo najviac optimalizovať z pohľadu využitia pamäte a rýchlosti výpočtov. Napríklad v prípade verejného kľúča nemusia zariadenia s obmedzeným množstvom pamäte ukladať celú matica $\mathbf{A}$, ale môžu ju pseudo-

Tabuľka 4.5: Porovnanie veľkostí kľúčov a podpisov pre jednotlivé verzie algoritmu CRYSTALS-Dilithium, resp. ML-DSA [82]

Algoritmus	**Súkromný kľúč [B]**	**Verejný kľúč [B]**	**Veľkosť podpisu [B]**
ML-DSA-44	2528	1312	2420
ML-DSA-65	4000	1952	3293
ML-DSA-87	4864	2592	4595

náhodne odvodiť od verejnej počiatočnej hodnoty ρ [82]. Veľkosť podpisu bola výrazne znížená podpisovaním hašu správy μ, nie samotnej správy M. Rovnako tak podpis obsahuje haš $\tilde{c}$ a nie celú hodnotu $\mathbf{w_1}$ [82].

Z pohľadu šetrenia pamäte a rýchlosti výpočtov považujeme za dôležité spomenúť, že podobne ako pri algoritme ML-KEM, tak aj ML-DSA využíva NTT transformáciu pri násobení polynómov. Na úkor vyššej náročnosti na pamäť si môžeme uložiť maticu $\mathbf{A}$ v NTT doméne, čím zrýchlime proces podpisovania a overovania podpisu. To už ale závisí od použitého zariadenia a implementácie algoritmu [82].

4.2.3 Bezpečnosť algoritmu

Bezpečnosť ML-DSA je založená MLWE probléme. Konkrétne ide o spojenie LWE problému a SelfTargerMSIS problému [114], čo je neštandardná forma SIS problému s využitím symetrických hašovacích funkcií. MLWE problém dokážeme úpravami zredukovať na SVP problém, podobne ako pri algoritme CRYSTALS-Kyber. Algoritmus dosahuje SUF-CMA úroveň bezpečnosti, čo znamená, že podpisová schéma dokáže odhaliť neautorizovanú modifikáciu dát, umožňuje overiť identitu podpisovateľa ako vlastníka privátneho kľúča a vygenerovaný podpis slúži zároveň ako dôkaz o tom, že bol vygenerovaný konkrétnou entitou [61]. Inak povedané, útočník nemôže vytvoriť ďalšie platné podpisy na základe podpisu ľubovoľnej správy. Autori na základe matematických výpočtov a analýz určili, že bezpečnosť algoritmu je možné nalomiť útokmi na LWE a SIS, resp. SVP problémy [61]. Najznámejší algoritmom na riešenie SVP problému v Euklidovských mriežka je BKZ algoritmus, ktorý však dosahuje exponenciálnu časovú zložitosť na klasických aj kvantových počítačoch [61]. Na LWE problém sa vzťahujú takzvaný primal útok a duálny útok, u ktorých sa však nepredpokladá vplyv na algoritmus z dôvodu definovaných parametrov, ktoré sa pri výpočtoch využívajú [61]. Spojením LWE a SIS vzniká nová štruktúra, ktorá už bola podro-

bená kryptoanalýze, avšak vyriešené boli problémy na konkrétnych mriežkach, ktoré sú z pohľadu použitých parametrov vzdialené od štruktúry, ktorá sa používa v ML-DSA [61]. Nalomenie algoritmu na základe riešenia SelfTargerMSIS problému by bolo možné iba prelomením hašovacej funkcie alebo využitím BKZ algoritmu na SIS problém [61].

Samozrejme, okrem hašovacích funkcií predpokladajú autori, rovnako ako odborníci z NIST-u, použitie schváleného generátora náhodných čísel v implementácii algoritmu. Z pohľadu implementácie sa tiež očakáva, že všetky medzivýpočty, ktoré by viedli aj k čiastočnému prezradeniu tajných dát budú zmazané hneď ako nebudú potrebné pre výpočet [61]. Za spomenutie tiež stojí, že kontrola dĺžky verejného kľúča a podpisu slúži ako prvotná kontrola platnosti a je súčasťou SUF-CMA bezpečnosti [61].

Keďže je ML-DSA pravdepodobnostný algoritmus, tak všetky výpočty nie je možné realizovať v konštantnom čase. Okrem toho bol algoritmus navrhnutý tak, aby bol odolný voči niektorým útokom cez postranné kanály, pretože nevyužíva Gaussovskú distribučnú funkciu [61]. Samozrejme, niektoré útoky sú stále realizovateľné pretože súvisia s detailami konkrétnej implementácie a od bezpečnostných prvkov zariadenia. Pri výpočtoch modulárnej aritmetiky sa odporúča používať Montgomeryho a Barrettovu redukciu, implementácie hašovacích funkcií by zase mali realizovať výpočty v konštantnom čase [61]. Zariadenia by mali používať rôzne protiopatrenia, napríklad maskovacie techniky proti DPA alebo DEMA [61]. V predošlej podkapitole sme tiež priblížili deterministickú verziu procesu generovania podpisu, ktorá nepoužíva náhodné generátory čísel. Časový útok predstavuje bezpečnostné riziko pre túto verziu algoritmu, pričom môže dôjsť k úniku tajných informácií. Z tohto dôvodu autori odporúčajú používať „slabý[19]“ generátor náhodných čísel než deterministický algoritmus [61]. Analýza ďalších útokov s využitím postranných kanálov, vrátane návrhov možných protiopatrení s minimálnym vplyvom na celkový výkon, je predmetom ďalších štúdií [115],[116],[117],[118],[119],[120].

To, že je bezpečnosť algoritmu veľmi dobre škálovateľná, predstavili autori vo forme špecifikácie parametrov na jej zníženie, resp. na jej zvýšenie. Predstavili tak bezpečnostnú úroveň L1-, ktorá dosahuje približne 60-bitovú bezpečnosť úroveň a L1-, ktorá má 90-bitovú bezpečnosť. Znížením bezpečnostných úrovní chcú autori čiastočne uľahčiť kryptoanalýzu algoritmu, pričom očakávajú, že čiastočné alebo celkové prelomenia týchto úrovní bude znamenať veľký pokrok v oblasti kryptoanalýzy a bude signálom pre používanie algoritmov s minimálnou úrov-

[19] z pohľadu bezpečnosti

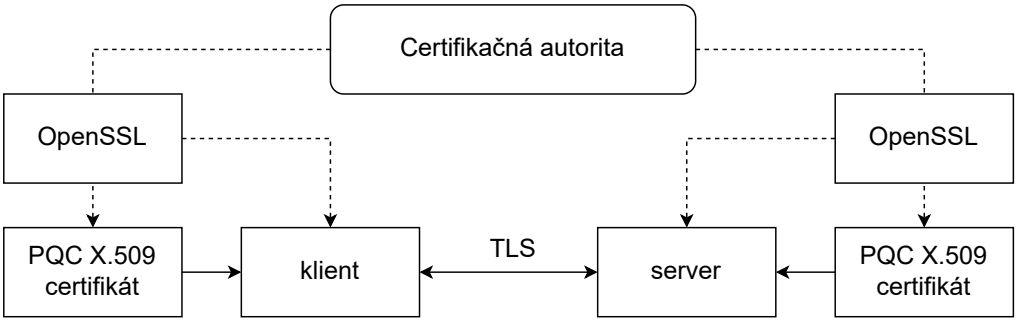
ňou L3 [61]. Naopak zvýšením parametrov, ktoré označili úrovňami L5+ a L5++, chceli predstaviť možnosť zvýšenia bezpečnosti na základe pôvodných výpočtových problémom bez zmien používaných symetrických algoritmov. Ekvivalentná bezpečnosť úrovne L5+ je približne 300 bitov, zatiaľ čo L5++ dosahuje približne 340-bitovú bezpečnosť. Autori odporúčajú L5+ a L5++ verzie algoritmu v prípade ohrozenia úrovne L3 [61]. Parametre pre jednotlivé úrovne L1-, L1-, L5+ a L5++ sú zobrazené tabuľke 4.6.

Tabuľka 4.6: Špecifikácia parametrov pre doplnkové úrovne bezpečnosti algoritmu CRYSTALS-Dilitihum [61]

Bezpečnostná úroveň / **Parameter**	**L1-**	**L1-**	**L5+**	**L5++**
q	8380417	8380417	8380417	8380417
d	10	13	13	13
τ	24	30	60	60
γ_1	2^{17}	2^{19}	2^{19}	2^{19}
γ_2	$(q\text{-}1)/128$	$(q\text{-}1)/128$	$(q\text{-}1)/32$	$(q\text{-}1)/32$
$k; l$	2; 2	3; 3	9; 8	10; 9
η	6	3	2	2
β	144	90	120	120
ω	10	80	85	90
n	5,2	4,87	4,59	5,48

5 Experimentálna časť

Cieľom našej diplomovej práce bolo navrhnúť demonštračnú aplikáciu založenú na architektúre klient-server s využitím TLS protokolu a s podporou PQC algoritmov, pričom pri implementácii sme sa snažili minimalizovať celkový počet použitých knižníc a softvérových nástrojov. Pri analýze zadania sme si určili predpokladaný výsledok, zobrazený na obrázku 5.1, od ktorého sa ďalej vyvíjalo naše smerovanie pri riešení danej problematiky. Našim zámerom bolo tiež vytvoriť



Obr. 5.1: Bloková schéma predpokladaného riešenia našej diplomovej práce

aplikáciu, ktorá bude nezávislá od použitého operačného systému (OS – Operating System). Kvôli lepšej softvérovej podpore sme však každú použitú knižnicu alebo aplikáciu najprv kompilovali a testovali v linuxovom prostredí a až potom na Windows platforme.

Na začiatku štúdia PQC algoritmov v TLS sme pracovali s knižnicou OQS-OpenSSL 1.1.1f. Táto knižnica je už stará a neaktuálna, ale v práci ju spomíname, pretože šlo o našu prvú skúsenosť s používaním PQC algoritmov pri komunikácii klient-server. Rovnako tak sme sa pri inštalácii tejto knižnice stretli s nástrojmi, ktoré sme vôbec v tej dobe nepoznali alebo predtým nikdy nepoužívali. Vďaka tejto knižnici sme získali skúsenosti a prehľad o použitých nástrojoch, ktoré sme následne mohli používať, meniť alebo úplne odstrániť pri inštalácii ďalších knižníc s podporou PQC algoritmov. Keďže sa oblasť kryptografie dynamicky mení a špecificky oblasť PQC sa kvôli prebiehajúcej štandardizácii menila pravidelne, museli sme preto reagovať na nové verzie knižníc, aktualizovať ich, kontrolovať

kompatibilitu, študovať a testovať vykonané zmeny. Okrem samotnej aplikácie sme tiež museli riešiť aj postup generovania a podpisovania post-kvantových certifikátov, s ktorým sme dovtedy tiež nemali veľa skúseností.

V nasledujúcich podkapitolách sa venujeme koncepčnému opisu nami zvoleného postupu, približujeme využívané softvérové nástroje, dôvody na použitie vybraných nástrojov a knižníc. Opisujeme tiež zmysel a spôsob vykonaných úprav v zdrojových kódoch a proces testovania všetkých aplikácií. Detailné postupy inštalácie a používanie najnovších verzií jednotlivých knižníc sú súčasťou príloh.

5.1 Softvérová špecifikácia

V priebehu štúdia a experimentov s PQC sme využívali viac počítačov, ktoré obsahovali rôzne verzie operačných systémov. Väčšie množstvo použitých operačných systémov súvisí predovšetkým s našimi experimentami pri inštalácii knižníc alebo konkrétnymi požiadavkami na použitý operačný systém našim školiteľom, ale tiež s modernizáciou hardvéru staršieho zariadenia, ktorú sme zároveň využili na overenie kompatibility našich experimentov na najnovšej verzii OS. S operačným systémom úzko súvisia aj verzie vývojárskych nástrojov a knižníc, ktoré sme pri práci využívali. Prehľad všetkých operačných systémov a nástrojov zobrazuje tabuľka 5.1.

Tabuľka 5.1: Softvérové parametre operačných systémov a nástrojov použitých pri vývoji

Operačný systém	GCC	cmake	make	ninja	Python	Perl	VS Build Tools 2022	cargo
Linux Ubuntu 20.04.6 LTS WLS2	9.4.0 Ubuntu 9.4.0	3.16.3	GNU Make 4.2.1 x86_64-pc-linux-gnu	1.10.0	3.8.10	–	–	1.73.0
Linux Ubuntu 22.04.3 LTS WSL2	11.4.0 Ubuntu 11.4.0	3.22.1	GNU Make 4.3 x86_64-pc-linux-gnu	1.10.1	3.10.12	–	–	–
Windows 10 10.0.19045	13.2.0 MinGW-W64 release 3 x86_64-ucrt-posix-seh	3.27.1	GNU Make 4.4.1 x86_64-w64-mingw32	1.11.1	3.12.0	5.32.1	17.5.5	1.73.0
Windows 11 10.0.22631	14.0.0[1] release 1 MinGW-W64 x86_64-ucrt-posix-seh	3.28.2	GNU Make 4.4.1 x86_64-w64-mingw32	1.11.1	3.12.3	–	–	–
Windows 7 Super Lite 6.1.7601	13.2.0 MinGW-W64 release 3 i686-ucrt-posix-dwarf	3.29.1	GNU Make 4.4.1 i686-w64-mingw32	1.11.1	3.7.1	–	–	–

Treba však zdôrazniť, že jedným z našich cieľov pri implementácii softvérového riešenia nášho zadania bolo minimalizovať celkový počet použitých nástrojov. Z tohto dôvodu sa pri niektorých nástrojoch objavuje znak „–“, čo znamená,

[1] experimentálna verzia

že sme tieto nástroje v určitom momente používali, ale s postupom práce sme ich prestali používať – buď sme za nich našli vhodnú náhradu alebo už neboli vôbec potrebné.

V priebehu riešenia danej problematiky sme pracovali s rôznymi knižnicami PQC algoritmov a TLS protokolu, pričom nami používané implementácie boli, podľa zadania, z väčšej časti napísané v jazyku C, resp. C++. Mnohé z týchto knižníc používali na automatizované generovanie kompilačných súborov CMake súbory. Úpravou príkazov v základnom CMakeList súbore sme mohli docieliť zmeny pri kompilácii celého projektu. CMake[2] podporuje veľké množstvo generátorov pre konfiguračné súbory, ktoré nám umožňujú definovať nástroj na konfiguráciu a inštaláciu projektu bez nutnosti poznať všetky špecifikácie a závislosti. Pri úprave jedného súboru projektu nám tieto nástroje tiež pomáhajú identifikovať a preinštalovať len súčasti aplikácie, ktoré sú priamo závislé od zmeneného súboru. Niekedy však nemôžeme rozhodnúť o tom, ktorý konfiguračný nástroj použijeme, pretože niektoré z týchto nástrojov sú prispôsobené pre konkrétny operačný systém, niektoré neobsahujú nami potrebné funkcie a niektoré zase môžu byť vyžadované zo strany konkrétnej aplikácie alebo knižnice.

Pri našej práci sme používali NMake Makefile, Unix Makefile, MinGW Makefile generátory, ktoré nám vytvorili konfiguračné Makefile súbory. Generované súbory sa môžu líšiť v štruktúre ale aj v obsahu, kvôli čomu nie sú niekedy vygenerované Makefile súbory kompatibilné. Konfiguračné súbory je potom možné spustiť prostredníctvom príkazov nmake, make, resp. mingw-make. Okrem toho sme pri mnohých projektoch používali aj Ninja generátor, ktorého výsledkom boli *.ninja* súbory.

NMake[3] bol inšpirovaný GNU Make-om, pričom ide o nástroj špecificky prispôsobený pre Windows. Je používaný editorom kódov Visual Studio od spoločnosti Microsoft cez príkazový riadok pre developerov[4] a najčastejšie sa používa v spojení s MSVC (Microsoft Visual C++) kompilátorom pre projekty v jazyku C alebo C++.

GNU Make[5], tiež označovaný ako gmake, je štandardný konfiguračný nástroj pre Linuxové a Unixové systémy, ktorý je dostupný aj pre OS Windows pod názvom MinGW Make. Ide o najznámejšiu a najrozšírenejšiu verziu make nástroja, pričom sa často používa v spojení s GCC (GNU Compiler Collection) kompiláto-

[2] `https://cmake.org/`

[3] `https://learn.microsoft.com/en-us/cpp/build/reference/c-cpp-building-reference?view=msvc-170`

[4] z ang. Developer Command Prompt

[5] `https://www.gnu.org/software/make/`

rom. Linuxová a Windows verzia sa vo funkciách nástroja nelíšia, odlišné názvy vznikli na odlíšenie 32 bitovej verzie a tiež od MSYS (Minimal System) Make.

Ninja[6] je open-source nástroj na plne automatizovanú konfiguráciu a inštaláciu podobne ako GNU Make. Ninja je označovaný ako rýchly a nenáročný, pričom využíva paralelizmus a dokáže detegovať nezmenené súbory, ktoré pri inštalácii ignoruje. Pre zachovanie jeho rýchlosti a efektívnosti sa neodporúča vytvárať build.ninja súbory iným spôsobom ako cez automatizovaný generátor. Vďaka týmto vlastnostiam sa však Ninja stáva stále populárnejší, čo môžeme vidieť aj v knižniciach, ktoré sme pri práci používali.

Na kompiláciu zdrojových kódov v jazyku C, resp. C++, sme používali GCC kompilátor. Perl[7] sme spolu s NMake-om krátkodobo používali na konfiguráciu knižnice OpenSSL 1.1.1 na Windows platforme priamo zo zdrojových kódov. Tento proces bol pomerne náročný a ťažkopádny, no mohli sme sa mu úplne vyhnúť a znížiť tak počet používaných nástrojov použitím predkompilovanej verzie OpenSSL[8]. Python[9] sme používali pri automatizovanej konfigurácii knižnice MIRACL core[10], ktorá bola súčasťou knižnice TiigerTLS[11]. Časť tejto knižnice je napísaná v jazyku Rust, a preto sme jednorázovo použili na kompiláciu aj nástroj rust cargo[12].

5.2 Dostupné kryptografické knižnice s podporou PQC algoritmov

Na začiatku riešenia problematiky našej diplomovej práce sme hľadali knižnicu v jazyku C, resp. C++, ktorá by obsahovala všetky PQC algoritmy a ideálne umožnila ich použitie cez API (Application Programming Interface). Narazili sme na mnohé knižnice a projekty, napísané v rôznych programovacích jazykoch a s rôznymi verziami implementácií PQC algoritmov. Z knižníc, ktoré obsahujú väčšinu PQC algoritmov môžeme spomenúť napríklad:

- QuantCrypt[13], PQcrypt[14] v Pythone,

[6] https://ninja-build.org/
[7] https://strawberryperl.com/
[8] https://www.firedaemon.com/download-firedaemon-openssl
[9] https://www.python.org/
[10] https://github.com/miracl/core
[11] https://github.com/Crypto-TII/TLS1.3
[12] https://doc.rust-lang.org/cargo/
[13] https://github.com/aabmets/quantcrypt
[14] https://github.com/kpdemetriou/pqcrypto

- CIRCL[15] v Go,
- pqc.js[16], noble-post-quantum[17] v TypeScripte, resp. JavaScripte,
- pqcrypto[18] v Ruste.

Avšak keďže sme vedeli, že chceme PQC algoritmy používať priamo v TLS, tak sme hľadali aj knižnice, ktoré by toto umožňovali. V podstate okamžite sme narazili na projekt Open Quantum Safe[19] (OQS), ktorého cieľom je podpora vývoja a hodnotenie PQC algoritmov a ich následná integrácia do protokolov a aplikácií. Celý projekt sa odkazuje na NIST a pracuje výlučne s algoritmami, ktoré sa do procesu štandardizácie zapojili, pričom po každom kole dochádzalo k aktualizáciám jednotlivých knižníc projektu. Za OQS stoja odborníci zo spoločností Cisco, IBM, AWS, Microsoftu a tiež výskumníci z Univerzity Waterloo. Tento projekt nás zaujal predovšetkým svojou knižnicou liboqs[20], ktorú používa v ďalších knižniciach a aplikáciach, ktorým sa venujeme v nasledujúcich podkapitolách.

Knižnica liboqs

Ide o open source knižnicu v jazyku C vytvorenú v rámci projektu Open Quantum Safe, ktorá obsahuje implementácie kvantovo odolných algoritmov pre výmenu kľúčov a algoritmov na digitálny podpis [96],[95]. Implementácie sú odvodené z referenčných a optimalizovaných kódov priamo zo súťaže NIST-u. Okrem finálnych algoritmov knižnica obsahuje niekoľko experimentálnych algoritmov, ktoré boli NIST-om v treťom kole súťaže označené ako alternatívne, napríklad NTRU Prime [121] alebo FrodoKEM [122].

Knižnica je multiplatformová, podporuje rôzne druhy kompilátorov a obsahuje tiež nástroje na medzi-platformovú kompiláciu. Súčasťou knižnice sú aj testy na porovnanie výkonu jednotlivých PQC algoritmov [95]. Liboqs tvorí základ pre rôzne ďalšie knižnice alebo priamo aplikácie a to bez závislosti na programovacom jazyku. Zároveň je liboqs použitá na integráciu PQC algoritmov do protokolov ako TLS, SSH a X.509 [95], ktoré sú ďalej využívané v aplikáciách a knižniciach ako OpenSSL, nginx, curl, Chromium alebo Wireshark [95]. Prostredníctvom wrapperov je možné knižnicu využívať v prostrediach programovacích ja-

[15] `https://github.com/cloudflare/circl`
[16] `https://github.com/Dashlane/pqc.js/`
[17] `https://github.com/paulmillr/noble-post-quantum`
[18] `https://github.com/rustpq/pqcrypto`
[19] `https://openquantumsafe.org/`
[20] `https://github.com/open-quantum-safe/liboqs`

zykov C++[21], Go[22], Java[23], .NET[24], Python[25], Rust[26] a PHP[27].

Iné PQC knižnice v jazyku C

Okrem liboqs sú dostupné aj iné knižnice v jazyku C, resp C++, ktoré buď obsahujú zdrojové kódy PQC algoritmov alebo ich priamo integrujú do TLS. Za spomenutie určite stoja:

- PQClean[28] – obsahuje minimalistické a testované verzie PQ algoritmov, ktoré sa využívajú v iných knižniciach ako liboqs, QuantCrypt alebo pqcrypto.
- Botan[29] – kryptografická knižnica v C++, na ktorú sme narazili až pri písaní našej práce, avšak môže byť súčasťou ďalšieho štúdia.
- AWS libcrypto (AWS-LC)[30] – kryptografická knižnica spravovaná spoločnosťou AWS, vytvorená na základoch BoringSSL a OpenSSL.
- pqm4[31] – kryptografická knižnica, ktorá obsahuje optimalizované implementácie PQ algoritmov pre ARM Cortex-M4 mikrokontroléry.
- pq-wolfSSL[32] – derivát knižnice wolfSSL s integrovanými PQ algoritmami do TLS protokolu pre vstavané[33] systémy.

5.3 Knižnica OQS-OpenSSL 1.1.1

Súčasťou projektu Open Quantum Safe bol aj derivát knižnice OpenSSL 1.1.1[34], ktorý prostredníctvom open-source knižnice liboqs integroval aktuálne verzie, v tej dobe ešte kandidátov tretieho kola, vybraných PQC algoritmov do TLS protokolu. Celá knižnica bola z väčšej časti tvorená kódmi v jazyku C a bola prispôsobená tak, aby ju bolo možné použiť na rôznych operačných systémoch. Po krát-

[21] https://github.com/open-quantum-safe/liboqs-cpp
[22] https://github.com/open-quantum-safe/liboqs-go
[23] https://github.com/open-quantum-safe/liboqs-java
[24] https://github.com/open-quantum-safe/liboqs-dotnet
[25] https://github.com/open-quantum-safe/liboqs-python
[26] https://github.com/open-quantum-safe/liboqs-rust
[27] https://github.com/Muzosh/liboqs-php
[28] https://github.com/PQClean/PQClean
[29] https://github.com/randombit/botan
[30] https://github.com/aws/aws-lc
[31] https://github.com/mupq/pqm4
[32] https://github.com/boschresearch/pq-wolfSSL
[33] z ang. embedded
[34] https://github.com/open-quantum-safe/openssl

kom naštudovaní priloženej dokumentácie a ďalších návodov dostupných na internete sa nám podarilo celú knižnicu nainštalovať a súčasne otestovať generovanie certifikátov rôznych PQ algoritmov a ich využitie v základných aplikáciach TLS klienta a serveru v OpenSSL.

Knižnicu sme testovali na OS Linux 20.04 a Windows 10. Postup a použité nástroje sa na oboch operačných systémoch mierne líšili, avšak výsledok bol rovnaký. V prípade Linuxu bola inštalácia pomerne bezproblémová a to aj napriek tomu, že návod inštalácie dostupný v knižnici nie je úplne detailný ani intuitívny. Našli sme však iný návod[35], podľa ktorého sme postupovali.

Všetky potrebné nástroje ako CMake alebo ninja sme si v linuxovom prostredí nainštalovali jedným príkazom

```
sudo apt install cmake gcc libtool libssl-dev make ninja-build git.
```

Následne sme si prostredníctvom gitu naklonovali repozitár knižnice liboqs a podľa dokumentácie sme ju nainštalovali. Potom sme stiahli a nakonfigurovali knižnicu OQS-OpenSSL, ktorá sa automaticky prepojila s knižnicou liboqs a sprístupnila PQC algoritmy pre použitie v TLS. Po vygenerovaní potrebných kľúčov a certifikátov sme mohli overiť funkčnosť celého protokolu prostredníctvom aplikácii klienta a serveru.

Kompilácia knižnice v prostredí Windows 10 bola o niečo náročnejšia, z dôvodu inštalácie nástrojov mimo príkazového riadku a tiež kvôli nejednoznačnému návodu, ktorý knižnica obsahuje. Najprv sme museli stiahnuť a nainštalovať nástroje Perl Strawberry, CMake a VS Build Tools 2022 ktorých súčasťou je NMake, pričom pre správnu funkcionalitu NMake-u v príkazovom riadku sme museli dodatočne spustiť súbor vcvarsall.bat, ktorý je tiež súčasťou MS Visual Studio. Následne sme stiahli repozitár OQS OpenSSL knižnice a liboqs, ktoré sme postupne nakonfigurovali a skompilovali, avšak tentokrát s využitím nástrojov ninja, perl a nmake. Aplikácie klienta a server fungovali rovnako ako v linuxovom prostredí.

Pri inštalácii tejto knižnice sme použili viacero nástrojov, a preto sme sa rozhodli nájsť spôsob ako ich celkové množstvo zredukovať. Rozhodli sme sa preto hľadať úplne inú knižnicu, resp. aplikáciu, ktorá by nám umožňovala používať PQC, pričom našim cieľom bolo nájsť čo najviac minimalizovanú aplikáciu, keďže OQS-OpenSSL nám prišla komplikovaná, neprehľadná a obsahovala, aspoň pre naše potreby, nepodstatné funkcie. Ďalším dôvodom bol aj fakt, že 23.9.2023 bolo

[35] `https://www.linkedin.com/pulse/practical-quantum-safe-tls-13-apache-web-server-setup-igor-barshteyn`

naplánované ukončenie podpory pre OpenSSL 1.1.1f[36] kvôli čomu mala skončiť aj podpora pre OQS-OpenSSL.

Celý návod na kompiláciu knižnice pre Linux a Windows sa nachádza na našom Gite[37] a je tiež súčasťou prílohy B, avšak keďže je táto knižnica už viac ako pol roka neaktualizovaná, tak neodporúčame jej používanie.

5.4 OpenSSL, liboqs a oqs-provider

Pri hľadaní novej knižnice, ktorá by nám umožňovala používať PQ algoritmy v TLS sme sa zamerali na jej kompatibilitu s vtedy aktuálnou verziou OpenSSL 3.0, resp. 3.1. Nemuseli sme hľadať dlho, pretože projekt Open Quantum Safe obsahoval aj knižnicu oqs-provider[38].

OpenSSL momentálne neobsahuje žiadne PQC algoritmy. Napriek tomu sa však tvorcovia OpenSSL, podobne aj ďalšie spoločnosti ako Microsoft [123], Google [124] , AWS [125] alebo IBM [126], postupne pripravujú na implementáciu PQC algoritmov do svojich produktov. Väčšina spoločností však označuje tieto úpravy za experimentálne a čaká na oficiálne výsledky NIST štandardizácie.

OpenSSL s verziou 3.0.1 predstavilo koncept providerov, nový systém pridávania dynamických kryptografických modulov do OpenSSL, ktorý nahradil engine API. Provider je v podstate knižnica, ktorú si môžeme predstaviť ako kontajner, ktorý nám po pripojení k OpenSSL rozširuje funkcie alebo kryptografické primitíva prostredníctvom vysokoúrovňového API, čo znamená, že nové funkcie sa vykonávajú na úrovni providera, nie OpenSSL. Samotné OpenSSL natívne obsahuje niekoľko providerov, ale tiež umožňuje vytvoriť a pripojiť vlastného providera.

Knižnica oqs-provider umožňuje prepojiť knižnicu liboqs s OpenSSL a používať tak PQC algoritmy v TLS komunikácii.

Našou úlohou bolo nainštalovať OpenSSL, liboqs, oqs-provider a spolu ich prepojiť. Najprv sme začali inštaláciou všetkých súčastí v linuxovom prostredí. Aj keď je OpenSSL súčasťou linuxových balíkov, podobne ako všetky ostatné nástroje, ktoré sme pri inštalácii potrebovali, tak v prípade Ubuntu 20.04 šlo v tej dobe o aktuálnu verziu 1.1.1f. Rozhodli sme sa preto nahradiť túto verziu OpenSSL novšou verziou 3.0.7. Vyskúšali sme rôzne návody dostupné na internete a vždy sa javilo, že inštalácia prebehla správne, avšak stále sme narazili na chyby, ktoré

[36] https://www.openssl.org/blog/blog/2023/03/28/1.1.1-EOL/
[37] https://git.kemt.fei.tuke.sk/js331zc/MastersThesis
[38] https://github.com/open-quantum-safe/oqs-provider

boli spôsobené nesprávnym linkovaním súborov. Tieto chyby sme väčšinou odhalili až pri inštalácii knižnice liboqs, ktorá pri inštalácii vypisuje do príkazového riadku informácie o verziách knižníc. Nakoniec sa nám podarilo nájsť postup[39], prostredníctvom ktorého sme si nainštalovali najnovšiu verziu OpenSSL. Po určitej dobe sme zistili že Ubuntu 22.04 natívne obsahuje OpenSSL 3.0.2 a s myšlienkou zníženia nutných inštalácii sme prešli na túto distribúciu Linuxu.

Inštalácia knižníc liboqs a oqs-provider bola tiež pomerne komplikovaná, pretože inštalačné postupy sa nám na prvý pohľad nezdali úplne intuitívne a obsahovali z nášho pohľadu nové nástroje ako napríklad ninja. Opakovanie inštalácií kvôli zlej verzii OpenSSL zapríčinenej nesprávnym linkovaním súborov nám však pomohlo zorientovať sa v jednotlivých krokoch inštalácie aj v použitých nástrojoch. Knižnice sa nám podarilo úspešne nainštalovať, pričom funkčnosť liboqs sme mohli experimentálne overiť spustením vzorových zdrojových kódov[40], ktoré sú dostupné v dokumentácii knižnice.

Nakoniec sme museli aktivovať oqs-providera v OpenSSL. To je možné urobiť buď priamo cez API funkciu v zdrojovom kóde aplikácie alebo cez konfiguračný súbor OpenSSL. My sme zvolili druhý variant aktivácie, pretože sme potrebovali mať providera aktivovaného vždy kvôli generovaniu PQC certifikátov. Do súboru openssl.cnf sme podľa dokumentácie knižnice doplnili príkazy 5.1, ktorými sme aktivovali modul oqs-providera.

Zdrojový kód 5.1: Príkazy doplnené do súboru openssl.cnf na statickú aktiváciu oqs-providera v OpenSSL

```
1 [provider_sect]
2 default = default_sect
3 oqsprovider = oqsprovider_sect
4 [oqsprovider_sect]
5 activate = 1
```

Musíme zdôrazniť, že základný provider, označený ako *default provider*, musí byť tiež aktivovaný. Aktivácia providerov prebehla okamžite po uložení konfiguračného súboru a mohli sme ju overiť príkazom

openssl list -providers,

ktorý nám vrátil výsledok zobrazený na obrázku 5.2.

Pri testovaní funkčnosti všetkých knižníc sa nám podarilo vygenerovať postkvantové certifikáty, ale pri spustení serveru s týmito certifikátmi sme stále dostávali chybu o neznámom certifikáte. To bolo spôsobené tým, že OpenSSL 3.0 a 3.1

[39] https://www.golinuxcloud.com/install-openssl-ubuntu/

[40] https://openquantumsafe.org/liboqs/examples/kem

```
josi@DESKTOP-RQURBU2:~$ openssl list -providers
Providers:
  default
    name: OpenSSL Default Provider
    version: 3.2.0
    status: active
  oqsprovider
    name: OpenSSL OQS Provider
    version: 0.5.4-dev
    status: active
```

Obr. 5.2: Overenie aktivácie knižnice oqs-provider v OpenSSL

nepodporovali PQC algoritmy priamo v TLS komunikácii klient/server. Tento problém sme však veľmi rýchlo vyriešili, keďže na OpenSSL Gite už bola dostupná vývojárska verzia 3.2-dev, ktorú sme podľa rovnakého návodu nainštalovali. Následne sme preinštalovali knižnice liboqs a oqs-provider. Po aktivácii providera sme znovu vygenerovali testovací post-kvantový certifikát a úspešne ho použili pri spustení natívneho serveru, ktorý je súčasťou OpenSSL. K serveru sme sa pripojili cez OpenSSL klienta.

Inštaláciu na zariadení s Windowsom sme hneď nerealizovali, keďže ako sme mohli zistiť pri OQS-OpenSSL, tak inštalácia a konfigurácia OpenSSL pre Windows priamo zo zdrojových kódov nie je triviálna. Pár týždňov po našom experimente bola OpenSSL 3.2 oficiálne zverejnená a my sme mohli overiť náš experiment aj na OS Windows. Pri experimente sme použili balík nástrojov WinLibs, ktorý obsahuje prekladač gcc, cmake aj nástroj ninja. Zaujímavosťou je, že na stránke knižnice WinLibs sa nachádza aj experimentálna verzia prekladača gcc 14.0.1, ku ktorej však nie sú pridané niektoré nástroje ako napríklad CMake. Túto verziu prekladača sme úspešne otestovali, ale kvôli nadbytočnej inštalácii CMake-u sme prešli späť na verziu 13.2. OpenSSL sme neinštalovali, ale využili sme predkompilovanú verziu[41] celej knižnice. Celý postup inštalácie knižníc liboqs a oqs-provider bol veľmi podobný ako v linuxovom prostredí, avšak pri generovaní konfiguračných súborov cez CMake sme museli dodatočne definovať cestu k OpenSSL, resp. k liboqs v prípade oqs-providera, čo bolo spôsobené absenciou niektorých systémových premenných. Pri inštalácii oqs-providera sme však narazili na problém spojený s testami, ktoré sú súčasťou knižnice. Test s názvom *oqs_test_tlssig.c* obsahuje časť kódu, ktorá je špecifická pre linuxové prostredie a vo Windows prostredí spôsobí chybu. Konkrétne ide o definíciu funkcie *mkdir*() na vytvorenie nového priečinku, ktorá je v teste použitá. Zatiaľ čo v linu-

[41] `https://www.firedaemon.com/download-firedaemon-openssl`

xovom prostredí umožňuje funkcia definovať okrem názvu, resp. cesty, aj prístupové práva k novému priečinku, tak na Windows platforme funkcia obsahuje iba jeden parameter a to je názov, resp. cesta k priečinku. Tento problém sa dá vyriešiť dvoma spôsobmi:

- upravením testu *oqs_test_tlssig.c*, kde doplníme podmienenú kompiláciu pre Windows s upravenou funkciou ako zobrazuje kód 5.2,
- vylúčením testov z procesu kompilácie odstránením alebo zakomentovaním vybraných riadkov v základom súbore *CMakeLists.txt*, tak ako v kóde 5.3.

Zdrojový kód 5.2: Úprava kódu pomocou podmienky pre-procesora v teste oqs_test_tlssig.c kvôli problémom s jeho kompiláciou na Windows platforme

```
#ifdef __WIN32
    if (mkdir(certsdir)) {
        if (errno != EEXIST) {
            fprintf(stderr, "Couldn't create certsdir %s: Err = %d\
               n", certsdir,
                    errno);
            ret = -1;
            goto err;
        }
    }
#else
        if (mkdir(certsdir, 0700)) {
             if (errno != EEXIST) {
            fprintf(stderr, "Couldn't create certsdir %s: Err = %d\
               n", certsdir,
                    errno);
            ret = -1;
            goto err;
        }
    }
#endif
```

Zdrojový kód 5.3: Pridanie komentárov k testom v *CMakeLists.txt* pred inštaláciou knižnice oqs-provider, aby sme zabránili prerušeniu inštalácie na Windows platforme kvôli chybe v teste oqs_test_tlssig.c

```
# Testing
# enable_testing()
# add_subdirectory(test)
```

Úspešnú inštaláciu sme následne overili aktiváciou providera, vygenerovaním certifikátov a spustením OpenSSL servera.

Detailný postup inštalácie pre Linux aj Windows sú súčasťou prílohy C nachádza sa tiež na našom Gite[42].

Aplikácie klienta a servera s podporou PQC algoritmov pre potreby výučby

Napriek tomu, že sa nám podarilo používať PQ algoritmy v TLS komunikácii, tak v prípade konfigurácie OpenSSL priamo zo zdroja ide o pomerne rozsiahlu knižnicu s množstvom funkcií, podobne ako OQS-OpenSSL. Preto sme sa rozhodli vytvoriť vlastné aplikácie klienta a servera, ktoré budú používať OpenSSL knižnicu.

Ako základ sme použili zdrojové kódy klienta a servera, ktoré sa používajú v rámci výučby predmetu Bezpečnosť počítačových systémov (BPS). Ide o pomerne jednoduché kódy, ktoré obsahujú klasickú soketovú komunikáciu nezávislú od použitého operačného systému, v jednej verzii cez podmienenú kompiláciu, v druhej verzii pomocou BIO štruktúr, pričom v oboch prípadoch prebehne po nadviazaní spojenia medzi klientom a serverom TLS handshake prostredníctvom výmeny kľúčov a overením certifikátov. S týmto balíkom sme sa stretli už skôr počas štúdia a tiež pri testovaní jeho kompatibility s OpenSSL 3.2.

Pri úpravách sme začali s kódom serveru so soketovou komunikáciou realizovanou cez BIO štruktúru[43]. Ako sme už spomenuli, tak oqs-provider funguje ako aplikácia, ktorá nemení, ale rozširuje funkcie OpenSSL, čo znamená, že provider nemal žiadny vplyv na už použité funkcie v zdrojovom kóde a pri úpravách sme tiež používali výlučne OpenSSL funkcie.

Proces výmeny kľúčov je v OpenSSL definovaný štruktúrou predvolených skupín[44], čo je vlastne zoznam, ktorý obsahuje základné algoritmy na výmenu kľúčov pri TLS handshaku. Pri pripojení klienta k serveru, porovnáva server štruktúru predvolených skupín so skupinami, ktoré podporuje klient. Pri nájdení zhodného algoritmu prebehne výmena kľúčov. Ak klient a server nenájdu zhodný algoritmus, tak dôjde k prerušeniu handshaku. Táto štruktúra pôvodne vznikla kvôli rôznym druhom eliptických kriviek pri ECDH algoritme. Jednotlivé algoritmy sú v zozname oddelené dvojbodkou „:“, pričom prvý algoritmus v zozname je základný a primárne preferovaný, v štandardnom TLS 1.3 ide o algoritmus X25519. Knižnica oqs-provider pridala medzi definované skupiny aj PQC algoritmy na výmenu kľúčov. Do zdrojového kódu sme pridali funkciu *SSL_set1_groups_list*(), ktorá umožňuje meniť štruktúru predvolených skupín. Ukážka po-

[42] `https://git.kemt.fei.tuke.sk/js331zc/MastersThesis`

[43] na realizácii soketovej komunikácie nezáleží

[44] z ang. default groups

užitia tejto funkcie je znázornená kódom 5.4. Aby sme upravovanie skupín čiastočne uľahčili, tak sme celý zoznam definovali ako globálnu premennú s názvom DEFAULT_GROUPS, ktorú môže používateľ pomerne jednoducho upravovať. Táto premenná nemôže ostať prázdna, inak server spadne. Naopak v prípade, že zoznam obsahuje neznámy alebo nepodporovaný algoritmus a server na neho narazí pri hľadaní zhodného algoritmu s klientom, tak skončí s oznámením o neznámom algoritme.

Zdrojový kód 5.4: Globálna premenná DEFAULT_GROUPS, ktorá definuje zoznamu predvolených skupín na výmenu kľúčov pri TLS hanshaku spolu s funkciou, ktorá tento zoznam načíta do SSL kontextu

```
1  // zoznam algoritmov na vymenu klucov, ktore bude server podporovat
2  #define DEFAULT_GROUPS "kyber768:frodo976aes:kyber1024"
3
4  // nastavenie predvolenych skupin pre OpenSSL
5  // bez tejto funkcie by server pouzival na vymenu klucov X25519
6  if (SSL_set1_groups_list(ssl, DEFAULT_GROUPS) != 1){
7      // funkcia nesmie obsahovat prazdny zoznam
8      printf("KEX/KEM algorithms undefined - check DEFAULT_GROUPS
           variable\n");
9      ERR_print_errors_fp(stderr);
10 }
```

Čo sa týka PQC algoritmov pre digitálny podpis, ich podpora je priamo definovaná prostredníctvom OpenSSL a API oqs-providera, pretože identifikácia použitého algoritmu prebieha pri načítaní certifikátu a pri jeho overovaní. Pre lepší prehľad použitých algoritmov sme do zdrojových kódov doplnili aj štandardné funkcie na výpis podpisovej schémy certifikátu serveru, funkciu na výpis použitého KEX/KEM algoritmu a výpis podpisovej schémy certifikátu klienta, ktoré opäť kvôli oqs-providerovi dokážu vypísať informácie aj o PQC algoritmoch. Pri výpisoch sme však museli dávať pozor na správnu formu návratovej hodnoty, pretože funkcie väčšinou vracajú číselnú hodnotu algoritmu alebo smerník na štruktúru. Tento problém sme vyriešili volaním ďalších funkcií, ako napríklad *SSL_group_to_name*() alebo *OBJ_nid2sn*(), ktoré dokážu premeniť číselnú hodnotu objektu algoritmu na jeho názov. Okrem toho sme v zdrojovom kóde urobili aj drobné úpravy, ktoré sa týkali lepšej kontroly návratových hodnôt niektorých funkcií a sprehľadnenia výpisov aplikácie.

V zdrojom kóde klienta sme vykonali úplne rovnaké úpravy ako v kóde serveru, čo znamená, že sme definovali zoznam predvolených algoritmov na výmenu kľúčov ako globálnu premennú DEFAULT_GROUPS a potom ju použili vo funkcii *SSL_set1_groups_list*(). Doplnili sme tiež výpisy s informáciami o po-

užitých algoritmoch a čiastočne „upratali“ celý kód.

Okrem toho celý balík pre výučbu obsahuje aj zložku *CERTIFICATES*, ktorá obsahuje skripty na automatické generovanie RSA, resp. ECC certifikátov. S týmito skriptami sme sa bližšie zoznámili pri testovaní s verziou OpenSSL 3.2, kedy sme museli upraviť skript na generovanie RSA certifikátov, kde sme zvýšili veľkosť súkromného kľúča z 1024 bitov na 2048 bitov podľa nového bezpečnostného levelu SSL (Secure Sockets Layer)/TLS[45]. Pre post-kvantové algoritmy sme vytvorili nový priečok s názvom PQ, kde sme pripravili skript na generovanie post-kvantových certifikátov. Konfiguračné súbory sú zhodné s tými, ktoré sa nachádzajú v RSA, resp. ECC zložke. Vytvorený skript obsahuje príkazy v podobnej štruktúre ako v skripte pre RSA, resp. ECC certifikáty a to preto, aby študenti mohli jasne vidieť minimálny rozdiel medzi generovaním týchto certifikátov. V štruktúre príkazov sa skript líši hlavne prepínačom „*-algorithms*“, za ktorým môžeme špecifikovať názov PQ algoritmu pre vygenerovanie súkromného kľúča. Do priečinku sme pridali aj krátky README súbor, ktorý obsahuje zoznam všetkých podporovaných post-kvantových algoritmov a to vo forme presných názvov, ktoré sa pri generovaní dajú použiť.

Celý vývoj a testovanie sme najprv realizovali v linuxovom prostredí, kde sme sa nestretli so žiadnymi vážnejšími problémami. Následne sme prešli na Windows, pričom sme skúšali overiť kompatibilitu medzi zariadeniami Windows 10 a Windows 11. Narazili sme však na problém, kedy naše aplikácie serveru a klienta označovali použité post-kvantové algoritmy v certifikátoch aj pri výmene kľúčov za neznáme, čo malo za následok predčasné ukončenie aplikácií. Táto chyba sa objavovala aj napriek aktivovanému oqs-providerovi, prostredníctvom ktorého sme generovali úplne nové PQ certifikáty. Knižnice liboqs a oqs-provider sme sa pokúšali viackrát preinštalovať s domnením, že sme pri konfigurácii nepoužili správne prepínače alebo sme urobili chyby v ich definícii. Nakoniec sa nám na zariadení s Windows 11 podarilo obe aplikácie bez problémov spustiť s použitím post-kvantových algoritmov a to aj napriek tomu, že sme pri konfigurácii knižníc neurobili žiadnu zmenu. Toto správanie sme označili za „bug“ spojený s OpenSSL a oqs-providerom na Windows platforme. Problém sme vyriešili aktiváciou oqs-providera priamo v zdrojovom kóde našich aplikácií pomocou API funkcie *OSSL_PROVIDER_load*() podľa príkladu[46], ktorý je súčasťou knižnice. Zdrojový kód 5.5 zobrazuje pridané funkcie na aktiváciu oqs-providera,

[45] `https://www.openssl.org/docs/man3.1/man3/SSL_CTX_set_security_level.html`

[46] `https://github.com/open-quantum-safe/oqs-provider/blob/main/examples/static_oqsprovider.c`

spolu s aktiváciou základného providera, ktorý je nutné tiež aktivovať. Zároveň sme doplnili aj funkciu na uvoľnenie pamäte pri skončení programu.

Zdrojový kód 5.5: Aktivácia základného providera a oqs-providera pomocou funkcie v zdrojových kódoch klienta a servera

```
1  OSSL_PROVIDER* provider;
2      provider = OSSL_PROVIDER_load(NULL, "default");
3      if (provider == NULL) {
4          printf("Failed to load Default provider\n");
5              abort();
6      }
7
8      OSSL_PROVIDER* custom_provider = OSSL_PROVIDER_load(NULL, "
           oqsprovider");
9      if (custom_provider == NULL){
10             printf("Failed to load OQS-provider\n");
11         OSSL_PROVIDER_unload(provider);
12             abort();
13     }
```

Keďže sa celý balík v rámci predmetu BPS používa na virtuálnom obraze OS Windows 7, tak sme sa rozhodli overiť jeho funkčnosť aj na tomto zariadení. Po rýchlej príprave virtuálneho stroja sme mohli prejsť na inštaláciu potrebných knižníc. Pôvodne sme mali obavy, pretože sme nevedeli akým spôsobom bude inštalácia prebiehať keďže šlo o 32 bitovú verziu OS. Jediný problém, na ktorý sme narazili, bol nefunkčný nástroj CMake, ktorý bol súčasťou distribúcie WinLibs. Po nahradení CMake-u z oficiálnej stránky[47] sme mohli pokračovať v inštalácii celého projektu úplne rovnakým spôsobom ako pri predošlých zariadeniach. Následne sme mohli potvrdiť funkčnosť celého projektu aj na tomto zariadení. Pre rýchle a jednoduché používanie aplikácie študentmi pri výúčbe, sme k celému balíku pridali už vygenerovaný *.dll* súbor knižnice oqs-provider, kompatibilný s 32-bitovou verziou OpenSSL na virtuálnom stroji s Windows 7.

Celý balík sme dodatočne otestovali tesne pred odovzaním s najnovšími verziami všetkých knižníc, teda OpenSSL 3.3.0, liboqs 0.10.0 a oqs-provider 0.6.0. Aktualizovali sme tiež priložený súbor *oqs-provider.dll*.

Balík, označený ako PQ_PROJECT_SSL_TLS, je súčasťou prílohy A. Postup na kompiláciu a použitie jednotlivých komponentov balíka je súčasťou prílohy D. Celý projekt je tiež dostupný online na našom Gite[48]

[47] `https://cmake.org/download/`

[48] `https://git.kemt.fei.tuke.sk/js331zc/MastersThesis`

5.5 TiigerTLS

Pri hľadaní knižníc s post-kvantovou kryptografiou sme boli našim vedúcim práce upozornení na knižnicu TiigerTLS[49] od Michaela Scotta. Ide o open-source implementáciu TLS protokolu v jazyku C++ a Rust, pričom autor sa zameral na vytvorenie čo najjednoduchších a prehľadných zdrojových kódov, vhodných aj pre vstavané systémy [89]. Napriek tomu, že veľká časť knižnice je napísaná v Ruste kvôli jeho rozširovaniu v oblasti kryptografie a vstavaných systémov, tak Scott pripravil aj implementáciu klienta v jazyku C++. Knižnica obsahuje implementáciu TLS pričom používa kryptografické knižnice MIRACL core[50] a tiež libsodium[51]. Z MIRACL core používa všetky potrebné kryptografické funkcie, vrátane post-kvantových algoritmov CRYSTALS-Kyber a CRYSTALS-Dilithium. Libsodium slúži ako voliteľné rozšírenie aplikácie kvôli zaujímavej implementácii generátoru náhodných čísel a rýchlej implementácii algoritmov ChaCha20 a X25519 [89]. Autor zároveň napísal kód tak, aby dôsledne oddelil komunikačnú vrstvu od bezpečnostnej vrstvy (SAL – Security Abstraction Layer), čo znamená, že používateľ nemusí nutne používať knižnicu MIRACL core, ale môže ju nahradiť inou knižnicou, bez veľkého zásahu do zdrojových kódov celej aplikácie, ak zachová štruktúru funkcií bezpečnostnej vrstvy [89]. Celá SAL vrstva je zároveň veľmi jednoduchá a zložená z pomerne malého množstva funkcií, pričom funkcie sú od seba nezávislé, čo znamená, že pri potrebe upravovať SAL vrstvu nie je potrebné robiť úpravy v celej vrstve. Funkcie SAL vrstvy sú zobrazené v zdrojovom kóde 5.6.

Zdrojový kód 5.6: Zoznam funkcií SAL vrstvy knižnice TiigerTLS

```
1  char *SAL_name()
2  bool SAL_initLib()
3  void SAL_endLib()
4  int SAL_ciphers(int *ciphers_suites)
5  int SAL_groups(int *groups)
6  int SAL_sigs(int *sigAlgs)
7  int SAL_sigCerts(int *sigAlgs)
8  int SAL_hashType(int cipher_suite)
9  int SAL_hashLen(int hash_type)
10 int SAL_aeadKeylen(int cipher_suite)
11 int SAL_aeadTaglen(int cipher_suite)
12 int SAL_randomByte()
```

[49] https://github.com/Crypto-TII/TLS1.3

[50] https://github.com/miracl/core

[51] https://doc.libsodium.org/

```
13 void SAL_randomOctad(int len,octad *R)
14 void SAL_hkdfExtract(int hash_type,octad *PRK,octad *SALT,octad *
     IKM)
15 void SAL_hkdfExpand(int hash_type, int olen, octad *OKM, octad *PRK
     , octad *INFO)
16 void SAL_hmac(int hash_type,octad *T,octad *K,octad *M)
17 void SAL_hashNull(int hash_type,octad *H)
18 void SAL_hashInit(int hash_type,unihash *h)
19 void SAL_hashProcessArray(unihash *h,char *b,int len)
20 int SAL_hashOutput(unihash *h,char *d)
21 void SAL_aeadEncrypt(crypto *send,int hdrlen,char *hdr,int ptlen,
     char *pt,octad *TAG)
22 bool SAL_aeadDecrypt(crypto *recv,int hdrlen,char *hdr,int ctlen,
     char *ct,octad *TAG)
23 void SAL_generateKeyPair(int group,octad *SK,octad *PK)
24 void SAL_generateSharedSecret(int group,octad *SK,octad *PK,octad *
     SS)
25 bool SAL_tlsSignatureVerify(int sigAlg,octad *BUFF,octad *SIG,octad
      *PUBKEY)
26 void SAL_tlsSignature(int sigAlg,octad *KEY,octad *BUFF,octad *SIG)
```

Na začiatku sme sa rozhodli s knižnicou viac zoznámiť preložením a zlinkovaním jej súčasti. Začali sme v linuxovom prostredí s kompiláciou kódov v jazyku C++, ale prostredníctvom prekladača gcc, pričom sme postupovali podľa skriptu, ktorý je súčasťou knižnice. Najprv sme si stiahli zdrojové kódy knižnice MIRACL core a nainštalovali Python, ktorý slúži na automatickú kompiláciu konkrétnej časti MIRACL core knižnice[52]. Prešli sme do zložky pre C++ a skompilovali sme zdrojové kódy príkazom

```
python3 config64.py test
```

Tento príkaz nám skompiloval celú knižnicu MIRACL core, ktorá je pomerne rozsiahla, no knižnica TiigerTLS nevyžaduje použitie všetkých jej súčastí. Rozsah knižnice MIRACL core môžeme zmenšiť definovaním parametrov pri Python kompilácii priamo príkazom

```
python3 ./config64.py --options=1 --options=2 --options=3 \\
--option=7 --options=8 --options=31 --options=42 --options=44
```

Vygenerované súbory sme následne skopírovali do priečinku */sal/* v knižnici TiigerTLS. Potom sme pomocou CMake-u vygenerovali inštalačné súbory. Keďže celá knižnica je v jazyku C++ a my sme použili prekladač gcc, tak pri zavolaní

[52] závisí od požadovaného programovacieho jazyku

príkazu *make* sa objavila chyba, ktorú sme museli ošetriť pridaním prepínača „*-lstdc++*“ na koniec súboru *CMakeFiles/client.dir/link.txt*.

Aby sme sa dozvedeli viac o funkciách celej knižnice, rozhodli sme sa tiež nainštalovať aplikácie klienta a servera napísané v Ruste. Zo stránky[53] sme si stiahli a nainštalovali kompilačný nástroj pre Rust, označený ako Cargo[54]. Pomocou predošlého Python príkazu sme vygenerovali súbory knižnice MIRACL core, tentokrát pre Rust. V súbore *server/Cargo.toml*, resp. *client/Cargo.toml* sme potom aktualizovali cestu k vygenerovaným súborom. Obe aplikácie sme následne v ich zložkách nainštalovali a spustili príkazmi *cargo build* a *cargo run*, pričom okrem kontroly spojenia medzi klientom a serverom v Ruste sme overili aj komunikáciu medzi serverom v Ruste a klientom v C++.

Zaujímalo nás do akej miery je knižnica kompatibilná s Windows platformou, pretože autor sa o tom nikde nezmienil. Rozhodli sme sa teda celú knižnicu nainštalovať aj na počítači s OS Windows. V prípade Rust kódov bol postup kompilácie veľmi podobný s pár medzikrokmi, ktoré boli spôsobené tým, že sme podporu pre Rust a Python inštalovali mimo príkazového riadku. Rovnakým príkazom sme si vygenerovali Rust súbory knižnice MIRACL core a aktualizovali cestu k nim v súboroch *server/Cargo.toml*, resp. *client/Cargo.toml*. Následne sme obe aplikácie spustili v rôznych oknách PowerShellu príkazmi *cargo build* a *cargo run*.

Pri C++ klientovi sme pôvodne postupovali rovnakým spôsobom. Vygenerované C++ súbory MIRACL core sme si skopírovali do zložky */sal/* a použili sme nástroj CMake na vytvorenie inštalačných súborov. Potom sme do súboru *CMakeFiles/client.dir/link.txt* doplnili prepínač „*-lstdc++*“. Kompilácia klienta sa však prerušila kvôli chybe v zdrojovom kóde súboru *tls_sockets.cpp*. Error spôsobila funkcia *setsockopt*(), ktorá je v prostredí Windows definovaná s inými typmi vstupných argumentov. Pomocou pre-procesorovej podmienky sme kód rozvetvili a upravili sme parametre funkcie, tak aby spĺňala preddefinované typy pre konkrétny operačný systém. Následne sme ešte museli do súboru *tls_sockets.h* pridať hlavičkové súbory typické pre soketovú komunikáciu na Windows platforme. Keďže sme so soketmi pracovali v rámci bakalárskej práce, tak sme vedeli, že aplikácie vo Windowse ešte musia obsahovať WSADATA štruktúru, ktorá obsahuje informácie o soketovej komunikácii, pričom pre správnu funkcionalitu tejto štruktúry musíme knižnicu skompilovať s prepínačom „*-lws2_32*“. Aplikáciu klienta sa nám podarilo spustiť a overiť spojenie so serverom, pričom sme sa skúšali pripojiť k Rust serveru na OS Windows aj OS Linux Ubuntu. Pri testovaní sme

[53] `https://sh.rustup.rs`

[54] `https://doc.rust-lang.org/cargo/`

v niektorých prípadoch narazili na chybu označenú ako „Hostname NOT found in certificate“, ktorá bola spôsobená rozdielnymi podsieťami medzi našimi zariadeniami, keďže linuxové prostrednie u nás bežalo vo forme WSL2 (Windows Subsystem for Linux 2). Problém sme vyriešili vygenerovaním nových certifikátov.

Postup inštalácie pôvodnej verzie knižnice TiigerTLS sa nachádza v prílohe E a je tiež dostupný online na našom Gite[55].

Po úspešnej inštalácii sme sa oboznámili s funkciami klienta a servera. Dôležité pre nás hlavne bolo, akým spôsobom môžeme sprístupniť používanie post-kvantových algoritmov. Knižnica je napísaná veľmi prehľadne s množstvom komentárov, takže sme nemali problém zorientovať sa v zdrojových kódoch. Po krátkom štúdiu sme sa rozhodli overiť konektivitu knižnice s natívnymi aplikáciami klienta a servera v OpenSSL. Narazili sme pri tom na niekoľko problémov. Prvá chyba bola spojená s malou veľkosťou vyrovnávacej pamäte, kvôli ktorej potom klient ohlásil prerušenie spojenia. Problém sme vyriešili zväčšením tohto buffera. Ďalší problém sa javil o niečo komplikovanejšie a znovu súvisel s certifikátom. Nakoniec sme zistili, že OpenSSL 3.2, s ktorou sme knižnicu testovali, na rozdiel od verzie 3.1 vyžaduje, aby certifikáty spĺňali X.509v3 štandard, v ktorom musia certifikáty pre server obsahovať [*v3_ca*] modul s definovaným parametrom

```
[ v3_ca ]
basicConstraints = critical, CA:true.
```

S novými certifikátmi, ktoré obsahovali X.509v3 rozšírenie prebehla komunikácia úspešne. Posledná chyba sa vyskytla pri testovaní komunikácie s post-kvantovými algoritmami, kedy OpenSSL v spojení s oqs-providerom nepoznalo Kyber768 ako enkapsulačný algoritmus ani Dilitihum3 ako podpisovú schému. Problém bol spojený s identifikačným číslom algoritmov, ktoré boli rozdielne v TiigerTLS a oqs-providerovi. Zhodné IDs algoritmov nám umožnili úspešné pripojenie k OpenSSL serveru s využitím post-kvantových algoritmov.

Keďže knižnica MIRACL core obsahovala iba implementácie PQ algoritmov Kyber768 a Dilitihum3, tak sme sa rozhodli do TiigerTLS pridať inú knižnicu, aby sme zvýšili celkový počet podporovaných PQ algoritmov klienta. Konkrétne šlo o knižnicu liboqs, ktorú sme po experimentoch s OpenSSL už mali nainštalovanú na našom zariadení. Začali sme postupne, pričom najprv sme nahradili pôvodnú funkciu *KYBER768_keypair*() pre generovanie kľúčov a funkciu *KYBER768_decrypt*() na dekapsuláciu z knižnice MIRACL core novými funkciami *OQS_KEM_*-

[55] `https://git.kemt.fei.tuke.sk/js331zc/MastersThesis`

kyber_768_keypair() a *OQS_KEM_kyber_768_decaps*(). Následne sme vymenili funkciu *DLTHM_signature_3*() pre generovanie podpisu a funkciu *DLTHM_verify_3*() na overenie podpisu liboqs funkciami *OQS_SIG_dilithium_3_sign*() a *OQS_SIG_dilithium_3_verify*(). Pri dopĺňaní týchto funkcií sme museli dávať pozor na správnu typy vstupných parametrov a tiež na návratovú hodnotu. Príklad novej funkcie znázorňuje zdrojový kód 5.7. Knižnicu sme potom skompilovali podľa predošlého postupu s tým rozdielom, že sme k prekladu kódu pridali prepínač „*-loqs*“. Vytvorenú aplikáciu klienta sa nám potom podarilo pripojiť k OpenSSL serveru.

Zdrojový kód 5.7: Ukážka funkcie z knižnice liboqs na generovanie kľúčov pre post-kvantový enkapsulačný algoritmus CRYSTALS-Kyber768

```
1  /*
2      input:
3      group - nazov KEX\/KEM algoritmu
4      PK - struktura verejneho kluca
5      SK - struktura sukromneho kluca
6  */
7  void SAL_generateKeyPair(int group,octad *SK,octad *PK){
8  if (group==KYBER768){
9      // liboqs API funkcie vracaju hodnoty OQS_STATUS
10     OQS_STATUS rc = OQS_KEM_kyber_768_keypair((uint8_t*) PK->val, (
           uint8_t*) SK->val);
11         // generovanie klucov zlyhalo
12             if (rc != OQS_SUCCESS) {
13                     fprintf(stderr, "ERROR:
                           OQS_KEM_kyber_768_keypair failed!\n");
14             SK->len=0;
15             PK->len=0;
16         }
17         // generovanie klucov prebehlo uspesne
18         else{
19             SK->len=OQS_KEM_kyber_768_length_secret_key;
20             PK->len=OQS_KEM_kyber_768_length_public_key;
21         }
22     }
23 }
```

Zaradom sme pridávali do zdrojového kódu funkcie pre ďalšie post-kvantové enkapsulačné algoritmy, pričom našim cieľom bolo implementovať čo najviac dostupných algoritmov. Najprv sme doplnili chýbajúce verzie Kyberu, teda Kyber512 a Kyber1024, potom 3 verzie algoritmu HQC, 6 verzií FrodoKEM a 3 verzie BIKE algoritmu. Knižnica liboqs obsahuje aj NTRU-Prime a niekoľko verzií Classic McE-

liece, avšak pretože tieto algoritmy nie sú podporované knižnicou oqs-provider, tak by sme nemohli ich implementáciu otestovať, takže sme sa rozhodli ich vynechať. Pri každom doplnení algoritmu sme postupovali rovnako – vždy sme v hlavičkovom súbore TLS zadefinovali identifikačné číslo algoritmu a doplnili sme ho do zoznamu dostupných KEM algoritmov, ktorý predstavovala premenná *groups*[]. Potom sme pre daný algoritmus pridali do funkcie *SAL_generateKeyPair*() podmienku s funkciou na generovanie kľúčov a do funkcie *SAL_generateSharedSecret*() podmienku s funkciou na enkapsuláciu. Zároveň sme vždy overili, či netreba upraviť premenné

- TLS_MAX_KEX_PUB_KEY_SIZE, ktorá definuje maximálnu veľkosť verejného kľúča,
- TLS_MAX_KEX_SECRET_KEY_SIZE, ktorá definuje maximálnu veľkosť súkromného kľúča,
- TLS_MAX_KEX_CIPHERTEXT_SIZE, ktorá definuje maximálnu veľkosť šifrovaného textu.

Po pridaní každého algoritmu sme overili jeho funkčnosť pripojením na OpenSSL server, pričom klienta sme vždy skompilovali na Linuxe aj Windowse. Po chvíli sme narazili na chybu „Segmentation fault“ spôsobenú nevhodným prístupom k pamäti. Chybu sme okamžite opravili navýšením celkového počtu podporovaných algoritmov cez premennú TLS_MAX_SUPPORTED_GROUPS. Pri algoritme HQC-256 sme dodatočne museli zvýšiť premennú TLS_MAX_EXTENSIONS, pri algoritme FrodoKEM1344 sme zase upravili premennú TLS_MAX_IBUFF_SIZE. Ako posledné sme implementovali algoritmy BIKEL1, BIKEL2 a BIKEL5, ktoré nie sú na Windows platforme podporované, a preto sme ich skompilovanie podmienili linuxovým prostredím cez podmienku pre-procesora. Na záver sme doplnili všetky nové enkapsulačné algoritmy do informačných výpisov v súbore *tls_logger.cpp*. Do súboru *client.cpp* sme k informačného výpisu pridali aj veľkosti generovaných kľúčov každého algoritmu. Náš upravený klient, v tom čase s knižnicou liboqs 0.5.3, podporoval celkovo 15 post-kvantových enkapsulačných algoritmov na Linuxe, resp. 12 na Windows platforme.

Pri implementácii nových post-kvantových algoritmov pre digitálny podpis sme zvolili podobný postup ako s enkapsulačnými algoritmami. Začali sme s doplnením verzií algoritmu CRYSTALS-Dilithium, teda Dilithium2 a Dilithium5. Funkcie pre Dilithium2 už boli súčasťou klienta, pretože ich autor používal pri testovaní hybridného algoritmu p256_dilithium2, čo znamená, že nám stačilo MI-

RACL core funkcie na podpis a overenie podpisu iba jednoducho nahradiť funkciami z liboqs. Implementácia Dilithium5 už bola o niečo náročnejšia, pretože sme museli upraviť viac častí kódu mimo bezpečnostnej vrstvy aplikácie. Šlo o funkcie, ktoré sa týkali predovšetkým X.509 štruktúry certifikátov a extrakcie kľúčov z certifikátu. Postupne sa nám však podarilo do aplikácie implementovať podporu pre Dilithium5, ale tiež dve verzie algoritmu FALCON a 4 verzie algoritmu SPHINCS+[56].

Každému algoritmu sme v hlavičkovom súbore TLS zadefinovali identifikačné číslo, do bezpečnostnej vrstvy sme pridali algoritmus do zoznamov pre digitálne podpisy *sigAlgs*[] a *sigAlgsCert*[], kde sme tiež doplnili funkcie pre generovanie podpisu a overovanie podpisu spolu s podmienkou, kedy budú tieto funkcie použité. Avšak ako sme už spomínali, tak digitálne podpisy úzko súvisia s certifikátmi a ich X.509 štruktúrou. Do súboru *tls_x509.cpp* sme pre každý algoritmus museli podľa dokumentácie[57] knižnice oqs-provider definovať identifikačné číslo X.509 objektu. Museli sme ale vyriešiť problém ako má klient určiť, ktorý post-kvantový algoritmus použiť na overenie podpisu bez toho, aby sme museli meniť všetky funkcie. Aplikácia bola navrhnutá tak, aby z certifikátu najprv odvodila druh algoritmu, napr. ECC, RSA alebo PQ a až potom zisťovala presný algoritmus použitý na podpis, čím sa znížil počet prehľadávaných algoritmov. Pri ECC algoritmoch nás zaujala štruktúra *curve*, ktorá definovala číselnú reprezentáciu použitej krivky. Túto štruktúru sme použili pri našich PQ algoritmoch – ak klient zistil, že ide post-kvantový certifikát, tak na základe X.509 OIDs priradil k certifikátu aj číselnú hodnotu *curve*. Pôvodne sme predpokladali, že táto hodnota bude špecifická pre každý algoritmus, avšak pri testovaní sme zistili, že pri algoritmoch SPHINCS+-SHA2-128f-simple a SPHINCS+-SHAKE-128f-simple je táto hodnota rovnaká, pretože algoritmy sa líšia len v použitej hašovacej funkcii, čo znamená, že vysokoúrovňovo sa javia ako rovnaké kvôli zhodným veľkostiam kľúčov aj podpisov. Návratová hodnota funkcií pre overovanie podpisu je typu *bool*, čo znamená, že aplikácie vie či je podpis platný alebo nie. Na základe tejto vlastnosti sme mohli do kódu doplniť spoločnú hodnotu parametra *curve* pre oba algoritmy a pri splnení podmienky zavolať funkcie pre overenie podpisu oboch algoritmov súčasne.

[56] celkovo existuje 12 verzií, avšak len 4 sú podporované oqs-providerom, čo znamená, že by sme nemohli ďalšie verzie testovať

[57] `https://github.com/open-quantum-safe/oqs-provider/blob/main/ALGORITHMS.md#oids`

Zdrojový kód 5.8: Volanie funkcií na overovanie podpisu pri splnení druhu certifikátu a hodnoty premennej curve

```
1 if (st.type== X509_PQ && st.curve==USE_DILITHIUM2)
2    res=SAL_tlsSignatureVerify(DILITHIUM2,CERT,SIG,PUBKEY);
3 if (st.type== X509_PQ && st.curve==USE_DILITHIUM3)
4     res=SAL_tlsSignatureVerify(DILITHIUM3,CERT,SIG,PUBKEY);
5 if (st.type== X509_PQ && st.curve==USE_DILITHIUM5)
6     res=SAL_tlsSignatureVerify(DILITHIUM5,CERT,SIG,PUBKEY);
```

Testovanie pridaných algoritmov bolo o niečo zdĺhavejšie, pretože sme vždy museli generovať nový certifikát. Na zrýchlenie generovania sme si pripravili jednoduchý shell skript, ktorý obsahoval všetky príkazy na generovanie PQC certifikátov, pričom názov certifikátu bral ako vstupný parameter z príkazového riadku. Testovací skript s názvom *gen_test_cert_ubuntu.sh* je dostupný na našom Gite a je súčasťou prílohy A.

Pri testovaní algoritmu SPHINCS+-SHA2-192f-simple sme narazili na chybu, pri ktorej klient prerušil spojenie so serverom kvôli starému, a teda neplatnému certifikátu a to aj napriek tomu, že certifikát bol nový a platný. Funkcia vracala nezmyselný rok exspirácie certifikátu, čo nedávalo zmysel, keďže pri predošlých certifikátoch sme s týmto nemali problém. Je pravda, že podpis tohto certifikátu a celkovo zreťazený certifikát je niekoľkonásobne väčší ako pri predchádzajúcich algoritmoch, no ani zväčšenie vyrovnávacej pamäte nám tento problém nepomohlo vyriešiť. Dôvod tejto chyby sa nám nepodarilo odhaliť, a preto sme podporu tohto algoritmu z nášho klienta odstránili.

Pri dopĺňaní nových algoritmov sme sa znovu stretli s potrebou upraviť niektorý premenné protokolu. Pozor sme hlavne dávali na premenné:

- TLS_MAX_SIG_PUB_KEY_SIZE, ktorá definuje maximálnu veľkosť verejného kľúča,
- TLS_MAX_SIG_SECRET_KEY_SIZE, ktorá definuje maximálnu veľkosť súkromného kľúča,
- TLS_MAX_SIGNATURE_SIZE, ktorá definuje maximálnu veľkosť podpisu.

Navýšili sme tiež počet podporovaných algoritmov pre podpis cez premennú TLS_MAX_SUPPORTED_SIGS a pri algoritme SPHINCS+-SHAKE-128f-simple sme museli upraviť premennú TLS_MAX_IBUFF_SIZE, kvôli veľkosti zreťazeného certifikátu. Na záver sme pridali do súborov *logger.cpp* a *client.cpp* informačné výpisy o nových PQC algoritmoch. Celkovo náš upravený klient s knižnicou liboqs 0.5.3 podporoval 8 PQC algoritmov pre digitálny podpis.

Pri stálom upravovaní a opakovanej kompilácii celej aplikácie na rôznych operačných systémoch sme museli vždy zmazať všetky vygenerované súbory a znovu ich podľa návodu vygenerovať a skompilovať. Návod nebol náročný technicky, ale skôr časovo, a preto sme sa ho rozhodli upraviť. Zo súboru *CMakeLists.txt* sme odstránili nepotrebné podmienky a parametre. Linkovanie knižníc, teda „*-loqs*“, „*-lstdc++*“, resp. „*-lws2_32*“, sme doplnili priamo do tohto súboru. Aktualizovali sme tiež celý postup kompilácie, ktorý je súčasťou prílohy F a je dostupný online na našom Gite[58].

Bezpečnostná vrstva aplikácie zároveň obsahuje funkcie na generovanie náhodných čísel. Tieto funkcie sme viackrát upravovali až do konečného stavu, kedy sa náhodne generované číslo vrátené funkciou *OQS_randobytes*() použije ako počiatočná hodnota generátora knižnice MIRACL core.

Pri finalizácii celého projektu sme tiež čiastočne prečistili zdrojové kódy od funkcií, ktoré autor používal pri testovaní na Arduine. Odstránili sme nepotrebné súbory a do zložky */sal/* sme pripravili zložky s vygenerovanými súbormi knižnice MIRACL core pre rôzne operačné systémy. Do kódov sme doplnili detailné komentáre ku každej zmenenej funkcii a tiež README so stručným postupom kompilácie a prehľadom úprav v každom súbore. Okrem toho sme k projektu pridali aj súbor CHANGES, ktorý obsahuje prehľad všetkých našich úprav, ktoré sme urobili počas celého riešenia našej práce.

Pár týždňov pred odovzaním práce vyšla nová verzia knižnice liboqs 0.10.0[59], do ktorej boli doplnené algoritmy ML-KEM a ML-DSA. Na základe novej verzie knižnice sme sa rozhodli doplniť nášho klienta aj o tieto algoritmy a použiť ich tiež pri ďalších experimentoch. Implementácia troch verzií enkapsulačného algoritmu ML-KEM sme realizovali rovnakým spôsobom ako pri predošlých algoritmoch, ale pri ML-DSA sme narazili na problém. Pôvodne definovaný parameter *curves* pre verzie algoritmu Dilitihum bol zhodný s verziami algoritmu ML-DSA, čo náš neprekvapilo, keďže ide o takmer rovnaký algoritmus, no donútilo nás to premýšľať nad postupom implementácie, pretože volanie viacerých funkcií tak ako pri algoritmoch SPHINCS+-SHA2-128f-simple a SPHINCS+-SHAKE-128f-simple nám prišlo nevhodné a neefektívne. Podľa toho, ako bola celá aplikácia napísaná sme mali na výber 2 možnosti. Buď algoritmy od seba odlíšiť definovaním použitej hašovacej funkcie, ktorá je napríklad pri SPHINCS+ verziách odlišná, no v prípade Dilithia a ML-DSA rovnaká, čo by však neprekážalo, pretože hašovanie sa realizuje na úrovni API funkcie a nie priamo v TLS. Druhou možnosťou bolo

[58] `https://git.kemt.fei.tuke.sk/js331zc/MastersThesis`
[59] ku dňu 23.3.2024

použitie už spomínanej premennej *curves*. Aby to nebolo zbytočne mätúce, tak sme sa rozhodli pre druhú možnosť, pričom pre každý PQ algoritmus na digitálny podpis sme vytvorili samostatnú konštantu, ktorú podľa použitého certifikátu používame v premennej *curves* a na základe ktorej môžeme volať funkcie na overovanie podpisu pre jeden konrétny algoritmus. Týmto spôsobom sme rozlíšili Dilithium od ML-DSA a tiež SPHINCS+-SHA2-128f-simple od SPHINCS+-SHAKE-128f-simple. Okrem toho sme dodatočne aktualizovali niektoré identifikátory algoritmov, ktoré boli upravené v rámci novej verzie knižnice oqs-provider.

Všetky pridané algoritmy sme experimentálne otestovali prostredníctvom lokálneho OpenSSL serveru. Celkovo náš klient podporuje 18 PQC enkapsulačných algoritmov na Linuxe, 15 na Windows platforme a 11 PQC algoritmov pre digitálny podpis.

5.6 Certifikačná autorita pre generovanie post-kvantových certifikátov

Proces generovania, podpisovania a overovania certifikátov úzko súvisí s TLS komunikáciou. Keďže sme si pri práci s balíkom klient/server pre predmet BPS všimli, že tento proces je trochu ťažkopádny a neodzrkadľuje skutočný priebeh generovania a podpisovania certifikátov, tak sme sa rozhodli vytvoriť vlastnú vzorovú certifikačnú autoritu CA.

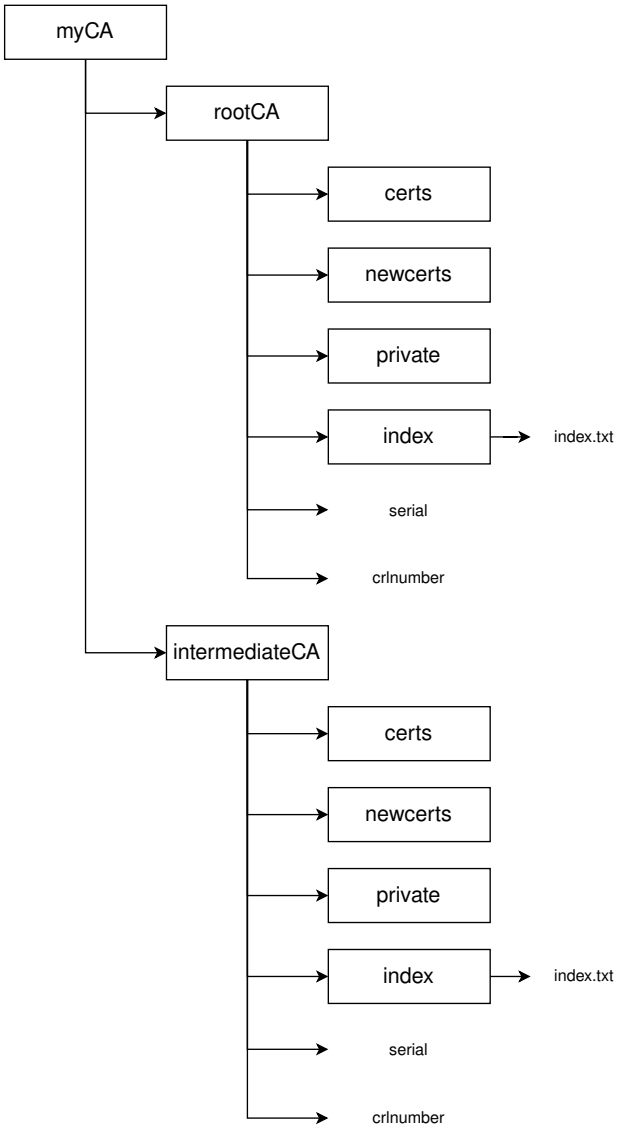
Pripravili sme balík súborov, ktorý obsahuje:

- konfiguračné súbory pre hlavnú (koreňovú)[60] a sekundárnu[61] CA,
- skript *gen_CA.bat* na generovanie celej štruktúry CA vrátane vygenerovania a podpísania vzorových zreťazených certifikátov,
- skript *gen_client_certificate* na čiastočnú automatizáciu generovania a podpisovania certifikátov pre klienta.

Skript *gen_CA.bat* po spustení vytvorí celú štruktúru priečinkov. Pri návrhu tejto štruktúry sme sa snažili nájsť nejakú štandardizovanú alebo všeobecne platnú

[60] root
[61] intermediate

verziu, ale narazili sme na rôzne verzie[62][63][64][65], a preto sme sa rozhodli vytvoriť vlastnú štruktúru. Aj keď sme sa pôvodne snažili minimalizovať celkový počet priečinkov, tak postupom času sme zistili, že pri procese podpisovania certifikátov sú automaticky generované ďalšie súbory, ktoré sme kvôli lepšej prehľadnosti nakoniec presunuli do samostatných priečinkov. Naša finálna štruktúru priečinkov a súborov je zobrazená na obrázku 5.3.



Obr. 5.3: Štruktúra priečinkov a súborov nášho návrhu certifikačnej autority

Pri príprave skriptu sme použili klasický príkazový riadok na Windows za-

[62] https://stackoverflow.com/questions/15075488/how-to-structure-a-ca-directory-for-usage-with-openssl

[63] https://openssl-ca.readthedocs.io/en/latest/introduction.html

[64] https://superuser.com/questions/126121/how-to-create-my-own-certificate-chain

[65] https://www.golinuxcloud.com/openssl-create-certificate-chain-linux/

riadení, takže použité príkazy nemusia fungovať v linuxovom prostredí alebo v aplikácii Windows PowerShell. Avšak špeciálne pre PowerShell sme do skriptu doplnili alternatívne príkazy, vrátane zaujímavých postrehov, vo forme komentárov. Po vygenerovaný priečinkov skopíruje skript konfiguračné súbory do hlavných priečinkov certifikačných autorít. Zvyšok skriptu potom obsahuje klasické OpenSSL príkazy.

Skript najprv prostredníctvom príkazu *openssl req* súčasne vygeneruje súkromný kľúč a certifikát pre hlavnú koreňovú CA, pričom na definovanie X.509v3 vlastností a identifikačných informácii (napr. názov, organizácia, štát, email) použije práve konfiguračný súbor pre hlavnú CA. Takýto príkaz môžeme vyzerať napríklad takto:

Súkromný kľúč a certifikát sú potom uložené do samostatných priečinkov. V skutočných podmienkach musí mať priečinok so súkromnými kľúčmi obmedzený prístup kvôli bezpečnosti. Rovnaký príkaz s inými parametrami potom vygeneruje súkromný kľúč a certifikát pre sekundárnu CA, pričom tentokrát použije druhý konfiguračný súbor.

Certifikát sekundárnej CA je potom pomocou príkazu *openssl ca* podpísaný súkromným kľúčom hlavnej CA. Do skriptu sme doplnili aj príkazy *openssl verify* na overenie podpisu, aby bol používateľ priebežne informovaný a v prípade chyby vedel, kde približne ju hľadať. Vykonaním príkazu dôjde k podpísaniu certifikátu a zvýši sa počítadlo v súbore *serial*. Ide o jedinečné sériové číslo, ktoré sa používa na identifikáciu konkrétneho certifikátu. Zároveň sa aktualizuje databáza v súbore *index.txt*, ktorá zaznamenáva informácie o podpísanom certifikáte, ako sú status, dátum platnosti, sériove číslo a identifikačné informácie (napr. názov, organizácia, štát, email). Po overení podpisu, spojí skript certifikáty koreňovej a sekundárnej CA, čím vznikne zreťazený certifikát sekundárnej CA.

Analogicky sú potom vygenerované súkromné kľúče pre server a klienta, pričom identifikačné údaje už nie sú načítane z konfiguračných súborov, ale sú priamo súčasťou príkazu. Vytvorené certifikáty sú podpísané súkromným kľúčom sekundárnej CA, ale na overenie podpisu sa použije zreťazený certifikát.

Do skriptu sme zo zaujímavosti doplnili aj príklad príkazov na zneplatnenie certifikátu. V podstate ide o odvolanie platnosti certifikátu, ktoré môže nastať pri kompromitácii súkromného kľúča, alebo ak bol certifikát vygenerovaný podvodníkom, alebo sa identifikačné údaje certifikátu zmenili. Na zrušenie platnosti sa používa príkaz *openssl ca* s prepínačom „*-revoke*“. Zrušením platnosti dôjde k zvýšeniu počítadla zneplatnených certifikátov *crlnumber*, k zvýšeniu hodnoty *serial* a k pridaniu zápisu do databázy *index.txt*.

Skript *gen_client_certificate* obsahuje iba vzorové príkazy na generovanie a podpisovanie certifikátov určených pre klienta. Pri všetkých príkazoch sú do detailov opísané použité prepínače a vstupné parametre. Tento skript však slúži len na čiastočnú automatizáciu pri tvorbe certifikátov, pretože používateľ musí vždy upraviť požadovaný algoritmus použitý pri generovaní kľúča, ale tiež identifikačné údaje certifikátu (napr. názov, organizácia, štát, email). Generovanie certifikátu s rovnakými parametrami skončí chybovou hláškou. Zároveň správny výsledok príkazov predpokladá umiestnenie súkromného kľúča na konkrétnom mieste a s presne definovaným názvom. Zatiaľ čo pri generovaní je súkromný kľúč súčasťou príkazu, tak pri podpisovaní je jeho umiestnenie definované priamo v konfiguračnom súbore sekundárnej certifikačnej autority.

Skripty *gen_CA.bat* a *gen_client_certificate* sú súčasťou prílohy A. Vzorový certifikát a informácie o ňom sú zase súčasťou prílohy G. Všetky súčasti vzorovej CA sú tiež dostupné online na našom Gite[66].

[66] `https://git.kemt.fei.tuke.sk/js331zc/MastersThesis`

6 Experimentálne výsledky

Pripojenie TiigerTLS k testovaciemu Open Quantum Safe serveru

Po dokončení našej upravenej verzie TiigerTLS sme sa rozhodli overiť konektivitu našej aplikácie aj s iným serverom ako bol lokálny OpenSSL server. Skúšali sme hľadať či existuje nejaký verejný testovací server, ktorý umožňuje testovať aj PQC algoritmy. Súčasťou projektu Open Quantum Safe je presne takýto testovací server. Podľa informácii z oficiálnej stránky[1] podporuje server PQC algoritmy na enkapsuláciu kľúčov a digitálne podpisy a to v rôznych kombináciách pre každý port, pričom nastavenie TLS parametrov ako IDs alebo X.509 OIDs má byť rovnaké ako pri knižnici oqs-provider. Testovanie sme realizovali dvakrát, pričom pri druhom pokuse sme použili novšiu verziu knižnice liboqs 0.10.0 a tiež už bola dostupná nová verzia oqs-providera 0.6.0[2], ktorý síce nie je pre našu aplikáciu potrebný, ale na základe neho má server definované IDs algoritmov, čo sa odrazilo aj na výsledkoch testovania.

Pri prvom testovaní sme celkovo overili 40 pripojení k testovaciemu serveru pri použití rôznych kombinácií post-kvantových algoritmov. Pri meraní sme použili zariadenie s OS Ubuntu 22.04, aby sme mohli otestovať aj použitie algoritmu BIKE, ktorý nie je na Windows platforme dostupný. Pri testovaní sme narazili na dva problémy. Prvý problém sa týkal algoritmu HQC, pričom použitie akejkoľvek verzie tohto enkapsulačného algoritmu neumožnilo nadviazať spojenie, pretože TLS nedokázal rozpoznať tento algoritmus. Keďže sme funkcionalitu HQC testovali lokálne na OpenSSL serveri, čo znamenalo, že chyba nie je v našej aplikácii ani v dokumentácii knižnice, tak sme predpokladali, že chyba bola na strane testovacieho serveru OQS, ktorý podľa nás nemal správne definované ID pre HQC algoritmus. Na druhý problém sme narazili pri algoritme FALCON pre digitálne podpisy, pričom klient nám pri pripojení vždy ohlásil, že podpis cer-

[1] `https://test.openquantumsafe.org/`

[2] ku dňu 12.4.2024

tifikátu nie je v poriadku. Aj keď nevieme presne povedať prečo tento problém nastal, tak funkcionalitu algoritmu FALCON sme tiež testovali lokálne, a preto si dovolíme tvrdiť, že znovu šlo o chybu na strane serveru, ktorú nedokážeme ovplyvniť.

Pri druhom testovaní sme realizovali celkovo 66 pokusov o pripojenie k testovaciemu server a všetky dopadli pozitívne. Na základe tohto testovania sa potvrdila naša pôvodná domnienka o chybe na strane serveru z predošlého testovania, keďže opakované pokusy o pripojenie s použitím algoritmov HQC a FALCON prebehli úspešne.

Celý záznam testovania je súčasťou prílohy H.

Meranie času PQC algoritmov

O tom, že post-kvantové algoritmy pracujú s výrazne väčšími kľúčmi, čo môže mať vplyv na celkovú dobu TLS handshaku, sme už v tejto práci spomínali. Na internete sú dostupné rôzne publikácie, ktoré porovnávajú výpočtovú náročnosť a priebeh nadviazanie spojenia v TLS protokole pri použití klasických pre-kvantových algoritmoch a post-kvantových algoritmov. Často ide o meranie počtu CPU cyklov pri výpočtoch na konkrétnom zariadení, meranie celkového času nadviazania spojenia alebo množstva prenesených bajtov medzi klientom a serverom.

Podobný experiment realizoval aj Micheal Scott kvôli čomu do knižnice TiigerTLS implementoval funkcie na meranie času pri generovaní kľúčov KEX/KEM algoritmov, pri generovaní podpisov a pri overovaní podpisov. Tieto funkcie sme sa rozhodli použiť pri experimentoch s PQC algoritmami, ktoré sme do našej verzie TiigerTLS doplnili.

Meranie času generovania KEX/KEM kľúčov

Meranie času generovania kľúčov enkapsulačných algoritmov bolo pomerne jednoduché a jasné. Klient po spustení s prepínačom „-s“ z príkazového riadku postupne volal funkcie KEM algoritmov na generovanie kľúčov a ku každému algoritmu vypisoval čas generovania a veľkosti kľúčov v bajtoch. Nebolo vôbec prekvapivé, že najdlhší čas generovania mali verzie algoritmu FrodoKEM, keďže majú najväčšie dĺžky kľúčov. Napríklad medzi Kyber1024 a FrodoKEM-1344-SHAKE, ktoré sú na rovnakej bezpečnostnej úrovni L5, ide o 13-krát väčší súkromný kľúč a takmer 60-krát dlhší čas generovania. Generovanie kľúčov je však v prípade enkapsulačných algoritmov realizované zo strany klienta, a preto nemusíme riešiť kumulovanie tohto času na strane servera.

Meranie sme zopakovali tesne pred odovzdaním po aktualizácii knižnice liboqs na verziu 0.6.0, aby sme porovnali či nastavali nejaké zmeny v časoch generovania kľúčov a zároveň, aby sme doplnili časy k algoritmom ML-KEM a ML-DSA. Výraznou zmenou bol viac ako 10-násobný nárast celkového času generovania kľúčov pri verziách algoritmu HQC. To je spôsobené tým, že v novej verzii bola aktualizovaná implementácia algoritmu HQC[3] z pôvodnej implementácie tretieho kola NIST štandardizácie na novšiu verziu štvrtého kola. Generovania kľúčov HQC algoritmu teraz dosahuje podobné výsledky ako SHAKE verzie algoritmu FrodoKEM a to aj napriek pomerne veľkým rozdielom vo veľkostiach kľúčov. Zmeny v implementácii tiež ovplyvnili veľkosti kľúčov, súkromný kľúč sa zväčšil v každej verzii približne o 20 bajtov. Merania pre ML-KEM a ML-DSA ukázali, že dosahujú takmer zhodné výsledky ako ich predchodcovia Kyber a Dilithium.

Musíme však spomenúť, že merania sme realizovali na veľkom stolnom počítači s vysokým výkonom procesora (CPU – Central Processing Unit), grafickou kartou (GPU – Graphics Processing Unit) a množstvom dostupnej operačnej pamäte (RAM – Random-Access Memory). Preto sme sa rozhodli urobiť nové merania na odlišnom zariadení s oveľa nižším výkonom. Základné parametre oboch počítačov sú znázornené v tabuľke 6.1

Tabuľka 6.1: Základné parametre použitých počítačov pri experimentálnych meraniach celkového času generovania kľúčov vybraných algoritmov

Zariadenie	**CPU**	**GPU**	**RAM**
počítač A	AMD Ryzen 5 7600 3,8 GHz	NVIDIA GeForce RTX 4060	32 GB
počítač B	AMD A12-9720P 2,7 GHz	AMD Radeon R7	8 GB

Výsledkom meraní bol takmer 3-násobný nárast celkového času generovania kľúčov, pri niektorých algoritmoch až takmer 5-násobný. Porovnanie výsledkov všetkých meraní zobrazuje tabuľka 6.2. Pre porovnanie sme do tabuľky pridali tiež algoritmy x25519, secp256r1 a secp384r1, aby čitateľ mohol porovnať výsledky pre-kvantových a post-kvantových algoritmov. Merania sme realizovali na Windows platforme, a preto tabuľka neobsahuje výsledky meraní pre algoritmus BIKE. Detailný prehľad výsledkov je súčasťou prílohy I, dostupný tiež online na našom Gite[4].

[3] `https://github.com/open-quantum-safe/liboqs/pull/1585`

[4] `https://git.kemt.fei.tuke.sk/js331zc/MastersThesis`

[5] nezávislé od knižnice liboqs

Tabuľka 6.2: Výsledky meraní času generovania kľúčov rôznych KEX/KEM algoritmov pri použití rozdielnych verzií knižnice liboqs spolu s porovnaním výsledkov dvoch rozdielnych počítačov

	počítač A		počítač B
	liboqs 0.5.3	liboqs 0.6.0	liboqs 0.6.0
x25519[5]	0,13 s		0,30 s
secp256r1[5]	0,25 s		0,59 s
secp384r1[5]	0,53 s		1,51 s
KYBER512	0,09 ms	0,09 ms	0,35 ms
KYBER768	0,11 ms	0,10 ms	0,41 ms
KYBER1024	0,12 ms	0,12 ms	0,44 ms
MLKEM512	–	0,05 ms	0,20 ms
MLKEM768	–	0,07 ms	0,24 ms
MLKEM1024	–	0,09 ms	0,28 ms
HQC128	0,12 ms	1,32 ms	2,70 ms
HQC192	0,22 ms	3,85 ms	7,29 ms
HQC256	0,37 ms	6,95 ms	14,93 ms
FRODO640AES	0,23 ms	0,22 ms	1,20 ms
FRODO640SHAKE	1,50 ms	1,45 ms	4,59 ms
FRODO976AES	0,41 ms	0,39 ms	1,33 ms
FRODO976SHAKE	3,29 ms	3,20 ms	9,28 ms
FRODO1344AES	0,67 ms	0,66 ms	3,58 ms
FRODO1344SHAKE	5,95 ms	5,76 ms	15,80 ms

Meranie času overovania certifikátu

Pri pokuse merať trvanie procesu overovania podpisov jednotlivých PQC algoritmov nám aplikácia často vracala nulové výsledky. Tento problém bol spôsobený funkciou *clock*(), ktorá nemá dostatočné rozlíšenie. Funkciu sme nahradli inštrukciou RDTSC[6] (Read Time-Stamp Counter), ktorá vracia 64-bitovú hodnotu časovej pečiatky procesora. So znalosťou taktovacej rýchlosti nášho procesora sme rozdiel hodnoty RDTSC pred začiatkom overovania certifikátu a hodnoty RDTSC po overení certifikátu mohli vyjadriť vo forme milisekúnd.

Celkovo sme merali trvanie overovania certifikátov pri použití 13 podpisových schém. Začali sme s klasickými algoritmami RSA_PSS_RSAE_SHA256 a ECDSA_SECP384R1_SHA384. Následne sme merali časy pri použití ďalších 11 PQC algoritmov, ktoré náš klient podporuje. Klient pri nadväzovaní spojenia cez TLS pro-

[6] `https://learn.microsoft.com/en-us/cpp/intrinsics/rdtsc?view=msvc-170`

tokol overuje celkovo 3 certifikáty – certifikát serveru, certifikát sekundárnej CA, certifikát koreňovej CA. Vo výsledku sme teda všetky 3 hodnoty sčítali. Pri každom algoritme sme vykonali 3 merania, pričom v tabuľke 6.3 sú uvedené najnižšie namerané časy. Pri meraniach sme použili dva rôzne počítače, podobne ako pri predchádzajúcom meraní, aby sme dokázali porovnať výpočtovú náročnosť testovaných algoritmov. Detailný prehľad výsledkov meraní je súčasťou prílohy J.

Tabuľka 6.3: Najnižšie hodnoty nameraných časov pri overovaní certifikátov zo strany klienta

Algoritmus	počítač A	počítač B
RSA_PSS_RSAE_SHA256	0,56 ms	1,22 ms
ECDSA_SECP384R1_SHA384	1,67 ms	1,76 ms
Dilithium2	0,88 ms	2,01 ms
Dilithium3	0,96 ms	2,41 ms
Dilithium5	1,13 ms	2,92 ms
FALCON512	0,82 ms	1,77 ms
FALCON1024	0,88 ms	1,84 ms
ML-DSA44	0,89 ms	2,40 ms
ML-DSA65	0,98 ms	2,46 ms
ML-DSA87	1,11 ms	2,87 ms
SPHINCS+-SHA2-128f-simple	3,6 ms	11,85 ms
SPHINCS+-SHA2-128s-simple	1,81 ms	4,99 ms
SPHINCS+-SHAKE-128f-simple	5,2 ms	16,16 ms

Napriek tomu, že namerané hodnoty sa pohybujú v milisekundách, tak aj pri porovnaní najlepších časov PQC algoritmov s výsledkami pre-kvantových algoritmov môžeme vidieť niekoľkonásobný nárast celkového času pri overovaní certifikátov. Zároveň môžeme pozorovať aj pomerne veľký rozdiel medzi výsledkami meraní rovnakých algoritmov na rozdielnych počítačoch s iným výpočtovým výkonom. Naše výsledky tak potvrdzujú tvrdenie, že medzi hlavné výzvy pri integrácii PQC algoritmov do TLS protokolu patrí popri analýze bezpečnosti tiež skúmanie optimalizácie jednotlivých algoritmov.

7 Záver

Cieľom tejto práce bolo analyzovať post-kvantovú kryptografiu a možnosti jej implementácie do TLS protokolu pri využití TCP/IP komunikácie v modeli klient-server.

V teoretickej časti sme sa venovali problematike kvantových počítačov a ich vplyvu na kryptografiu, čím sme vysvetlili dôvody vzniku samotnej post-kvantovej kryptografie. Predstavili sme rôzne druhy PQC algoritmov podľa ich matematických základov a opísali sme priebeh celej štandardizácie vo forme NIST súťaže. Venovali sme sa tiež spôsobu a dôsledkom integrácie PQC algoritmov na výmenu kľúčov a PQC algoritmov pre digitálne podpisy do TLS protokolu. Následne sme rozobrali matematické postupy, parametre a bezpečnosť predbežných štandardov víťazných PQC algoritmov.

V rámci praktickej časti tejto práce bolo našou úlohou analyzovať a navrhnúť aplikáciu pre štandardnú TCP/IP komunikáciu na základe architektúry klient-server s využitím TLS protokolu a PQC certifikátov. Pri implementácii sme sa zamerali na minimalizáciu celkového množstva použitých softvérových nástrojov a kryptografických knižníc.

Prvým výsledkom praktickej časti bolo integrovanie PQC algoritmov do kryptografickej knižnice OpenSSL. Okrem už neaktuálnej verzie OpenSSL 1.1.1 s podporou pre PQC algoritmy, označenej ako OQS-OpenSSL, sme vytvorili inštalačný postup pre knižnice liboqs a oqs-provider, ktoré je možné integrovať do najnovšej verzie OpenSSL 3.3.0. Na účely výučby v rámci predmetov BPS a BIKS sme pripravili demonštračné implementácie klienta a serveru v jazyku C, ktoré po inštalácii a aktivácii oqs-providera umožňujú používať PQC algoritmy v TLS komunikácii. Aplikácie klienta a serveru sú nezávislé od použitej platformy. Pre pohodlie používateľov sme k balíku súborov pridali už skompilovanú knižnicu oqs-provider v aktuálne najnovšej verzii 0.6.0, kompatibilnú s 32-bitovým systémom, v ktorom je realizovaná výučba.

Druhým výsledkom je upravená verzia knižnice TiigerTLS. Pôvodná knižnica TiigerTLS je minimalistická implementácia TLS protokolu v aplikácii klienta na-

písaná v jazyku C++, ktorá používa funkcie pre kryptografické primitíva, vrátane vybraných PQC algoritmov, z knižnice MIRACL core. TiigerTLS sme zlinkovali s knižnicou liboqs, ktorá obsahuje viac druhov PQC algoritmov, je pravidelne aktualizovaná a pomerne malá. Integráciou liboqs sme tak rozšírili podporu klienta o nové PQC algoritmy. Celkovo ide o 18 PQC enkapsulačných algoritmov na Linuxe, resp. 15 na Windows platforme a 11 PQC algoritmov pre digitálny podpis. Pri úpravach zdrojových kódov, ktorými sme rozšírili kryptografické funkcie klienta, sme tiež doplnili funkcie na kompatibilitu s OS Windows. Tiež sme upravili súbor používaný pri automatizácii kompilácie, čím sme výrazne tento proces zjednodušili. Podporu nových PQC algoritmov sme overovali prostredníctvom pripojení k lokálnemu OpenSSL serveru, ale tiež pomocou testovacieho serveru, ktorý je súčasťou projektu OQS. Pri experimentálnych meraniach časovej náročnosti procesov generovania kľúčov pri enkapsulačných algoritmoch a overovania certifiktátov pri schémach pre digitálny podpis sme používali pôvodné funkcie aplikácie. Vykonali sme 42 testov s generovaním KEX/KEM kľúčov a poukázali sme na rozdiely medzi pre-kvantovými a post-kvantovými algoritmami, rovnako ako na rozdiely vo výsledkoch meraní post-kvantových algoritmov na počítačoch s rozdielynm výkonom. Zatiaľ čo pri generovaní kľúčov bolo časové rozlíšenie pôvodnej funkcie dostatočné, tak pri overovaní certifikátov sme museli do aplikácie pridať RDTSC inštrukciu, ktorá umožňuje merať čas na základe registra CPU, čím sme docielili vysoké rozlíšenie meraní, založené na taktovacej rýchlosti CPU. Týmto spôsobom sme aplikáciu pripravili pre ďalšie experimenty s novými verziami implementácii PQC algoritmov alebo úplne novými PQC algoritmami. Realizovali sme tiež 26 testov a pri porovnaní výsledkov sme potvrdili predpoklad, že overovanie PQC certifikátov trvá niekoľkonásobne dlhšie ako overovanie klasických certifikátov, pričom sme mohli pozorovať výrazný nárast výsledných časov pri meraniach na počítačoch s odlišným výpočtovým výkonom.

Oba vytvorené balíky umožňujú používať PQC algoritmy v TLS protokole. Zároveň majú pomerne jasnú a jednoduchú štruktúru knižníc. Inštalačné postupy všetkých súčastí sme čo najviac zjednodušili a použili sme pri tom kompaktný zoznam nástrojov. V linuxovom prostredí je inštalácia potrebných nástrojov pomerne priamočiara, v prípade Windows platformy nám zase stačí balík nástrojov WinLibs.

Keďže so zabezpečenou komunikáciou prostredníctvom TLS protokolu úzko súvisia aj certifikáty, tak v rámci praktickej časti sme tiež pripravili vzorovú certifikačnú autoritu, ktorá umožňuje generovať a podpisovať PQC certifikáty. Celú štruktúru CA si používateľ môže vygenerovať pripraveným skriptom a pomocou

ďalšieho skriptu môže generovať a podpisovať nové certifikáty pre klienta.

Rozšíriť túto prácu by bolo možné analýzou ďalších PQC algoritmov alebo ich testovaním spolu s našimi aplikáciami na malých zariadeniach alebo na vstavaných procesoroch. Medzi hlavné výzvy PQC algoritmov tiež patrí skúmanie odolnosti voči útokom s využitím postranných kanálov.

Literatúra

1. STANDARDS, National Institute of; TECHNOLOGY. *Advanced Encryption Standard (AES)*. 2001. Tech. spr. U.S. Department of Commerce. Dostupné z DOI: https://doi.org/10.6028/NIST.FIPS.197-upd1. online 15.4.2024 - https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197-upd1.pdf.
2. TURAN, Meltem Sonmez; MCKAY, Kerry; CHANG, Donghoon; BASSHAM, Lawrence E.; KANG, Jinkeon; WALLER, Noah D.; KELSEY, John M.; HONG, Deukjo. Status Report on the Final Round of the NIST Lightweight Cryptography Standardization Process. 2023. Dostupné z DOI: https://doi.org/10.6028/NIST.IR.8454. online 11.3.2024 - https://nvlpubs.nist.gov/nistpubs/ir/2023/NIST.IR.8454.pdf.
3. SCHNEIDER, Josh; SMALLEY, Ian. What is quantum cryptography? *IBM*. 2023. online 7.3.2024 - https://www.ibm.com/topics/quantum-cryptography.
4. XU, Feihu; MA, Xiongfeng; ZHANG, Qiang; LO, Hoi-Kwong; PAN, Jian-Wei. Secure quantum key distribution with realistic devices. *Reviews of Modern Physics*. 2020. Dostupné z DOI: https://doi.org/10.48550/arXiv.1903.09051. online 7.3.2024 - https://arxiv.org/pdf/1903.09051.pdf.
5. NSA CYBERSECURITY. Quantum Key Distribution (QKD) and Quantum Cryptography (QC). 2021. online 7.3.2024 - https://www.nsa.gov/Cybersecurity/Quantum-Key-Distribution-QKD-and-Quantum-Cryptography-QC/.
6. KUMAR, Ajay; GARHWAL, Sunita. State-of-the-Art Survey of Quantum Cryptography. *Archives of Computational Methods in Engineering*. 2021. Dostupné z DOI: https://doi.org/10.1007/s11831-021-09561-2. online 7.3.2024 - https://link.springer.com/article/10.1007/s11831-021-09561-2.

7. FEDERAL OFFICE FOR INFORMATION SECURITY. Quantum-safe cryptography - fundamentals, current developments and recommendations. 2021. online 7.3.2024 - https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/Brochure/quantum-safe-cryptography.pdf?__blob=publicationFile&v=4.
8. JORDAN, Stephen. Quantum Algorithm Zoo. 2022. online 7.3.2024 - https://quantumalgorithmzoo.org/.
9. NÁRODNÍ ÚŘAD PRO KYBERNETICKOU A INFORMAČNÍ BEZPEČNOST. KVANTOVÁ HROZBA A KVANTOVĚ ODOLNÁ KRYPTOGRAFIE Příloha k dokumentu: Minimální požadavky na kryptografické algoritmy. 2023. online 7.3.2024 - https://nukib.gov.cz/download/uredni_deska/Priloha%20-%20Minimalni%20pozadavky%20na%20kryptograficke%20algoritmy.pdf.
10. HOVANOVÁ, Tatiana. *Kvantově bezpečná kryptografie*. 2019. Diz. pr. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií. online 8.3.2024 - https://www.vut.cz/www_base/zav_prace_soubor_verejne.php?file_id=192221.
11. CHUANG, Isaac L.; GERSHENFELD, Neil; KUBINEC, Mark. Experimental Implementation of Fast Quantum Searching. *Physical Review Letters*. 1998. Dostupné z DOI: 10.1103/PhysRevLett.80.3408. online 8.3.2024 - https://link.aps.org/doi/10.1103/PhysRevLett.80.3408.
12. DAVIS, Robert. It's been 20 years since "15" was factored on quantum hardware. *IBM*. 2022. online 8.3.2024 - https://www.ibm.com/quantum/blog/factor-15-shors-algorithm.
13. THEORETICAL PHYSICS, Perimeter Institute for. 12-qubits Reached In Quantum Information Quest. *ScienceDaily*. 2006. online 8.3.2024 - www.sciencedaily.com/releases/2006/05/060508164700.htm.
14. KNIGHT, Will. IBM Raises the Bar with a 50-Qubit Quantum Computer. *MIT Technology Review*. 2017. online 8.3.2024 - https://www.technologyreview.com/2017/11/10/147728/ibm-raises-the-bar-with-a-50-qubit-quantum-computer/.
15. CONOVER, Emily. Google moves toward quantum supremacy with 72-qubit computer. *Science News*. 2018. online 8.3.2024 - https://www.sciencenews.org/article/google-moves-toward-quantum-supremacy-72-qubit-computer.

16. GIBNEY, Elizabeth. D-Wave upgrade: How scientists are using the worlds most controversial quantum computer. *Nature*. 2017. online 8.3.2024 - https://www.nature.com/articles/541447b.

17. DAIGLE, Alex. D-Wave Announces 1,200+ Qubit Advantage2 Prototype in New, Lower-Noise Fabrication Stack, Demonstrating 20x Faster Time-to-Solution on Important Class of Hard Optimization Problems. *D-Wave*. 2024. online 8.3.2024 - https://www.dwavesys.com/company/newsroom/press-release/d-wave-announces-1-200-qubit-advantage2-prototype-in-new-lower-noise-fabrication-stack-demonstrating-20x-faster-time-to-solution-on-important-class-of-hard-optimization-problems/.

18. KULHÁNEK, Petr. Kvantový počítač IBM Q. *Aldebaran bulletin*. 2017. online 9.3.2024 - https://www.aldebaran.cz/bulletin/2017_38_ibq.php.

19. TEMPERTON, James. Got a spare $ 15 million? Why not buy your very own D-Wave quantum computer. *Wired*. 2017. online 8.3.2024 - https://www.wired.co.uk/article/d-wave-2000q-quantum-computer.

20. HARRIS, Mark. D-Wave Launches Free Quantum Cloud Service Canadian company joins IBM and Rigetti in offering online access to pricey hardware. *IEEE Spectrum*. 2018. online 8.3.2024 - https://spectrum.ieee.org/dwave-launches-free-quantum-cloud-service.

21. COLLINS, Hugh. IBM Unveils 400 Qubit-Plus Quantum Processor and Next-Generation IBM Quantum System Two. *IBM*. 2022. online 9.3.2024 - https://newsroom.ibm.com/2022-11-09-IBM-Unveils-400-Qubit-Plus-Quantum-Processor-and-Next-Generation-IBM-Quantum-System-Two.

22. GAMBETTA, Jay. The hardware and software for the era of quantum utility is here. *IBM*. 2023. online 9.3.2024 - https://www.ibm.com/quantum/blog/quantum-roadmap-2033.

23. SHAIB, Ali; NAIM, Mohamad Hussein; FOUDA, Mohammed E. Efficient noise mitigation technique for quantum computing. *Nature*. 2023. Dostupné z DOI: https://doi.org/10.1038/s41598-023-30510-5. online 9.3.2024 - https://www.nature.com/articles/s41598-023-30510-5.

24. WUETZ, Brian Paquelet; ESPOSTI, Davide Degli; ZWERVER, Anne-Marije J. Reducing charge noise in quantum dots by using thin silicon quantum wells. *Nature*. 2023. Dostupné z DOI: https://doi.org/10.1038/s41467-

023-37548-z. online 9.3.2024 - https://www.nature.com/articles/s41467-023-37548-z.

25. LIU, G.; CAO, X.; ZHOU, C. Noise Reduction in Qubit Readout with a Two-Mode Squeezed Interferometer. *Physical Review Applied*. 2022. Dostupné z DOI: 10.1103/PhysRevApplied.18.064092. online 9.3.2024 - https://journals.aps.org/prapplied/pdf/10.1103/PhysRevApplied.18.064092.

26. ZEWE, Adam. A technique for making quantum computing more resilient to noise, which boosts performance. *Phys.org*. 2022. online 9.3.2024 - https://phys.org/news/2022-03-technique-quantum-resilient-noise-boosts.html.

27. LASRADO, Avishma. Novel superconducting cavity qubit pushes the limits of quantum coherence. *Physics World*. 2024. online 9.3.2024 - https://physicsworld.com/a/novel-superconducting-cavity-qubit-pushes-the-limits-of-quantum-coherence/.

28. KWON, Sammi. Yale researchers achieve breakthrough in extending qubit's lifetime above break-even point. *The Yale Daily News*. 2023. online 9.3.2024 - https://yaledailynews.com/blog/2023/03/31/yale-researchers-achieve-breakthrough-in-extending-qubits-lifetime-above-break-even-point/.

29. ARGONNE NATIONAL LABORATORY. Quantum Breakthrough: Scientists Extend Qubit Lifetimes. *SCITECHDAILY*. 2024. online 9.3.2024 - https://scitechdaily.com/quantum-breakthrough-scientists-extend-qubit-lifetimes/.

30. DUPLIJ, S.A.; SHAPOVAL, I.I. Quantum computations: fundamentals and algorithms. 2007. online 11.3.2024 - https://arxiv.org/ftp/arxiv/papers/0712/0712.1098.pdf.

31. RAKHADE, Kaustubh. Shor's Algorithm (for Dummies). *Medium*. 2020. online 11.3.2024 - https://kaustubhrakhade.medium.com/shors-factoring-algorithm-94a0796a13b1.

32. MOORE, Tristan. Quantum Computing and Shor's Algorithm. 2016. online 11.3.2024 - https://sites.math.washington.edu/~morrow/336_16/2016papers/tristan.pdf.

33. EKERT, Artur; HOSGOOD, Timothy; KAY, Alastair. *Introduction to Quantum Information Science*. 2024. Dostupné z DOI: 10.1007/978-3-031-50594-2_22. online 12.4.2024 - https://qubit.guide/10.7-simons-algorithm.

34. LI, Jun; PENG, Xinhua; DU, Jiangfeng. An Efficient Exact Quantum Algorithm for the Integer Square-free Decomposition Problem. *Medium*. 2012. online 11.3.2024 - https://www.nature.com/articles/srep00260.

35. JR, Samuel J. Lomonaco. Shor's Quantum Factoring Algorithm. 2000. online 11.3.2024 - https://arxiv.org/pdf/quant-ph/0010034.pdf.

36. AARONSON, Scott. Introduction to Quantum Information Science. 2018. online 11.3.2024 - https://www.scottaaronson.com/qclec.pdf.

37. CHU, Jennifer. The beginning of the end for encryption schemes? *MIT News Office*. 2016. online 9.3.2024 - https://news.mit.edu/2016/quantum-computer-end-encryption-schemes-0303.

38. DRIDI, Raouf; ALGHASSI, Hedayat. Prime factorization using quantum annealing and computational algebraic geometry. *Nature*. 2017. Dostupné z DOI: https://doi.org/10.1038/srep43048. online 11.3.2024 - https://www.nature.com/articles/srep43048.

39. DASH, Avinash; SARMAH, Deepankar. Exact search algorithm to factorize large biprimes and a triprime on IBM quantum computer. *Nature*. 2018. Dostupné z DOI: https://doi.org/10.48550/arXiv.1805.10478. online 11.3.2024 - https://arxiv.org/pdf/1805.10478.pdf.

40. YAN, Bao; TAN, Ziqi; WEI, Shijie. Factoring integers with sublinear resources on a superconducting quantum processor. 2022. online 11.3.2024 - https://arxiv.org/pdf/2212.12372.pdf.

41. WILLSCH, Dennis; WILLSCH, Madita; JIN, Fengping; DE RAEDT, Hans; MICHIELSEN, Kristel. Large-Scale Simulation of Shor's Quantum Factoring Algorithm. *Mathematics*. 2023. Dostupné z DOI: 10.3390/math11194222. online 11.3.2024 - https://www.mdpi.com/2227-7390/11/19/4222.

42. MISHRA, Satyam. Quantum Computing: Grover's Algorithm for Dummies. *Medium*. 2020. online 11.3.2024 - https://tensorslow.medium.com/quantum-computing-grovers-algorithm-for-dummies-5e6a75476815.

43. LOPEZ, Sonia; HUDEK, Ted. Theory of Grover's search algorithm. 2023. online 11.3.2024 - https://learn.microsoft.com/en-us/azure/quantum/concepts-grovers.

44. LEARNING, IBM Quantum. Grover's algorithm. [B.r.]. online 11.3.2024 - https://learning.quantum.ibm.com/course/fundamentals-of-quantum-algorithms/grovers-algorithm.

45. BERNSTEIN, Daniel J.; LANGE, Tanja. Post-quantum cryptography. *Nature*. 2017. Dostupné z DOI: https://doi.org/10.1038/nature23461. online 11.3.2024 - https://www.nature.com/articles/nature23461.

46. MAIRE, Jules; VERGNAUD, Damien. *Efficient Zero-Knowledge Arguments and Digital Signatures via Sharing Conversion in the Head* [Cryptology ePrint Archive, Paper 2024/286]. 2024. Dostupné z DOI: 10.1007/978-3-031-50594-2_22. online 17.4.2024 - https://eprint.iacr.org/2024/286.

47. MCELIECE, Robert J. A public key cryptosystem based on algebraic coding theory. In: 1978. online 15.4.2024 - https://ntrs.nasa.gov/api/citations/19780016269/downloads/19780016269.pdf#page=123.

48. SENDRIER, N. Code-Based Cryptography: State of the Art and Perspectives. IEEE Security and Privacy. *IEEE Security & Privacy*. 2017. Dostupné z DOI: doi:10.1109/msp.2017.3151345. online 11.3.2024 - https://ieeexplore.ieee.org/document/8012331.

49. PEREZ, Ben. A Guide to Post-Quantum Cryptography. *Medium*. 2019. online 11.3.2024 - https://medium.com/hackernoon/a-guide-to-post-quantum-cryptography-d785a70ea04b.

50. LAMPORT, Leslie. *Constructing Digital Signatures from a One Way Function*. 1979. Tech. spr. online 15.4.2024 - https://www.microsoft.com/en-us/research/uploads/prod/2016/12/Constructing-Digital-Signatures-from-a-One-Way-Function.pdf.

51. MERKLE, Ralph C. Secrecy, authentication, and public key systems. In: 1979. online 15.4.2024 - https://archive.org/details/secrecyauthentic0000merk.

52. MCGREW, David; CURCIO, Michael; FLUHRER, Scott. *Leighton-Micali Hash-Based Signatures* [RFC 8554]. 2019. Dostupné z DOI: 10.17487/RFC8554. online 15.4.2024 - https://www.rfc-editor.org/rfc/rfc8554.html.

53. HUELSING, Andreas; BUTIN, Denis; GAZDAG, Stefan-Lukas; RIJNEVELD, Joost; MOHAISEN, Aziz. *XMSS: eXtended Merkle Signature Scheme* [RFC 8391]. 2018. Dostupné z DOI: 10.17487/RFC8391. online 15.4.2024 - https://www.rfc-editor.org/rfc/rfc8391.html.

54. DAM, Duc-Thuan; TRAN, Thai-Ha; HOANG, Van-Phuc. A Survey of Post-Quantum Cryptography: Start of a New Race. *Cryptography*. 2023. Dostupné z DOI: 10.3390/cryptography7030040. online 11.3.2024 - https://www.mdpi.com/2410-387X/7/3/40.

55. NIST. NISTIR 8105 Report on Post-Quantum Cryptography. 2016. Dostupné z DOI: http://dx.doi.org/10.6028/NIST.IR.8105. online 11.3.2024 - https://nvlpubs.nist.gov/nistpubs/ir/2016/NIST.IR.8105.pdf.

56. BERNSTEIN, Daniel J.; HOPWOOD, Daira; HÜLSING, Andreas; LANGE, Tanja; NIEDERHAGEN, Ruben; PAPACHRISTODOULOU, Louiza; SCHNEIDER, Michael; SCHWABE, Peter; WILCOX-O'HEARN, Zooko. SPHINCS: Practical Stateless Hash-Based Signatures. In: *Advances in Cryptology – EUROCRYPT 2015*. Springer Berlin Heidelberg, 2015. ISBN 978-3-662-46800-5. online 15.4.2024 - https://eprint.iacr.org/2014/795.pdf.

57. RICHTER, Maximilian; SEIDENSTICKER, Jasper; BERTRAM, Magdalena. A somewhat gentle introduction to lattice-based post-quantum cryptography. 2023. online 11.3.2024 - https://www.cybersecurity.blog.aisec.fraunhofer.de/en/a-somewhat-gentle-introduction-to-lattice-based-post-quantum-cryptography/.

58. AJTAI, M. Generating hard instances of lattice problems (extended abstract). In: *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*. Association for Computing Machinery, 1996. ISBN 0897917855. Dostupné z DOI: 10.1145/237814.237838. onlne 15.4.2024 - https://dl.acm.org/doi/pdf/10.1145/237814.237838.

59. HOFFSTEIN, Jeffrey; PIPHER, Jill; SILVERMAN, Joseph H. NTRU: A ring-based public key cryptosystem. In: *Algorithmic Number Theory*. Springer Berlin Heidelberg, 1998. ISBN 978-3-540-69113-6. onlne 15.4.2024 - https://www.ntru.org/f/hps98.pdf.

60. AVANZI, Roberto; BOS, Joppe; DUCAS, Léo. *CRYSTALS-Kyber Algorithm Specifications And Supporting Documentation* (*version 3.02*). 2021. Tech. spr. online 12.4.2024 - https://pq-crystals.org/kyber/data/kyber-specification-round3-20210804.pdf.

61. BAI, Shi; DUCAS, Leo; KILTZ, Eike. *CRYSTALS-Dilithium Algorithm Specifications and Supporting Documentation* (*Version 3.1*). 2021. online 12.4.2024 - https://pq-crystals.org/dilithium/data/dilithium-specification-round3-20210208.pdf.

62. NIST. Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process. 2016. online 11.3.2024 - https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf.

63. STANDARDS, National Institute of; TECHNOLOGY. *Digital Signature Standard (DSS)*. 2023. Tech. spr. U.S. Department of Commerce. Dostupné z DOI: https://doi.org/10.6028/NIST.FIPS.186-5. online 15.4.2024 - https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-5.pdf.

64. BARKER, Elaine; CHEN, Lily; ROGINSKY, Allen; VASSILEV, Apostol; DAVIS, Richard; SIMON, Scott. *Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography*. 2018. Tech. spr. U.S. Department of Commerce. Dostupné z DOI: https://doi.org/10.6028/NIST.SP.800-56Ar3. online 15.4.2024 - https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Ar3.pdf.

65. BARKER, Elaine; CHEN, Lily; ROGINSKY, Allen; VASSILEV, Apostol; DAVIS, Richard; SIMON, Scott. *Recommendation for Pair-Wise Key Establishment Using Integer Factorization Cryptography*. 2018. Tech. spr. U.S. Department of Commerce. Dostupné z DOI: https://doi.org/10.6028/NIST.SP.800-56Br2. online 15.4.2024 - https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Br2.pdf.

66. SOSNOWSKI, Markus; WIEDNER, Florian; HAUSER, Eric. The Performance of Post-Quantum TLS 1.3. 2023. online 11.3.2024 - https://www.net.in.tum.de/fileadmin/bibtex/publications/papers/sosnowski2023PQTLS13.pdf.

67. NISTIR 8240. Status Report on the First Round of the NIST Post-Quantum Cryptography Standardization Process. 2019. Dostupné z DOI: https://doi.org/10.6028/NIST.IR.8240. online 11.3.2024 - https://nvlpubs.nist.gov/nistpubs/ir/2019/NIST.IR.8240.pdf.

68. NISTIR 8309. Status Report on the Second Round of the NIST Post-Quantum Cryptography Standardization Process. 2020. Dostupné z DOI: https://doi.org/10.6028/NIST.IR.8309. online 11.3.2024 - https://nvlpubs.nist.gov/nistpubs/ir/2020/NIST.IR.8309.pdf.

69. NIST IR 8413-UPD1. Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process. 2022. Dostupné z DOI: https://doi.org/10.6028/NIST.IR.8413-upd1. online 11.3.2024 -

https://nvlpubs.nist.gov/nistpubs/ir/2022/NIST.IR.8413-upd1.pdf.

70. ARAGON, Nicolas; BARRETO, Paulo S. L. M.; BETTAIEB, Slim; BIDOUX, Loic; BLAZY, Olivier; DENEUVILLE, Jean-Christophe; GABORIT, Philippe; GHOSH, Santosh; GUERON, Shay; MELCHOR, Carlos A.; MISOCZKI, Rafael; PERSICHETTI, Edoardo; RICHTER-BROCKMANN, Jan; SENDRIER, Nicolas; TILLICH, Jean-Pierre; VASSEUR, Valentin; ZEMOR, Gilles. BIKE: Bit Flipping Key Encapsulation. In: 2022. online 15.4.2024 - https://bikesuite.org/files/v5.0/BIKE_Spec.2022.10.10.1.pdf.

71. BERNSTEIN, Daniel J.; CHOU, Tung; CID, Carlos; GILCHER, Jan; LANGE, Tanja; MARAM, Varun; MAURICH, Ingo von; MISOCZKI, Rafael; PERSICHETTI, Edoardo; NIEDERHAGEN, Ruben; SENDRIER, Nicolas; PETERS, Christiane; SZEFER, Jakub; TJHAI, Cen Jung; TOMLINSON, Martin; WANG, Wen. Classic McEliece: conservative code-based cryptography: cryptosystem specification. In: 2022. online 15.4.2024 - https://classic.mceliece.org/mceliece-spec-20221023.pdf.

72. MELCHOR, Carlos Aguilar; ARAGON, Nicolas; BETTAIEB, Slim; BIDOUX, Loic. Hamming Quasi-Cyclic HQC. In: 2024. online 15.4.2024 - https://pqc-hqc.org/doc/hqc-specification_2024-02-23.pdf.

73. COSTELLO, Craig; FEO, Luca De; JAO, David; LONGA, Patrick. Supersingular Isogeny Key Encapsulation. In: 2022. online 15.4.2024 - https://sike.org/files/SIDH-spec.pdf.

74. GOODIN, Dan. Post-quantum encryption contender is taken out by single-core PC and 1 hour. 2022. online 11.3.2024 - https://arstechnica.com/information-technology/2022/08/sike-once-a-post-quantum-encryption-contender-is-koed-in-nist-smackdown/.

75. KIRCHNER, Pierre-Alain Fouque Jeffrey Hoffstein Paul. Falcon: Fast-Fourier Lattice-based Compact Signatures over NTRU. In: 2020. online 15.4.2024 - https://falcon-sign.info/falcon.pdf.

76. BERNSTEIN, Daniel J.; AUMASSON, Jean-Philippe; BEULLENS, Ward; DOBRAUNIG, Christoph; EICHLSEDER, Maria; HÜLSING, Andreas; KAMPANAKIS, Panos; SCHWABE, Peter; LANGE, Tanja. SPHINCS+. In: 2022. online 15.4.2024 - https://sphincs.org/data/sphincs+-r3.1-specification.pdf.

77. WESTERBAAN, Bas. The state of the post-quantum Internet. 2024. online 11.3.2024 - https://blog.cloudflare.com/pq-2024.

78. NIST. Call for Additional Digital Signature Schemes for the Post-Quantum Cryptography Standardization Process. 2022. online 11.3.2024 - https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/call-for-proposals-dig-sig-sept-2022.pdf.

79. BEULLENS, Ward; CHEN, Ming-Shing; DING, Jintai; GONG, Boru; KANNWISCHER, Matthias J.; PATARIN, Jacques; PENG, Bo-Yuan; SCHMIDT, Dieter; SHIH, Cheng-Jhih Shih; TAO, Chengdong; YANG, Bo-Yin. UOV: Unbalanced Oil and Vinegar. In: 2023. online 15.4.2024 - https://drive.google.com/file/d/1NdMHuCyyFG6xgQGrpssM99kyiNwA9JG-/view.

80. BEULLENS, Ward; CAMPOS, Fabio; CELI, Sofia; HESS, Basil; KANNWISCHER, Matthias J. MAYO. In: 2023. online 15.4.2024 - https://pqmayo.org/assets/specs/mayo.pdf.

81. STANDARDS, National Institute of; TECHNOLOGY. *Module-Lattice-Based Key-Encapsulation Mechanism Standard*. 2023. Tech. spr. U.S. Department of Commerce. Dostupné z DOI: https://doi.org/10.6028/NIST.FIPS.203.ipd. online 11.3.2024 - https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.203.ipd.pdf.

82. NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. *Module-Lattice-Based Digital Signature Standard*. 2023. Tech. spr. U.S. Department of Commerce. Dostupné z DOI: https://doi.org/10.6028/NIST.FIPS.204.ipd. online 11.3.2024 - https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.204.ipd.pdf.

83. STANDARDS, National Institute of; TECHNOLOGY. *Stateless Hash-Based Digital Signature Standard*. 2023. Tech. spr. U.S. Department of Commerce. Dostupné z DOI: https://doi.org/10.6028/NIST.FIPS.205.ipd. online 11.3.2024 - https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.205.ipd.pdf.

84. NIST. Comments Requested on Three Draft FIPS for Post-Quantum Cryptography. 2023. online 11.3.2024 - https://csrc.nist.gov/news/2023/three-draft-fips-for-post-quantum-cryptography.

85. MOODY, Dustin. Public Comments posted for draft FIPS 203, 204, and 205. 2023. online 11.3.2024 - https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/Tc5XMkyn8aM.

86. NIST. NIST to Standardize Encryption Algorithms That Can Resist Attack by Quantum Computers. 2023. online 11.3.2024 - https://www.nist.gov/news-events/news/2023/08/nist-standardize-encryption-algorithms-can-resist-attack-quantum-computers.

87. TASOPOULOS, George; LI, Jinhui; FOURNARIS, Apostolos P. Performance Evaluation of Post-Quantum TLS 1.3 on Resource-Constrained Embedded Systems. 2022. Dostupné z DOI: https://dx.doi.org/10.14722/ndss.2020.24203. online 11.3.2024 - https://eprint.iacr.org/2021/1553.pdf.

88. SIKERIDIS, Dimitrios; KAMPANAKIS, Panos; DEVETSIKIOTIS, Michael. Post-Quantum Authentication in TLS 1.3: A Performance Study. 2020. Dostupné z DOI: https://dx.doi.org/10.14722/ndss.2020.24203. online 11.3.2024 - https://eprint.iacr.org/2020/071.pdf.

89. SCOTT, Michael. On TLS for the Internet of Things, in a Post Quantum world. 2023. online 11.3.2024 - https://eprint.iacr.org/2023/095.pdf.

90. DOUGLAS STEBILA Scott Fluhrer, Shay Gueron. *Hybrid key exchange in TLS 1.3*. Internet Engineering Task Force, 2024. Internet-Draft, draft-ietf-tls-hybrid-design-10. Dostupné tiež z: https://datatracker.ietf.org/doc/draft-ietf-tls-hybrid-design/10/. online 12.4.2024 - https://www.ietf.org/archive/id/draft-ietf-tls-hybrid-design-10.html.

91. BRAINARD, John; KALISKI, Burt; TURNER, Sean; RANDALL, James. *Use of the RSA-KEM Key Transport Algorithm in the Cryptographic Message Syntax (CMS)* [RFC 5990]. 2010. Dostupné z DOI: 10.17487/RFC5990. online 15.3.2024 - https://www.rfc-editor.org/info/rfc5990.

92. TAMVADA, Goutam; CELI, Sofía. Deep dive into a post-quantum key encapsulation algorithm. 2022. online 11.3.2024 - https://blog.cloudflare.com/post-quantum-key-encapsulation/.

93. RESCORLA, Eric. *The Transport Layer Security (TLS) Protocol Version 1.3* [RFC 8446]. 2018. Dostupné z DOI: 10.17487/RFC8446. online 11.3.2024 - https://www.rfc-editor.org/info/rfc8446.

94. LANGLEY, Adam; HAMBURG, Mike; TURNER, Sean. *Elliptic Curves for Security* [RFC 7748]. 2016. Č. 7748. Dostupné z DOI: 10.17487/RFC7748. online 11.3.2024 - https://www.rfc-editor.org/rfc/rfc7748.

95. *Open Quantum Safe project*. [B.r.]. online 12.4.2024 - hhttps://openquantumsafe.org/.

96. STEBILA, Douglas; MOSCA, Michele. *Post-quantum key exchange for the Internet and the Open Quantum Safe project*. 2016. online 12.4.2024 - `https://eprint.iacr.org/2016/1017.pdf`.

97. GE, Jiangxia; SHAN, Tianshu; XUE, Rui. *On the Fujisaki-Okamoto transform: from Classical CCA Security to Quantum CCA Security* [Cryptology ePrint Archive, Paper 2023/792]. 2023. Dostupné tiež z: `https://eprint.iacr.org/2023/792`. online 11.3.2024 - `https://eprint.iacr.org/2023/792.pdf`.

98. GONZALEZ, Ruben. Kyber - How does it work? 2021. online 25.3.2024 - `https://cryptopedia.dev/posts/kyber/`.

99. YUE, Steven; HOWE, James. Adventures in PQC: Exploring Kyber in Python - Part I. 2022. online 25.3.2024 - `https://cryptographycaffe.sandboxaq.com/posts/kyber-01/`.

100. LI, Jianwei; NGUYEN, Phong Q. *A Complete Analysis of the BKZ Lattice Reduction Algorithm* [Cryptology ePrint Archive, Paper 2020/1237]. 2020. online 14.4.2024 - `https://eprint.iacr.org/2020/1237.pdf`.

101. WU, Han; WANG, Xiaoyun; XU, Guangwu. *Reducing an LWE Instance by Modular Hints and its Applications to Primal Attack, Dual Attack and BKW Attack* [Cryptology ePrint Archive, Paper 2022/1404]. 2022. online 25.3.2024 - `https://eprint.iacr.org/2022/1404`.

102. LAARHOVEN, Thijs; WALTER, Michael. *Dual lattice attacks for closest vector problems (with preprocessing)* [Cryptology ePrint Archive, Paper 2021/557]. 2021. online 25.3.2024 - `https://eprint.iacr.org/2021/557`.

103. BI, Lei; LU, Xianhui; LUO, Junjie. *Hybrid Dual Attack on LWE with Arbitrary Secrets* [Cryptology ePrint Archive, Paper 2021/152]. 2021. online 25.3.2024 - `https://eprint.iacr.org/2021/152`.

104. POSTLETHWAITE, Eamonn W.; VIRDIA, Fernando. *On the Success Probability of Solving Unique SVP via BKZ*. 8. vyd. Springer International Publishing, 2021. ISBN 9780130676108. online 25.3.2024 - `https://link.springer.com/chapter/10.1007/978-3-030-75245-3_4`.

105. LI, Jianwei; NGUYEN, Phong Q. *A Complete Analysis of the BKZ Lattice Reduction Algorithm* [Cryptology ePrint Archive, Paper 2020/1237]. 2020. online 25.3.2024 - `https://eprint.iacr.org/2020/1237`.

106. ROY, Kumar; DARSHAN, Shiva. Analyzing CRYSTALS-Kyber's Susceptibility to Side Channel Attacks: An Empirical Exploration. 2024. Dostupné z DOI: 10.21203/rs.3.rs-4015385/v1. online 25.3.2024 - https://www.researchgate.net/publication/378797241_Analyzing_CRYSTALS-Kyber's_Susceptibility_to_Side_Channel_Attacks_An_Empirical_Exploration.

107. TOULAS, Bill. KyberSlash attacks put quantum encryption projects at risk. 2024. online 25.3.2024 - https://www.bleepingcomputer.com/news/security/kyberslash-attacks-put-quantum-encryption-projects-at-risk/.

108. JOHNSON, Derek. Post-quantum algorithm vulnerable to side channel attacks. 2023. online 25.3.2024 - https://www.scmagazine.com/analysis/post-quantum-algorithm-attack.

109. KARAGIANNIS, Konstantinos. No, Post-Quantum Cryptography Finalist CRYSTALS-Kyber Wasn't Hacked. 2023. online 25.3.2024 - https://tcblog.protiviti.com/2023/03/23/no-post-quantum-cryptography-finalist-crystals-kyber-wasnt-hacked/.

110. WESTERBAAN, Bas. No, AI did not break post-quantum cryptography. 2023. online 25.3.2024 - https://blog.cloudflare.com/kyber-isnt-broken/?trk=article-ssr-frontend-pulse_little-text-block.

111. DUBROVA, Elena; NGO, Kalle; GÄRTNER, Joel. *Breaking a Fifth-Order Masked Implementation of CRYSTALS-Kyber by Copy-Paste* [Cryptology ePrint Archive, Paper 2022/1713]. 2022. online 25.3.2024 - https://eprint.iacr.org/2022/1713.

112. PRIMAS, Robert; PESSL, Peter; MANGARD, Stefan. Single-Trace Side-Channel Attacks on Masked Lattice-Based Encryption. In: Springer, 2017. Dostupné z DOI: 10.1007/978-3-319-66787-4_25. online 25.3.2024 - https://eprint.iacr.org/2017/594.pdf.

113. RICHTER, Maximilian; BERTRAM, Magdalena; SEIDENSTICKER, Jasper; TSCHACHE, Alexander. A Mathematical Perspective on Post-Quantum Cryptography. *Mathematics*. 2022. Dostupné z DOI: 10.3390/math10152579. online 25.3.2024 - https://www.mdpi.com/2227-7390/10/15/2579.

114. WANG, Geng; XIA, Wenwen; SHI, Gongyu; WAN, Ming; ZHANG, Yuncong; GU, Dawu. *Revisiting the Concrete Hardness of SelfTargetMSIS in CRYSTALS-*

Dilithium [Cryptology ePrint Archive, Paper 2022/1601]. 2022. online 14.4.2024 - https://eprint.iacr.org/2022/1601.pdf.

115. JENDRAL, Sönke. *A Single Trace Fault Injection Attack on Hedged CRYSTALS-Dilithium* [Cryptology ePrint Archive, Paper 2024/238]. 2024. online 12.4.2024 - https://eprint.iacr.org/2024/238.

116. MARZOUGUI, Soundes; ULITZSCH, Vincent; TIBOUCHI, Mehdi; SEIFERT, Jean-Pierre. *Profiling Side-Channel Attacks on Dilithium: A Small Bit-Fiddling Leak Breaks It All* [Cryptology ePrint Archive, Paper 2022/106]. 2022. online 12.4.2024 - https://eprint.iacr.org/2022/106.

117. WANG, Ruize; NGO, Kalle; GÄRTNER, Joel; DUBROVA, Elena. *Single-Trace Side-Channel Attacks on CRYSTALS-Dilithium: Myth or Reality?* [Cryptology ePrint Archive, Paper 2023/1931]. 2023. online 12.4.2024 - https://eprint.iacr.org/2023/1931.

118. ISLAM, Saad; MUS, Koksal; SINGH, Richa. *Signature Correction Attack on Dilithium Signature Scheme*. 2022. Dostupné z DOI: https://doi.org/10.48550/arXiv.2203.00637. online 12.4.2024 - https://arxiv.org/pdf/2203.00637.pdf.

119. BRONCHAIN, Olivier; AZOUAOUI, Melissa; ELGHAMRAWY, Mohamed; RENES, Joost; SCHNEIDER, Tobias. Exploiting Small-Norm Polynomial Multiplication with Physical Attacks: Application to CRYSTALS-Dilithium. *IACR Transactions on Cryptographic Hardware and Embedded Systems*. 2024, roč. 2024. Dostupné z DOI: 10.46586/tches.v2024.i2.359–383. online 12.4.2024 - https://tches.iacr.org/index.php/TCHES/article/view/11432/10937.

120. KRAHMER, Elisabeth; PESSL, Peter; LAND, Georg; GÜNEYSU, Tim. *Correction Fault Attacks on Randomized CRYSTALS-Dilithium* [Cryptology ePrint Archive, Paper 2024/138]. 2024. online 12.4.2024 - https://eprint.iacr.org/2024/138.

121. BERNSTEIN, Daniel J.; BRUMLEY, Billy Bob; CHEN, Ming-Shing; CHUENGSATIANSUP, Chitchanok; LANGE, Tanja; MAROTZKE, Adrian; PENG, Bo-Yuan; TUVERI, Nicola; VREDENDAAL, Christine van; YANG, Bo-Yin. NTRU Prime. In: 2020. online 15.4.2024 - https://ntruprime.cr.yp.to/nist/ntruprime-20201007.pdf.

122. ALKIM, Erdem; CAMPBOSOS, Joppe W.; DUCAS, Leo; LONGA, Patrick; MIRONOV, Ilya; NAEHRIG, Michael; NIKOLAENKO, Valeria; PEIKERT, Chris; RAGHUNATHAN, Ananth; STEBILA, Douglas. FrodoKEM: Learning With Errors Key Encapsulation. In: 2023. online 15.4.2024 - `https://frodokem.org/files/FrodoKEM-standard_proposal-20230314.pdf`.

123. BELL, Charlie. *Building a quantum-safe future*. 2023. online 12.4.2024 - `https://blogs.microsoft.com/blog/2023/05/31/building-a-quantum-safe-future/`.

124. VENABLES, Phil. *Building a quantum-safe future*. 2022. online 12.4.2024 - `https://cloud.google.com/blog/products/identity-security/how-google-is-preparing-for-a-post-quantum-world`.

125. AMAZON AWS. *Using hybrid post-quantum TLS with AWS KMS*. 2023. online 12.4.2024 - `https://docs.aws.amazon.com/kms/latest/developerguide/pqtls.html`.

126. IBM CLOUD. *Introduction to Quantum-safe Cryptography in TLS*. 2022. online 12.4.2024 - `https://cloud.ibm.com/docs/key-protect?topic=key-protect-quantum-safe-cryptography-tls-introduction`.

Zoznam príloh

A Obsah CD Média

Obsah tiež dostupný online na GITe:

`https://git.kemt.fei.tuke.sk/js331zc/MastersThesis`

- `./MastersThesis/` ..... Koreňový priečinok
  - `README.md`
  - `appendixes /` ..... Priečinok s prílohami
    - `falcon512_CA.crt` ..... Vzorový PQC certifikát
    - `gen_test_cert_ubuntu.sh` .... Skript na generovanie PQC certifikátov používaný pri testovaní
    - `info_falcon512_CA.txt` ..... Informácie získané zo vzorového PQC certifikátu
    - `Postup_instalacie_kniznice_OQS-OpenSSL_1.1.1f.md` .. Návod na inštaláciu knižnice OQS-OpenSSL
    - `Postup_instalacie_kniznic_OpenSSL_liboqs_oqs-provider.md` Návod na inštaláciu knižníc OpenSSL, liboqs, oqs-provider
    - `Postup_instalacie_TiigerTLS-original.md` .... Návod na inštaláciu originálnej verzie knižnice TiigerTLS
    - `Postup_kompilacie_projektu_PQ_PROJECT_SSL_TLS.txt` .. Návod na kompiláciu projektu PQ_PROJECT_SSL_TLS
    - `Postup_kompilacie_projektu_PQ_TiigerTLS.txt` ..... Návod na kompiláciu upravenej knižnice PQ_TiigerTLS
    - `Test_pripojenia_OQS_server.txt` ..... Výsledky experimentálnych pripojení k test serveru OQS
    - `Vysledky_merania_casov_KEX_KEM_gen_klucov.txt` Výsledky meraní času pri generovaní KEX/KEM kľúčov
    - `Vysledky_merania_casov_overovania_certifikatov.txt` . Výsledky meraní času pri overovaní PQC certifikátov
  - `CertifikacnaAutorita/` ... Priečinok so vzorovou certifikačnou autoritou
    - `gen_CA.bat` ..... Skript na generovanie vzorovej CA a PQC certifikátov
    - `gen_client_certificate.bat` ..... Skript na generovanie vzorového certifikátu pre klienta
    - `intermediate.cnf` ..... Konfiguračný súbor pre sekundárnu CA
    - `README.md`
    - `root.cnf` ..... Konfiguračný súbor pre koreňovú CA
  - `PQ_PROJECT_SSL_TLS/` ..... Priečinok s PQC aplikáciami klient-server pre účely výučby

- dokumentacia.pdf
- program_structure.svg
- readme
- CERIFICATEs/ ..... Priečinok venovaný generovaniu certifikátov pre aplikácie klienta a serveru
  - ECC/ ..... Priečinok so skriptami pre generovanie ECC certifikátov
    - certificate-authority-options.conf Konfiguračný súbor pre CA
    - client.ext .... Konfiguračné rozšírenia pre klientské certifikáty
    - gen_cert_ECC.bat ..... Skript na generovanie ECC certifikátov
    - options.conf ..... Doplnkové nastavenia certifikátov
    - server.ext ..... Konfiguračné rozšírenia pre certifikáty servera
  - PQ/ ..... Priečinok so skriptami pre generovanie PQC certifikátov
    - certificate-authority-options.conf Konfiguračný súbor pre CA
    - client.ext .... Konfiguračné rozšírenia pre klientské certifikáty
    - gen_PQ_cert.bat ..... Skript na generovanie PQC certifikátov
    - options.conf ..... Doplnkové nastavenia certifikátov
    - README.txt
    - server.ext ..... Konfiguračné rozšírenia pre certifikáty servera
  - RSA/..... Priečinok so skriptami pre generovanie RSA certifikátov
    - certificate-authority-options.conf Konfiguračný súbor pre CA
    - client.ext .... Konfiguračné rozšírenia pre klientské certifikáty
    - gen_cert_RSA.bat ..... Skript na generovanie RSA certifikátov
    - options.conf ..... Doplnkové nastavenia certifikátov
    - server.ext ..... Konfiguračné rozšírenia pre certifikáty servera
- CLIENT_SERVER_SECURE/ ..... Priečinok pre klient-server aplikácie nezávislé od použitého OS založené na podmienenej kompilácii
  - comp_client.bat ..... Batch skript na kompiláciu serveru
  - comp_server.bat ..... Batch skript na kompiláciu klienta
  - makefile..... MakeFile na automatizovanú kompiláciu aplikácii
  - myCA.pem ..... Koreňový certifikát
  - schematic.svg
  - start_client.bat ..... Batch skript na spustenie klienta
  - start_server.bat ..... Batch skript na kompiláciu serveru
  - CLIENT/ . Priečinok so zdrojovým kódom, certifikátom a privátnym kľúčom klienta
    - client.c ..... Zdrojový kód klienta
    - client.key ..... Súkromný kľúč klienta
    - client.pem ..... Certifikát klienta
  - SERVER/ . Priečinok so zdrojovým kódom, certifikátom a privátnym kľúčom servera
    - server.c ..... Zdrojový kód servera
    - server.key ..... Súkromný kľúč servera
    - server.pem ..... Certifikát servera
- CLIENT_SERVER_SECURE_BIO/ .... Priečinok pre klient-server aplikácie nezávislé od použitého OS založené BIO štruktúrach

- `comp_client.bat` ..... Batch skript na kompiláciu serveru
- `comp_server.bat` ..... Batch skript na kompiláciu klienta
- `makefile` ..... MakeFile na automatizovanú kompiláciu aplikácii
- `myCA.pem` ..... Koreňový certifikát
- `schematic.svg`
- `start_client.bat` ..... Batch skript na spustenie klienta
- `start_server.bat` ..... Batch skript na kompiláciu serveru
- `CLIENT` ... Priečinok so zdrojovým kódom, certifikátom a privátnym kľúčom klienta.
  - `client.c` ..... Zdrojový kód klienta
  - `client.key` ..... Súkromný kľúč klienta
  - `client.pem` ..... Certifikát klienta
- `SERVER/` . Priečinok so zdrojovým kódom, certifikátom a privátnym kľúčom servera
  - `server.c` ..... Zdrojový kód servera
  - `server.key` ..... Súkromný kľúč servera
  - `server.pem` ..... Certifikát servera
- `oqsprovider/` ..... Priečinok s knižnicou oqsprovider
  - `openssl.cnf` Vzorový konfiguračný súbor OpenSSL s aktivovaným oqs-providerom
  - `oqsprovider.dll` .. Vygenerovaná knižnica oqs-providera pre účely výučby na x86 Win7
  - `README.txt` ..... Stručný návod na použitie oqs-providera
- `PQ_TIIGER_TLS/` ..... Priečinok s TLS PQC klient na základe knižnice TiigerTLS
  - `CHANGES.md` ..... Prehľad vykonaných zmien v celom projekte
  - `README.txt`
  - `README_SCOTT.md`
  - `doc/` ..... Priečinok s dokumentáciou pôvodnej knižnice TiigerTLS
    - `Doxyfile`
    - `list.txt`
    - `refman.pdf`
    - `sal.pdf`
    - `tls.pdf`
    - `vision.pdf`
  - `include/` ..... Priečinok s hlavičkovými súbormi
    - `tls1_3.h`
    - `tls_bfibe.h`
    - `tls_certs.h`
    - `tls_cert_chain.h`
    - `tls_client_recv.h`
    - `tls_client_send.h`
    - `tls_keys_calc.h`
    - `tls_logger.h`
    - `tls_octads.h`
    - `tls_pqibe.h`
    - `tls_protocol.h`
    - `tls_sal.h`

- tls_sockets.h
- tls_tickets.h
- tls_x509.h

lib/ ..... Priečinok so zdrojovými kódmi aplikácie

- tls_cacerts.cpp
- tls_cert_chain.cpp
- tls_client_cert.cpp
- tls_client_recv.cpp
- tls_client_send.cpp
- tls_keys_calc.cpp
- tls_logger.cpp
- tls_octads.cpp
- tls_protocol.cpp
- tls_sal.cpp
- tls_sockets.cpp
- tls_tickets.cpp
- tls_x509.cpp
- ibe
  - tls_bfibe.cpp
  - tls_pqibe.cpp

liboqs/ ..... Priečinok s rôznymi verziami knižnice liboqs

- ubuntu22-18-04-24/ Priečinok knižnicou liboqs pre Linux Ubuntu 22.04
  - include/
    - oqs/
  - lib/
    - liboqs.a
- winx64-14-04-24/ .. Priečinok s knižnicou liboqs pre OS Windows x64
  - include/
    - oqs/
  - lib/
    - liboqs-internal.a
    - liboqs.a
- winx86-11-04-24/ .. Priečinok s knižnicou liboqs pre OS Windows x86
  - include/
    - oqs/
  - lib/
    - liboqs-internal.a
    - liboqs.a

sal/ ..... Priečinok knižnicou s liboqs pre OS Windows x64

- README.txt
- sal.pdf
- tls_sal_m.xpp ..... Zdrojový kód bezpečnostnej vrstvy aplikácie
- miracl-ubuntu22-11-04-24... Priečinok s knižnicou MIRACL core pre Linux Ubuntu 22.04
  - core.a

- includes/
- `miracl-winx64-15-04-24` . Priečinok s knižnicou MIRACL core pre OS Windows x64
  - `core.a`
  - `includes/`
- `miracl-winx86-11-04-24` . Priečinok s knižnicou MIRACL core pre OS Windows x86
  - `core.a`
  - `includes/`
- `src/`
  - `client.cpp` ..... Zdrojový kód klienta

B Postup inštalácie knižnice OQS-OpenSSL 1.1.1f

LINUX

1. Inštalácia potrebných nástrojov

```
sudo apt install cmake gcc libtool libssl-dev make ninja-build git -y
```

2. Stiahnutie a inštalácia liboqs

```
git clone --branch main https://github.com/open-quantum-safe/liboqs.
    git
cd liboqs
mkdir build && cd build
cmake -GNinja -DCMAKE_INSTALL_PREFIX=../../openssl/oqs ..
ninja
ninja install
```

3. Stiahnutie a inštalácia OQS-OpenSSL

```
git clone --branch OQS-OpenSSL_1_1_1-stable https://github.com/open-
    quantum-safe/openssl.git
cd ./openssl
./Configure no-shared linux-x86_64 -lm
make -j 1
sudo make install
```

WINDOWS

1. Príprava prostredia

GCC, ninja, CMake - súťasťou knižnice Winlibs - `https://winlibs.com/`
Perl - stiahnutie a inštalácia Perl-u strawberry-perl-5.32.1.1-64bit zo stránky `http`

s://strawberryperl.com/

NMAKE - stiahnutie a inštalácia nástrojov Build Tools for Visual Studio 2022 zo stránky https://visualstudio.microsoft.com/downloads/

Pridanie premenných prostredia

```
D:\Strawberry\perl\bin
D:\Program Files\CMake\bin
D:\Program Files (x86)\Microsoft Visual Studio\2022\BuildTools\VC\
    Tools\MSVC\14.35.32215\bin\Hostx86\x86\nmake.exe
```

V našom prípade systémová premenná nepomohla a bolo nutné ich nekonfigurovať manuálne v Powershell termináli príkazmi:

```
$env:path += ";D:\Program Files\cmake\bin"
$env:path += ";d:\Program Files (x86)\Microsoft Visual Studio\2022\
    BuildTools\VC\Tools\MSVC\14.35.32215\bin\Hostx86\x86\"
```

Pre správnu funkciu NMAKE je nutné spustiť súbor *vcvarsall.bat*
V našom prípade sa tento súbor nachádzal na adrese

```
d:\Program Files (x86)\Microsoft Visual Studio\2022\BuildTools\VC\
    Auxiliary\Build\
```

2. Stiahnutie OQS-OpenSSL a liboqs

OpenSSL OQS - stiahnutie OQS OpenSSL repozitáru z GITu - https://github.com/open-quantum-safe/openssl/tree/OQS-OpenSSL_1_1_1-stable
Liboqs - stiahnutie súborov z GITu - https://github.com/open-quantum-safe/liboqs

3. Príprava liboqs

```
cd liboqs
mkdir build
cd build
cmake -GNinja -DCMAKE_INSTALL_PREFIX='D:\Program Files\openssl\openssl
    -OQS-OpenSSL_1_1_1-stable\oqs' ..
ninja
ninja install
```

4. Inštalácia OQS OpenSSL

```
cd ./openssl
perl Configure VC-WIN64A no-shared
nmake
```

C Postup inštalácie knižníc OpenSSL, liboqs a oqs-provider

LINUX

Inštalácia a konfigurácia OpenSSL 3.3

```
sudo apt install build-essential checkinstall -y
wget https://github.com/openssl/openssl/releases/download/openssl
    -3.3.0/openssl-3.3.0.tar.gz
tar -xvf openssl-3.3.0.tar.gz
cd openssl-3.3.0
./Configure
make
sudo make install

export PATH="/usr/local:$PATH"
export LD_LIBRARY_PATH="/usr/local/lib64:$PATH"

cd /etc/ld.so.conf.d/
sudo touch openssl-3.3.0.conf
echo "/usr/local/lib64" | sudo tee -a openssl-3.3.0.conf
sudo ldconfig -v

sudo apt install --reinstall ca-certificates
sudo update-ca-certificates -f
nano ~/.wgetrc
ca_certificate=/etc/ssl/certs/ca-certificates.crt
```

liboqs

```
sudo apt install astyle cmake ninja-build libssl-dev python3-pytest
    python3-pytest-xdist unzip xsltproc doxygen graphviz python3-yaml
    valgrind -y
```

```
wget https://github.com/open-quantum-safe/liboqs/archive/refs/tags
    /0.10.0.tar.gz
tar -xvf 0.10.0.tar.gz
cd cd liboqs-0.10.0
mkdir build && cd build

cmake -GNinja .. -DOPENSSL_ROOT_DIR=/usr/local/lib64 -
    DOQS_ALGS_ENABLED=ALL -DBUILD_SHARED_LIBS=ON
ninja
sudo ninja install

cd /etc/ld.so.conf.d/
sudo touch liboqs.conf
echo "/usr/local/lib" | sudo tee -a liboqs.conf
sudo ldconfig -v
```

oqs-provider

```
wget https://github.com/open-quantum-safe/oqs-provider/archive/refs/
    tags/0.6.0.tar.gz
tar -xvf 0.6.0.tar.gz

cd oqs-provider-0.6.0
cmake -S . -B _build -DOPENSSL_ROOT_DIR=/usr/local/lib64 -Dliboqs_DIR
    =/usr/local -DBUILD_SHARED_LIBS=ON
cmake --build _build
sudo cmake --install _build
```

Po inštalácii pridáme do konfiguračného súboru **openssl.cnf** modul, ktorý permanentne aktivuje providera a jeho funkcie. Aktivujeme tiež „default“ providera.

```
cd /usr/local/ssl/
sudo nano openssl.cnf

[provider_sect]
default = default_sect
oqsprovider = oqsprovider_sect

[default_sect]
activate = 1


[oqsprovider_sect]
activate = 1
```

WINDOWS

WinLibs

- zo stránky `https://winlibs.com` stiahneme .zip archív, rozbalíme ho a skopírujeme celú zložku na nami určené miesto
- prejdeme do *Settings-System-Advanced system settings-Environment Variables-System variables-Path-Edit* a pridáme cestu k priečinku s prekladačom, napríklad: *C:/mingw64/bin/*
- nastaviť cestu k prekladaču môžeme aj úpravou premennej v príkazovom riadku príkazom *SET PATH=C:/mingw64/bin/;%PATH%*
- v priečinku */bin/* sa nachádza aplikácia **mingw32-make.exe**, z ktorej si urobíme kópiu a premenujeme ju na **make.exe**

CMake

Ak z nejakého dôvodu nemôžeme použiť cmake priamo z WinLibs knižnice, tak ho môžeme stiahnuť z oficiálnej stránky - `https://cmake.org/download/`

OpenSSL (FireDaemon OpenSSL)

- v našom prípade sme využívali predkompilovanú verziu OpenSSL zo stránky `https://www.firedaemon.com/download-firedaemon-openssl`
- stiahneme .zip archív, rozbalíme ho a všetky priečinky z priečinku */x64/* spolu s priečinkom */ssl/* skopírujeme na nami vybrané miesto
- z priečinku */bin/* skopírujeme súbory *libssl.lib* a *libcrypto.lib* do */lib/* priečinku nášho prekladača, v našom prípade teda do *C:/mingw64/lib/*
- nastavíme cestu k aplikácii OpenSSL a konfiguračnému súboru prostredníctvom konzolových príkazov:

```
set OPENSSL_CONF=C:\openssl\ssl\openssl.cnf
set PATH=C:\openssl\bin;%PATH%
```

LIBOQS

- stiahneme .zip súbor knižnice liboqs z oficiálneho GITu – `https://github.com/open-quantum-safe/liboqs`

- stiahnutý súbor rozbalíme na nami zvolené miesto
- následne knižnicu nainštalujeme týmito príkazmi:

```
cd liboqs-main
mkdir build
cd build
cmake -GNinja .. -DOQS_ALGS_ENABLED=ALL
ninja
ninja install
```

Po dokončení inštalácie skopírujeme vygenerované priečinky *include* a *lib* do zložky s naším gcc prekladačom, teda do *C:/mingw64/*
Poznámka: podľa nastavení systému môžu byť potrebné na vykonanie príkazu 'ninja install' administrátorské práva

OQS-PROVIDER

- stiahneme .zip súbor knižnice oqs-provider z oficiálneho GITu – `https://github.com/open-quantum-safe/oqs-provider`
- stiahnutý súbor rozbalíme na nami vybrané miesto
- v súbore *CMakeLists.txt* pridáme na riadok 90 a 91 pred funkcie ***enable_testing***() a ***add_subdirectory***(***test***) symbol „#", ktorým zakomentujeme a nezrealizujeme kompiláciu testov - pri kompilácii testu *oqs_test_tlssig.c* by nám vyskočila chyba, ktorá by spôsobila, že by celá kompilácia knižnice spadla

```
cmake -GNinja -DOPENSSL_ROOT_DIR=C:\openssl -S . -B _build
cd _build
ninja
ninja install
```

Po inštalácii pridáme do konfiguračného súboru **C:/OpenSSL/ssl/openssl.cnf** modul, ktorý permanentne aktivuje providera a jeho funkcie. Aktivujeme tiež „default" providera.

```
[provider_sect]
default = default_sect
oqsprovider = oqsprovider_sect

[default_sect]
activate = 1
```

```
[oqsprovider_sect]
activate = 1
```

D Postup kompilácie projektu PQ_-PROJECT_SSL_TLS pre výučbu

Tento postup kompilácia predpokladá, že používateľ už úspešne nainštaloval a aktivoval knižnicu **oqs-provider**.

oqs-provider

Súčasťou projektu je už skompilovaná knižnice oqs-provider 0.6.0 s využitím knižnice liboqs 0.10.0 na 32-bitovom systéme Windows 7, ktorý sa používa pri výučbe predmetov BPS a BIKS. Zároveň:

- predpokladáme, že systém obsahuje knižnice libcrypto a libssl
- na generovanie PQ certifikátov je potrebné mať OpenSSL.exe

Súbor *oqsprovider.dll* presunieme na náš virtuálny počítač na ľubovoľné miesto. Napríklad ho skopirujeme priamo na plochu, teda do *C:/Users/Administrator/Desktop* Cestu k súboru potom definujeme ako systémovú premennú **OPENSSL_MODULES** cez:

- PowerShell príkaz

```
[Environment]::SetEnvironmentVariable("OPENSSL_MODULES", "C:\
    Users\Administrator\Desktop\", "Machine")
```

- cmd príkaz

```
setx OPENSSL_MODULES "C:\Users\Administrator\Desktop\"
```

- manuálne cez nastavenia

```
Settings–System–Advanced system settings–Environment
    Variables–System variables–New
Name: OPENSSL_MODULES
Value: C:\Users\Administrator\Desktop\
```

LINUX

- stiahneme celý balík PQ_PROJECT_SSL_TLS a prejdeme napríklad do zložky *CLIENT_SERVER_SECURE*
- kompiláciu realizujeme príkazmi:

```
gcc -Wall -Wextra -o server_run ./SERVER/server.c -
    lcrypto -lssl
gcc -Wall -Wextra -o client_run ./CLIENT/client.c -
    lcrypto -lssl
```

- aplikácie spustíme príkazmi:

```
./server_run 5000 ./SERVER/server.pem ./SERVER/server.key
./client_run 127.0.0.1 5000 ./CLIENT/client.key ./CLIENT/
    client.pem
```

WINDOWS

- stiahneme celý balík PQ_PROJECT_SSL_TLS a prejdeme napríklad do zložky *CLIENT_SERVER_SECURE*
- kompiláciu spustíme priloženými skriptami *comp_server.bat* a *comp_client.bat*
- aplikácie spustíme skriptami *start_client.bat*, resp. *start_server.bat*

Certifikáty s podporou PQC

K zdrojovým kódom aplikácii sme pridali nami vygenerované testovacie certifikáty. Používateľ si môže vygenerovať vlastné certifikáty pomocou skriptu *gen_PQ_cert.bat*, ktorý sa nachádza v zložke *PQ_PROJECT_SSL_TLS/CERIFICATEs/PQ/*.
Staré *.pem* a *.key* súbory potom stačí nahradiť novo-vygenerovanými súbormi.

E Postup inštalácie knižnice (originálnej verzie) TiigerTLS

LINUX

Server

1. Inštalácia kompilačných nástrojov jazyku Rust

```
curl https://sh.rustup.rs -sSf | sh
```

2. Vygenerovanie súborov knižnice MIRACL core

```
sudo apt install python3
git clone https://github.com/miracl/core.git
cd core/rust
python3 config64.py test
cd ../..
```

3. Príprava Rust servera

```
git clone https://github.com/Crypto-TII/TLS1.3.git
cd TLS1.3/rust/server
nano Cargo.toml
```

V súbore *Cargo.toml* je nutné správne nalinkovať cestu k zložke *textbfmcore*, ktorá vznikla pri generovaní MIRACL core knižnice v predošlom kroku.

4. Kompilácia Rust servera

```
cargo build
cargo run
```

Klient

1. Vygenerovanie súborov knižnice MIRACL core

```
sudo apt install python3
git clone https://github.com/miracl/core.git
cd core/cpp
python3 config64.py test
cd ../..
```

2. Inštalácia knižnice libsodium (voliteľné)

```
wget https://download.libsodium.org/libsodium/releases/LATEST.tar.gz
tar xvf LATEST.tar.gz
cd libsodium-stable/
./configure
make && make check
sudo make install
```

3. Kompilácie knižnice TiigerTLS

```
git clone https://github.com/Crypto-TII/TLS1.3.git
cd TLS1.3/cpp
mkdir -p sal/miracl/includes
cp /core/cpp/core.a sal/miracl
cp /core/cpp/*.h sal/miracl/includes

// ak chceme pouzit libsodium tak pouzijeme prepinac -DSAL=
   MIRACL_SODIUM
cmake -DSAL=MIRACL -D CMAKE_CXX_COMPILER=/usr/bin/gcc
cd CMakeFiles/client.dir
nano link.txt
```

Na koniec riadku za prepínač „-lsodium“ pridáme prepínač „-lstdc++“

```
cd ../..
make
./client
```

WINDOWS

Server

1. Príprava knižníc a nástrojov

- stiahneme a naištalujeme balík nástrojov WinLibs – `https://winlibs.com`
- stiahneme a nainštalujeme Python pre Windows zo stránky – `https://www.python.org/downloads/windows/`
- stiahneme knižnicu MIRACL core – `https://github.com/miracl/core/archive/refs/heads/master.zip`
- stiahneme a nainštalujeme kompilačné nástroje pre jazyk Rust – `https://win.rustup.rs/`
- stiahneme knižnicu libsodium (voliteľné) – `https://download.libsodium.org/libsodium/releases/libsodium-1.0.19-stable-mingw.tar.gz`

2. Vygenerovanie súborov knižnice MIRACL core

```
cd core/rust
python config64.py test
cd ../..

cd ./TLS1.3/rust/server/
notepad.exe .\Cargo.toml
```

V súbore *Cargo.toml* je nutné správne nalinkovať cestu k zložke *textbfmcore*, ktorá vznikla pri generovaní MIRACL core knižnice v predošlom kroku.

4. Kompilácia Rust servera

```
cargo build
cargo run
```

Client

1. Vygenerovanie súborov knižnice MIRACL core

```
cd core/cpp
python config64.py test
cd ../..
```

2. Inštalácia knižnice libsodium (voliteľné)

```
cd libsodium-stable/
./configure
make && make check
sudo make install
```

- do zložky kompilátora *C:/mingw64/lib/* skopírujeme nainštalovaný súbor *libsodium.a*
- do zložky kompilátora *C:/mingw64/lib/* skopírujeme vygenerovanú zložku *libsodium-win64/include/*

3. Kompilácie knižnice TiigerTLS

Kompilácia klienta na Windows platforme vyhodí chybu, ktorú sme opravili v rámci našej diplomovej práce.

```
cd TLS1.3/cpp
mkdir -p sal/miracl/includes
cp /core/cpp/core.a sal/miracl
cp /core/cpp/*.h sal/miracl/includes

cmake -DSAL=MIRACL_SODIUM -D CMAKE_CXX_COMPILER=D:/mingw64/bin/gcc.exe
    -G "Unix Makefiles"

cd CMakeFiles/client.dir
```

Na koniec riadku za prepínač „-lsodium“ pridáme prepínače „-lstdc++“ a „-lws2_32“

```
cd ../..
make
```

F Postup kompilácie knižnice PQ_TigerTLS

Základná verzia celého projektu je prispôsobená pre jednoduchú kompiláciu na Windows platforme.

Používateľ si môže vygenerovať vlastné súbory knižníc MIRACL core a liboqs alebo môže využiť súbory projektu:

- obsahuje súbory knižnice MIRACL core v zložke */sal/* pre rôzne OS, pred kompiláciou projektu stačí premenovať konkrétny priečinok na *miracl*
- obsahuje súbory knižnice liboqs 0.6.0 v zložke */liboqs/* pre rôzne OS:
  - v prípade Windows platformy je potrebné tieto súbory skopírovať do zložky prekladača, napr. *C:/mingw64/*
  - v linuxovom prostredí je potrebné tieto súbory vložiť do zložky */usr/local/*

Kompiláciu projektu vykonáme príkazmi:

```
// Windows
cmake -G "MinGW Makefiles" -DCMAKE_CXX_COMPILER=C:/mingw64/bin/gcc.exe
mingw32-make
```

alebo príkazom (podľa toho ako máme pomenovaný nástroj make)

```
// Windows & Linux
cmake -DCMAKE_CXX_COMPILER=/usr/bin/gcc
make
```

Vytvorený súbor môžeme spustiť príkazom

```
./client.exe -r test.openquantumsafe.org:6109
```

G Vzorový PQC certifikát vygenerovaný a podpísaný navrhnutou certifikačnou autoritou

Certifikát vygenerovaný a podpísaný použitím PQC algoritmu FALCON512

—————BEGIN CERTIFICATE—————
MIIHdDCCBNCgAwIBAgIUTuy7XoosJK1kE2fAvQcKSPjiOagwBwYFK84PAwswSDEZ
MBcGA1UEAwwQSlNfbWFzdGVyX3RoZXNpczELMAkGA1UEBhMCU0sxDzANBgNVBAgM
Bktvc2ljZTENMAsGA1UECgwEVFVLRTAeFw0yNDA0MTgxODIwMTZaFw0yNTA0MTgx
ODIwMTZaMEgxGTAXBgNVBAMMEEpTX21hc3Rlcl90aGVzaXMxCzAJBgNVBAYTAlNL
MQ8wDQYDVQQIDAZLb3NpY2UxDTALBgNVBAoMBFRVS0UwggOPMAcGBSvODwMLA4ID
ggAJrbVRdijQGkcuxhNQ4ayOzlPCk2jWVUBF2bTvUHl0EXT2X29Vxa1rBCFylwRl
1biI3Wtl+kBFAoc+0aTqKY2WhQ0ecWHrdqSiF0XW5cA7xK2YXSkAfUbfdmFE8CPG
HMuNJdyDpVOngtFXZgoOgbjWYp2QxwNgDMVM+J2EQpLNfj3PGDIF/E3B0Gt/Kf1D
/YI1Agpyq2l4E+oILHaFtznFhhCH2oHBJs08Y4JqRA3c826Bt2dLz/t6AgE7vuz+
kmU9qpokHJY9hbtyI9Mu5iyh79+qczAv8U2iTh8cwJWZkKEN3NNlt2r5fcJeFK8R
5XK6o4uJj4KvbiF2PsTbAExCZ3PFAmUdV7cDwh45aHIgbdtPJXrcA+vZhYQVpNn7
E7Scrd+6ZJgpWAEye6jkY0FkxMTon19TXHQmkiQwBJVbJncImIpJWongTN10Bujl
PC79vE7QNcEeRDtIXdoXJtYbfQvI9lUjmYwwkKpmNbPQ1he1UbiDzYSTj4NqS3gf
etjEfAf582iOYACuRAR0WWplcYBNyi2W+1zi4QIxWbi4bkxhcw+Qiq5S6mYsNRed
jKB6JdCz2INVAmD0HOZo2DpX66GagSe8n96hBg6i0xcKljwuYZVmen85U3oHG2wu
LvCGUgjGnQCJFFCXUiJWxPfx03yXbXfzVp/ne97e8RFXRxyIegO2loQpgHxCQhrf
L4IP8n8HzBeqWXKcGQNouvcIN2FPUnl3xc/Pe1XJT5n7XtVTUC/B7c2lLUEmOE5Q
y1GMPKf0lhF5zRFAzZ5GNyhfKXLJW5xAqMVdhPSwJc+V+oWgC6U6ylzYtpaOGur9
AW23pnWmIka+ARz/qg3aWxyccmaeEKCsZUnXlYayUfD8irbHAl8t13kJb3w3lnS2
SYdIgFIXALkqz3+iZ3Zhqz8VtOIxxyYhXCps2phsk7ROTtVjaSQKpn/pYqkOXq22
mFGYs6/GdpDHYrW3+muiG6RCVuRaSLvnvZm0zFSm2elukglQ4qRFKtoF4+J1EGBS
WAITnVcugR75YMsmQNAeaNRi53L2QGBVgHpF7jjh6YwkJo4cF0TOE19Qz/DWhyke
Nayo+NubvvbqM6OEf0MNVer6VcdSehlZGo0wIiwt4//t1jnF93dM1cGzBjdhEhFW
lunzw4cU2EJ9MZF9ipw5jXHCN4jMtxosAhrGfiM9n+n0GrCjYzBhMB0GA1UdDgQW
BBS5iz8Dv5lh4XYAmCAuKWk1FvGQlDAfBgNVHSMEGDAWgBS5iz8Dv5lh4XYAmCAu
KWk1FvGQlDAPBgNVHRMBAf8EBTADAQH/MA4GA1UdDwEB/wQEAwIBhjAHBgUrzg8D
CwOCApMAOf2WzhWRKeKAMxyshxBCgrmzmUB+ThUYl/gnAELSfKNY9Iu2vjTzlAc/

```
IE4MwaVpffj29uyX7+135zcYs7BR4leMT8jqS+XlQF5UXZu3k5TOxuHe4CwlrzaI
yVpspifxJWaRmzEplXrr0OPy0Xxm53Bj8/hjrw6ZTUhVDoOhT2eags5VG6liXIA5
BMbv8sXm7nplXfarSp3cAgp6fq0TJSRKXValkIcohpHBwUtYlqL7wWqWaiuXrM6s
b+EUR01i/oy1btSSKpw0T7WK+J6wmmqI2PjJs8+aVqucp1KstqYeKxZFy2ce19P6
XQsyBL6WfQMEy0bQ5ioOhiFp7SR/MzEJfR5hQ+nR0DtnJrcUanXjGIwu+JLVgifT
64qyhTup/Z9egBZY55E61uYO/To0T8hMDnfNq6w+3m3rYfKA5fhV6092GzORV2RE
6IaqXrNIy6x2pyWZoecJ37YPTX+YUwL6MJrJPijCoGtMWtFCafG4t8c2T2lEXR3I
NrD04SdYaoirSuiR3PowS2UY8oMGJ0tUfdngscnH/nK9SCSwQ4MJmcKHppRwd9He
iqZl2K1yQrUbB1HWInyFEQQ48jrjBIVFdpkVL3qCyOjlb3m1fh9U0yCzc/BEIEEg
zc1ugpGQg1TEaDlvL6EEhdUYZ10M2pJ0ezSNabLd/FW6qMIQyPpbjYqdpp0qx2C5
uwQXJlNuKb2LLsHnHLfiL/WiKpa6uhvM0rj7oijC2V9NjiDLosZx+yfxJBnWnPmQ
Kuzv4MRgkJatcPniST6iEeNNh0MNZkmW2eKHTOQqs3mEq2+LznKxLSzl4uL2p5OS
aCU0Gcpj1+C7yEZ9jmO3yG6Zs3tmPTSEtrX5psDj5U68SjsAokvF6g==
-----END CERTIFICATE-----
```

Vlastnosti certifikátu

Vlastnosti a informácie o certifikáte získané príkazom:

```
openssl x509 -noout -text -in ./falcon512_CA.crt
```

```
    Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            4e:ec:bb:5e:8a:2c:24:ad:64:13:67:c0:bd:07:0a:48:f8:e2:39:
                a8
        Signature Algorithm: falcon512
        Issuer: CN=JS_master_thesis, C=SK, ST=Kosice, O=TUKE
        Validity
            Not Before: Apr 18 18:20:16 2024 GMT
            Not After : Apr 18 18:20:16 2025 GMT
        Subject: CN=JS_master_thesis, C=SK, ST=Kosice, O=TUKE
        Subject Public Key Info:
            Public Key Algorithm: falcon512
                falcon512 public key:
                PQ key material:
                    09:ad:b5:51:76:28:d0:1a:47:2e:c6:13:50:e1:ac:
                    8e:ce:53:c2:93:68:d6:55:40:45:d9:b4:ef:50:79:
                    74:11:74:f6:5f:6f:55:c5:ad:6b:04:21:72:97:04:
                    65:95:b8:88:dd:6b:65:fa:40:45:02:87:3e:d1:a4:
                    ea:29:8d:96:85:0d:1e:71:61:eb:76:a4:a2:17:45:
                    d6:e5:c0:3b:c4:ad:98:5d:29:00:7d:46:df:76:61:
                    44:f0:23:c6:1c:cb:8d:25:dc:83:a5:53:a7:82:d1:
                    57:66:0a:0e:81:b8:d6:62:9d:90:c7:03:60:0c:c5:
                    4c:f8:9d:84:42:92:cd:7e:3d:cf:18:32:05:fc:4d:
```

c1:d0:6b:7f:29:fd:43:fd:82:35:02:0a:72:ab:69:
78:13:ea:08:2c:76:85:b7:39:c5:86:10:87:da:81:
c1:26:**cd**:3c:63:82:6a:44:0d:dc:f3:6e:81:b7:67:
4b:cf:fb:7a:02:01:3b:be:ec:fe:92:65:3d:aa:9a:
24:1c:96:3d:85:bb:72:23:d3:2e:e6:2c:a1:ef:df:
aa:73:30:2f:f1:4d:a2:4e:1f:1c:c0:95:99:90:a1:
0d:dc:d3:65:b7:6a:f9:7d:c2:5e:14:af:11:e5:72:
ba:a3:8b:89:8f:82:af:6e:21:76:3e:c4:db:00:4c:
42:67:73:c5:02:65:1d:57:b7:03:c2:1e:39:68:72:
20:6d:db:4f:25:7a:dc:03:eb:d9:85:84:15:a4:d9:
fb:13:b4:9c:ad:df:ba:64:98:29:58:01:32:7b:a8:
e4:63:41:64:c4:c4:e8:9f:5f:53:5c:74:26:92:24:
30:04:95:5b:26:77:08:98:8a:49:5a:89:e0:4c:dd:
74:06:e8:e5:3c:2e:fd:bc:4e:d0:35:c1:1e:44:3b:
48:5d:da:17:26:d6:1b:7d:0b:c8:f6:55:23:99:8c:
30:90:aa:66:35:b3:d0:d6:17:b5:51:b8:83:**cd**:84:
93:8f:83:6a:4b:78:1f:7a:d8:c4:7c:07:f9:f3:68:
8e:60:00:ae:44:04:74:59:6a:65:71:80:4d:ca:2d:
96:fb:5c:e2:e1:02:31:59:b8:b8:6e:4c:61:73:0f:
90:8a:ae:52:ea:66:2c:35:17:9d:8c:a0:7a:25:d0:
b3:d8:83:55:02:60:f4:1c:e6:68:d8:3a:57:eb:a1:
9a:81:27:bc:9f:de:a1:06:0e:a2:d3:17:0a:96:3c:
2e:61:95:66:7a:7f:39:53:7a:07:1b:6c:2e:2e:f0:
86:52:08:c6:9d:00:89:14:50:97:52:22:56:c4:f7:
f1:d3:7c:97:6d:77:f3:56:9f:e7:7b:de:de:f1:11:
57:47:1c:88:7a:03:b6:96:84:29:80:7c:42:42:1a:
df:2f:82:0f:f2:7f:07:cc:17:aa:59:72:9c:19:03:
68:ba:f7:08:37:61:4f:52:79:77:c5:cf:cf:7b:55:
c9:4f:99:fb:5e:d5:53:50:2f:c1:ed:**cd**:a5:2d:41:
26:38:4e:50:cb:51:8c:3c:a7:f4:96:11:79:**cd**:11:
40:**cd**:9e:46:37:28:5f:29:72:c9:5b:9c:40:a8:c5:
5d:84:f4:b0:25:cf:95:fa:85:a0:0b:a5:3a:ca:5c:
d8:b6:96:8e:1a:ea:fd:01:6d:b7:a6:75:a6:22:46:
be:01:1c:ff:aa:0d:da:5b:1c:9c:72:66:9e:10:a0:
ac:65:49:d7:95:86:b2:51:f0:**fc**:8a:b6:c7:02:5f:
2d:d7:79:09:6f:7c:37:96:74:b6:49:87:48:80:52:
17:00:b9:2a:cf:7f:a2:67:76:61:ab:3f:15:b4:e2:
31:c7:26:21:5c:2a:6c:da:98:6c:93:b4:4e:4e:d5:
63:69:24:0a:a6:7f:e9:62:a9:0e:5e:ad:b6:98:51:
98:b3:af:c6:76:90:c7:62:b5:b7:fa:6b:a2:1b:a4:
42:56:e4:5a:48:bb:e7:bd:99:b4:cc:54:a6:d9:e9:
6e:92:09:50:e2:a4:45:2a:da:05:e3:e2:65:10:60:
52:58:02:13:9d:57:2e:81:1e:f9:60:cb:26:40:d0:
1e:68:d4:62:e7:72:f6:40:60:55:80:7a:45:ee:38:
e1:e9:8c:24:26:8e:1c:17:44:ce:13:5f:50:cf:f0:

```
                d6:87:29:1e:35:ac:a8:f8:db:9b:be:f6:ea:33:a3:
                84:7f:43:0d:55:ea:fa:55:c7:52:7a:19:59:1a:8d:
                30:22:2c:2d:e3:ff:ed:d6:39:c5:f7:77:4c:d5:c1:
                b3:06:37:61:12:11:56:96:e9:f3:c3:87:14:d8:42:
                7d:31:91:7d:8a:9c:39:8d:71:c2:37:88:cc:b7:1a:
                2c:02:1a:c6:7e:23:3d:9f:e9:f4:1a:b0
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                B9:8B:3F:03:BF:99:61:E1:76:00:98:20:2E:29:69:35:16:F1
                    :90:94
            X509v3 Authority Key Identifier:
                B9:8B:3F:03:BF:99:61:E1:76:00:98:20:2E:29:69:35:16:F1
                    :90:94
            X509v3 Basic Constraints: critical
                CA:TRUE
            X509v3 Key Usage: critical
                Digital Signature, Certificate Sign, CRL Sign
    Signature Algorithm: falcon512
    Signature Value:
        39:fd:96:ce:15:91:29:e2:80:33:1c:ac:87:10:42:82:b9:b3:
        99:40:7e:4e:15:18:97:f8:27:00:42:d2:7c:a3:58:f4:8b:b6:
        be:34:f3:94:07:3f:20:4e:0c:c1:a5:69:7d:f8:f6:f6:ec:97:
        ef:ed:77:e7:37:18:b3:b0:51:e2:57:8c:4f:c8:ea:4b:e5:e5:
        40:5e:54:5d:9b:b7:93:94:ce:c6:e1:de:e0:2c:25:af:36:88:
        c9:5a:6c:a6:27:f1:25:66:91:9b:31:29:95:7a:eb:d0:e3:f2:
        d1:7c:66:e7:70:63:f3:f8:63:af:0e:99:4d:48:55:0e:83:a1:
        4f:67:9a:82:ce:55:1b:a9:62:5c:80:39:04:c6:ef:f2:c5:e6:
        ee:7a:65:5d:f6:ab:4a:9d:dc:02:0a:7a:7e:ad:13:25:24:4a:
        5d:56:a5:90:87:28:86:91:c1:c1:4b:58:96:a2:fb:c1:6a:96:
        6a:2b:97:ac:ce:ac:6f:e1:14:47:4d:62:fe:8c:b5:6e:d4:92:
        2a:9c:34:4f:b5:8a:f8:9e:b0:9a:6a:88:d8:f8:c9:b3:cf:9a:
        56:ab:9c:a7:52:ac:b6:a6:1e:2b:16:45:cb:67:1e:d7:d3:fa:
        5d:0b:32:04:be:96:7d:03:04:cb:46:d0:e6:2a:0e:86:21:69:
        ed:24:7f:33:31:09:7d:1e:61:43:e9:d1:d0:3b:67:26:b7:14:
        6a:75:e3:18:8c:2e:f8:92:d5:82:27:d3:eb:8a:b2:85:3b:a9:
        fd:9f:5e:80:16:58:e7:91:3a:d6:e6:0e:fd:3a:34:4f:c8:4c:
        0e:77:cd:ab:ac:3e:de:6d:eb:61:f2:80:e5:f8:55:eb:4f:76:
        1b:33:91:57:64:44:e8:86:aa:5e:b3:48:cb:ac:76:a7:25:99:
        a1:e7:09:df:b6:0f:4d:7f:98:53:02:fa:30:9a:c9:3e:28:c2:
        a0:6b:4c:5a:d1:42:69:f1:b8:b7:c7:36:4f:69:44:5d:1d:c8:
        36:b0:f4:e1:27:58:6a:88:ab:4a:e8:91:dc:fa:30:4b:65:18:
        f2:83:06:27:4b:54:7d:d9:e0:b1:c9:c7:fe:72:bd:48:24:b0:
        43:83:09:99:c2:87:a6:94:70:77:d1:de:8a:a6:65:d8:ad:72:
        42:b5:1b:07:51:d6:22:7c:85:11:04:38:f2:3a:e3:04:85:45:
        76:99:15:2f:7a:82:c8:e8:e5:6f:79:b5:7e:1f:54:d3:20:b3:
```

73:f0:44:20:41:20:**cd**:**cd**:6e:82:91:90:83:54:c4:68:39:6f:
2f:a1:04:85:d5:18:67:5d:0c:da:92:74:7b:34:8d:69:b2:dd:
fc:55:ba:a8:c2:10:c8:fa:5b:8d:8a:9d:a6:9d:2a:c7:60:b9:
bb:04:17:26:53:6e:29:bd:8b:2e:c1:e7:1c:b7:e2:2f:f5:a2:
2a:96:ba:ba:1b:cc:d2:b8:fb:a2:28:c2:d9:5f:4d:8e:20:cb:
a2:c6:71:fb:27:f1:24:19:d6:9c:f9:90:2a:ec:ef:e0:c4:60:
90:96:ad:70:f9:e2:49:3e:a2:11:e3:4d:87:43:0d:66:49:96:
d9:e2:87:4c:e4:2a:b3:79:84:ab:6f:8b:ce:72:b1:2d:2c:e5:
e2:e2:f6:a7:93:92:68:25:34:19:ca:63:d7:e0:bb:c8:46:7d:
8e:63:b7:c8:6e:99:b3:7b:66:3d:34:84:b6:b5:f9:a6:c0:e3:
e5:4e:bc:4a:3b:00:a2:4b:c5:ea

H Výsledky overovania spojenia s testovacím serverom OQS

Testovanie č. 1

Testované na Ubuntu 22.04.3
liboqs 0.9.2, oqs-provider 0.5.3
1.3.2024

CERT	KEM	PORT	STATUS
dilithium2	bikel1	6087	OK
dilithium2	frodo640aes	6088	OK
dilithium2	frodo640shake	6089	OK
dilithium2	hqc128	6090	KEX error (kex=-1)
dilithium2	kyber512	6091	OK
dilithium3	bikel3	6105	OK
dilithium3	frodo976aes	6106	OK
dilithium3	frodo976shake	6107	OK
dilithium3	hqc192	6108	KEX error (kex=-1)
dilithium3	kyber768	6109	OK
dilithium5	bikel5	6121	OK
dilithium5	frodo1344aes	6122	OK
dilithium5	frodo1344shake	6123	OK
dilithium5	hqc256	6124	KEX error (kex=-1)
dilithium5	kyber1024	6125	OK

CERT	KEM	PORT	STATUS
falcon1024	bikel5	6132	Server Certificate sig is NOT OK
falcon1024	frodo1344aes	6133	Server Certificate sig is NOT OK
falcon1024	frodo1344shake	6134	Server Certificate sig is NOT OK
falcon1024	hqc256	6135	KEX error (kex=-1)
falcon1024	kyber1024	6136	Server Certificate sig is NOT OK
falcon512	bikel1	6143	Server Certificate sig is NOT OK
falcon512	frodo640aes	6144	Server Certificate sig is NOT OK
falcon512	frodo640shake	6145	Server Certificate sig is NOT OK
falcon512	hqc128	6146	KEX error (kex=-1)
falcon512	kyber512	6147	Server Certificate sig is NOT OK

CERT	KEM	PORT	STATUS
sphincssha2128fsimple	bikel1	6143	OK
sphincssha2128fsimple	frodo640aes	6144	OK
sphincssha2128fsimple	frodo640shake	6145	OK
sphincssha2128fsimple	hqc128	6146	KEX error (kex=-1)
sphincssha2128fsimple	kyber512	6147	OK
sphincssha2128ssimple	bikel1	6179	OK
sphincssha2128ssimple	frodo640aes	6180	OK
sphincssha2128ssimple	frodo640shake	6181	OK
sphincssha2128ssimple	hqc128	6182	KEX error (kex=-1)
sphincssha2128ssimple	kyber512	6183	OK
sphincsshake128fsimple	bikel1	6213	OK
sphincsshake128fsimple	frodo640aes	6214	OK
sphincsshake128fsimple	frodo640shake	6215	OK
sphincsshake128fsimple	hqc128	6216	KEX error (kex=-1)
sphincsshake128fsimple	kyber512	6217	OK

Testovanie č. 2

Testované na Ubuntu 22.04.3
liboqs 0.10.0, oqs-provider 0.6.0
15.4.2024

CERT	KEM	PORT	STATUS
dilithium2	bikel1	6109	OK
dilithium2	frodo640aes	6110	OK
dilithium2	frodo640shake	6111	OK
dilithium2	hqc128	6112	OK
dilithium2	kyber512	6113	OK
dilithium2	mlkem512	6114	OK
dilithium3	bikel3	6132	OK
dilithium3	frodo976aes	6133	OK
dilithium3	frodo976shake	6134	OK
dilithium3	hqc192	6135	OK
dilithium3	kyber768	6136	OK
dilithium3	mlkem768	6137	OK
dilithium5	bikel5	6152	OK
dilithium5	frodo1344aes	6153	OK
dilithium5	frodo1344shake	6154	OK
dilithium5	hqc256	6155	OK
dilithium5	kyber1024	6156	OK
dilithium5	mlkem1024	6157	OK

CERT	KEM	PORT	STATUS
falcon1024	bikel5	6165	OK
falcon1024	frodo1344aes	6166	OK
falcon1024	frodo1344shake	6167	OK
falcon1024	hqc256	6168	OK
falcon1024	kyber1024	6169	OK
falcon1024	mlkem1024	6170	OK
falcon512	bikel1	6178	OK
falcon512	frodo640aes	6179	OK
falcon512	frodo640shake	6180	OK
falcon512	hqc128	6181	OK
falcon512	kyber512	6182	OK
falcon512	mlkem512	6183	OK

CERT	KEM	PORT	STATUS
mldsa44	bikel1	6237	OK
mldsa44	frodo640aes	6238	OK
mldsa44	frodo640shake	6239	OK
mldsa44	hqc128	6240	OK
mldsa44	kyber512	6241	OK
mldsa44	mlkem512	6242	OK
mldsa65	bikel3	6260	OK
mldsa65	frodo976aes	6261	OK
mldsa65	frodo976shake	6262	OK
mldsa65	hqc192	6263	OK
mldsa65	kyber768	6264	OK
mldsa65	mlkem768	6265	OK
mldsa87	bikel5	6280	OK
mldsa87	frodo1344aes	6281	OK
mldsa87	frodo1344shake	6282	OK
mldsa87	hqc256	6283	OK
mldsa87	kyber1024	6284	OK
mldsa87	mlkem1024	6285	OK

CERT	KEM	PORT	STATUS
sphincssha2128fsimple	bikel1	6293	OK
sphincssha2128fsimple	frodo640aes	6294	OK
sphincssha2128fsimple	frodo640shake	6295	OK
sphincssha2128fsimple	hqc128	6296	OK
sphincssha2128fsimple	kyber512	6297	OK
sphincssha2128fsimple	mlkem512	6298	OK
sphincssha2128ssimple	bikel1	6316	OK
sphincssha2128ssimple	frodo640aes	6317	OK
sphincssha2128ssimple	frodo640shake	6318	OK
sphincssha2128ssimple	hqc128	6319	OK
sphincssha2128ssimple	kyber512	6320	OK
sphincssha2128ssimple	mlkem512	6321	OK
sphincsshake128fsimple	bikel1	6359	OK
sphincsshake128fsimple	frodo640aes	6360	OK
sphincsshake128fsimple	frodo640shake	6361	OK
sphincsshake128fsimple	hqc128	6362	OK
sphincsshake128fsimple	kyber512	6363	OK
sphincsshake128fsimple	mlkem512	6364	OK

I Výsledky meraní časov generovania KEM kľúčov a overovania PQ podpisov

Testovanie č. 1

Testované na Windows platforme, počítač A
liboqs 0.9.2
1.3.2024

KYBER512

Key Generation 0.09 ms

Public key size: 800 bytes

Secret key size: 1632 bytes

KYBER768

Key Generation 0.10 ms

Public key size: 1184 bytes

Secret key size: 2400 bytes

KYBER1024

Key Generation 0.12 ms

Public key size: 1568 bytes

Secret key size: 3168 bytes

HQC128

Key Generation 0.11 ms

Public key size: 2249 bytes

Secret key size: 2289 bytes

HQC192

Key Generation 0.21 ms

Public key size: 4522 bytes

Secret key size: 4562 bytes

HQC256

Key Generation 0.35 ms

Public key size: 7245 bytes

Secret key size: 7285 bytes

FRODO640AES

Key Generation 0.22 ms

Public key size: 9616 bytes

Secret key size: 19888 bytes

FRODO640SHAKE

Key Generation 1.43 ms

Public key size: 9616 bytes

Secret key size: 19888 bytes

FRODO976AES

Key Generation 0.39 ms

Public key size: 15632 bytes

Secret key size: 31296 bytes

FRODO976SHAKE

Key Generation 3.14 ms

Public key size: 15632 bytes

Secret key size: 31296 bytes

FRODO1344AES

Key Generation 0.64 ms

Public key size: 21520 bytes

Secret key size: 43088 bytes

FRODO1344SHAKE

Key Generation 5.67 ms

Public key size: 21520 bytes

Secret key size: 43088 bytes

Testovanie č. 2

Testované na Windows platforme, počítač A
liboqs 0.10.0
14.4.2024

KYBER512

Key Generation 0.09 ms

Public key size: 800 bytes

Secret key size: 1632 bytes

KYBER768

Key Generation 0.10 ms

Public key size: 1184 bytes

Secret key size: 2400 bytes

KYBER1024

Key Generation 0.12 ms

Public key size: 1568 bytes

Secret key size: 3168 bytes

MLKEM512

Key Generation 0.05 ms

Public key size: 800 bytes

Secret key size: 1632 bytes

MLKEM768

Key Generation 0.07 ms

Public key size: 1184 bytes

Secret key size: 2400 bytes

MLKEM1024

Key Generation 0.09 ms

Public key size: 1568 bytes

Secret key size: 3168 bytes

HQC128

Key Generation 1.32 ms

Public key size: 2249 bytes

Secret key size: 2305 bytes

HQC192

Key Generation 3.85 ms

Public key size: 4522 bytes

Secret key size: 4586 bytes

HQC256

Key Generation 6.95 ms

Public key size: 7245 bytes

Secret key size: 7317 bytes

FRODO640AES

Key Generation 0.22 ms

Public key size: 9616 bytes

Secret key size: 19888 bytes

FRODO640SHAKE

Key Generation 1.45 ms

Public key size: 9616 bytes

Secret key size: 19888 bytes

FRODO976AES

Key Generation 0.39 ms

Public key size: 15632 bytes

Secret key size: 31296 bytes

FRODO976SHAKE

Key Generation 3.20 ms

Public key size: 15632 bytes

Secret key size: 31296 bytes

FRODO1344AES

Key Generation 0.66 ms

Public key size: 21520 bytes

Secret key size: 43088 bytes

FRODO1344SHAKE

Key Generation 5.76 ms

Public key size: 21520 bytes

Secret key size: 43088 bytes

Testovanie č. 3

Testované na Windows platforme, počítač B
liboqs 0.10.0
16.4.2024

KYBER512

Key Generation 0.35 ms

Public key size: 800 bytes

Secret key size: 1632 bytes

KYBER768

Key Generation 0.41 ms

Public key size: 1184 bytes

Secret key size: 2400 bytes

KYBER1024

Key Generation 0.44 ms

Public key size: 1568 bytes

Secret key size: 3168 bytes

MLKEM512

Key Generation 0.20 ms

Public key size: 800 bytes

Secret key size: 1632 bytes

MLKEM768

Key Generation 0.24 ms

Public key size: 1184 bytes

Secret key size: 2400 bytes

MLKEM1024

Key Generation 0.28 ms

Public key size: 1568 bytes

Secret key size: 3168 bytes

HQC128

Key Generation 2.70 ms

Public key size: 2249 bytes

Secret key size: 2305 bytes

HQC192

Key Generation 7.29 ms

Public key size: 4522 bytes

Secret key size: 4586 bytes

HQC256

Key Generation 14.93 ms

Public key size: 7245 bytes

Secret key size: 7317 bytes

FRODO640AES

Key Generation 1.20 ms

Public key size: 9616 bytes

Secret key size: 19888 bytes

FRODO640SHAKE

Key Generation 4.59 ms

Public key size: 9616 bytes

Secret key size: 19888 bytes

FRODO976AES

Key Generation 1.33 ms

Public key size: 15632 bytes

Secret key size: 31296 bytes

FRODO976SHAKE

Key Generation 9.28 ms

Public key size: 15632 bytes

Secret key size: 31296 bytes

FRODO1344AES

Key Generation 3.58 ms

Public key size: 21520 bytes

Secret key size: 43088 bytes

FRODO1344SHAKE

Key Generation 15.80 ms

Public key size: 21520 bytes

Secret key size: 43088 bytes

J Výsledky meraní časovej náročnosti overovania PQ certifikátov

Test č. 1

Testovane na zariadeni A
liboqs 0.10.0
18.04.2024

RSA_PSS_RSAE_SHA256

Elapsed time: 0.20 miliseconds
Elapsed time: 0.18 miliseconds
Elapsed time: 0.18 miliseconds

Elapsed time: 0.21 miliseconds
Elapsed time: 0.19 miliseconds
Elapsed time: 0.18 miliseconds

Elapsed time: 0.29 miliseconds
Elapsed time: 0.19 miliseconds
Elapsed time: 0.20 miliseconds

ECDSA_SECP384R1_SHA384

Elapsed time: 0.29 miliseconds
Elapsed time: 0.72 miliseconds
Elapsed time: 0.77 miliseconds

Elapsed time: 0.20 miliseconds

Elapsed time: 0.72 miliseconds
Elapsed time: 0.84 miliseconds

Elapsed time: 0.20 miliseconds
Elapsed time: 0.72 miliseconds
Elapsed time: 0.75 miliseconds

DILITHIUM2

Elapsed time: 0.12 miliseconds
Elapsed time: 0.73 miliseconds
Elapsed time: 0.07 miliseconds

Elapsed time: 0.08 miliseconds
Elapsed time: 0.73 miliseconds
Elapsed time: 0.08 miliseconds

Elapsed time: 0.08 miliseconds
Elapsed time: 0.73 miliseconds
Elapsed time: 0.07 miliseconds

DILITHIUM3

Elapsed time: 0.13 miliseconds
Elapsed time: 0.72 miliseconds
Elapsed time: 0.11 miliseconds

Elapsed time: 0.12 miliseconds
Elapsed time: 0.73 miliseconds
Elapsed time: 0.11 miliseconds

Elapsed time: 0.13 miliseconds
Elapsed time: 0.73 miliseconds
Elapsed time: 0.11 miliseconds

DILITHIUM5

Elapsed time: 0.21 miliseconds
Elapsed time: 0.73 miliseconds

Elapsed time: 0.28 miliseconds

Elapsed time: 0.20 miliseconds
Elapsed time: 0.75 miliseconds
Elapsed time: 0.19 miliseconds

Elapsed time: 0.20 miliseconds
Elapsed time: 0.74 miliseconds
Elapsed time: 0.19 miliseconds

FALCON512

Elapsed time: 0.07 miliseconds
Elapsed time: 0.73 miliseconds
Elapsed time: 0.03 miliseconds

Elapsed time: 0.05 miliseconds
Elapsed time: 0.74 miliseconds
Elapsed time: 0.03 miliseconds

Elapsed time: 0.05 miliseconds
Elapsed time: 0.75 miliseconds
Elapsed time: 0.03 miliseconds

FALCON1024

Elapsed time: 0.10 miliseconds
Elapsed time: 0.74 miliseconds
Elapsed time: 0.06 miliseconds

Elapsed time: 0.08 miliseconds
Elapsed time: 0.74 miliseconds
Elapsed time: 0.06 miliseconds

Elapsed time: 0.08 miliseconds
Elapsed time: 0.74 miliseconds
Elapsed time: 0.06 miliseconds

MLDSA44

Elapsed time: 0.10 miliseconds
Elapsed time: 0.73 miliseconds
Elapsed time: 0.11 miliseconds

Elapsed time: 0.08 miliseconds
Elapsed time: 0.74 miliseconds
Elapsed time: 0.07 miliseconds

Elapsed time: 0.09 miliseconds
Elapsed time: 0.74 miliseconds
Elapsed time: 0.08 miliseconds

MLDSA65

Elapsed time: 0.14 miliseconds
Elapsed time: 0.75 miliseconds
Elapsed time: 0.11 miliseconds

Elapsed time: 0.13 miliseconds
Elapsed time: 0.74 miliseconds
Elapsed time: 0.11 miliseconds

Elapsed time: 0.13 miliseconds
Elapsed time: 0.74 miliseconds
Elapsed time: 0.17 miliseconds

MLDSA87

Elapsed time: 0.20 miliseconds
Elapsed time: 0.73 miliseconds
Elapsed time: 0.18 miliseconds

Elapsed time: 0.19 miliseconds
Elapsed time: 0.74 miliseconds
Elapsed time: 0.27 miliseconds

Elapsed time: 0.19 miliseconds

Elapsed time: 0.74 miliseconds
Elapsed time: 0.23 miliseconds

SPHINCS_SHA2_128F_SIMPLE

Elapsed time: 1.49 miliseconds
Elapsed time: 0.73 miliseconds
Elapsed time: 1.42 miliseconds

Elapsed time: 1.48 miliseconds
Elapsed time: 0.73 miliseconds
Elapsed time: 1.39 miliseconds

Elapsed time: 1.49 miliseconds
Elapsed time: 0.74 miliseconds
Elapsed time: 1.39 miliseconds

SPHINCS_SHA2_128S_SIMPLE

Elapsed time: 0.53 miliseconds
Elapsed time: 0.74 miliseconds
Elapsed time: 0.54 miliseconds

Elapsed time: 0.67 miliseconds
Elapsed time: 0.73 miliseconds
Elapsed time: 0.52 miliseconds

Elapsed time: 0.58 miliseconds
Elapsed time: 0.73 miliseconds
Elapsed time: 0.70 miliseconds

SPHINCS_SHAKE_128F_SIMPLE

Elapsed time: 2.24 miliseconds
Elapsed time: 0.72 miliseconds
Elapsed time: 2.24 miliseconds

Elapsed time: 2.23 miliseconds
Elapsed time: 0.74 miliseconds

Elapsed time: 2.26 miliseconds

Elapsed time: 2.23 miliseconds
Elapsed time: 0.72 miliseconds
Elapsed time: 2.43 miliseconds

Test č. 2

Testovane na zariadeni B
liboqs 0.10.0
18.04.2024

RSA_PSS_RSAE_SHA256

Elapsed time: 0.42 miliseconds
Elapsed time: 0.38 miliseconds
Elapsed time: 0.42 miliseconds

Elapsed time: 1.04 miliseconds
Elapsed time: 0.50 miliseconds
Elapsed time: 0.41 miliseconds

Elapsed time: 0.59 miliseconds
Elapsed time: 0.50 miliseconds
Elapsed time: 0.50 miliseconds

ECDSA_SECP256R1_SHA256

Elapsed time: 0.53 miliseconds
Elapsed time: 0.43 miliseconds
Elapsed time: 0.97 miliseconds

Elapsed time: 0.71 miliseconds
Elapsed time: 0.51 miliseconds
Elapsed time: 0.91 miliseconds

Elapsed time: 0.42 miliseconds

Elapsed time: 0.59 miliseconds
Elapsed time: 0.75 miliseconds

DILITHIUM2

Elapsed time: 0.42 miliseconds
Elapsed time: 1.41 miliseconds
Elapsed time: 0.18 miliseconds

Elapsed time: 0.45 miliseconds
Elapsed time: 1.40 miliseconds
Elapsed time: 0.21 miliseconds

Elapsed time: 0.45 miliseconds
Elapsed time: 1.44 miliseconds
Elapsed time: 0.81 miliseconds

DILITHIUM3

Elapsed time: 0.68 miliseconds
Elapsed time: 1.49 miliseconds
Elapsed time: 0.33 miliseconds

Elapsed time: 0.68 miliseconds
Elapsed time: 1.40 miliseconds
Elapsed time: 0.33 miliseconds

Elapsed time: 0.68 miliseconds
Elapsed time: 1.40 miliseconds
Elapsed time: 0.35 miliseconds

DILITHIUM5

Elapsed time: 1.12 miliseconds
Elapsed time: 1.40 miliseconds
Elapsed time: 0.40 miliseconds

Elapsed time: 1.02 miliseconds
Elapsed time: 2.36 miliseconds

Elapsed time: 0.62 miliseconds

Elapsed time: 1.07 miliseconds
Elapsed time: 1.40 miliseconds
Elapsed time: 0.52 miliseconds

FALCON1024

Elapsed time: 0.59 miliseconds
Elapsed time: 3.83 miliseconds
Elapsed time: 0.35 miliseconds

Elapsed time: 0.18 miliseconds
Elapsed time: 1.51 miliseconds
Elapsed time: 0.15 miliseconds

Elapsed time: 0.28 miliseconds
Elapsed time: 1.91 miliseconds
Elapsed time: 0.19 miliseconds

FALCON512

Elapsed time: 0.47 miliseconds
Elapsed time: 4.07 miliseconds
Elapsed time: 0.20 miliseconds

Elapsed time: 0.28 miliseconds
Elapsed time: 1.41 miliseconds
Elapsed time: 0.08 miliseconds

Elapsed time: 0.29 miliseconds
Elapsed time: 3.69 miliseconds
Elapsed time: 0.26 miliseconds

MLDSA44

Elapsed time: 0.55 miliseconds
Elapsed time: 3.73 miliseconds
Elapsed time: 0.27 miliseconds

Elapsed time: 0.46 miliseconds
Elapsed time: 3.07 miliseconds
Elapsed time: 0.21 miliseconds

Elapsed time: 0.23 miliseconds
Elapsed time: 1.92 miliseconds
Elapsed time: 0.25 miliseconds

MLDSA65

Elapsed time: 0.74 miliseconds
Elapsed time: 1.48 miliseconds
Elapsed time: 0.24 miliseconds

Elapsed time: 0.67 miliseconds
Elapsed time: 2.50 miliseconds
Elapsed time: 0.37 miliseconds

Elapsed time: 0.68 miliseconds
Elapsed time: 1.42 miliseconds
Elapsed time: 0.61 miliseconds

MLDSA87

Elapsed time: 1.09 miliseconds
Elapsed time: 1.42 miliseconds
Elapsed time: 0.38 miliseconds

Elapsed time: 1.05 miliseconds
Elapsed time: 1.50 miliseconds
Elapsed time: 0.39 miliseconds

Elapsed time: 1.04 miliseconds

Elapsed time: 1.42 miliseconds
Elapsed time: 0.41 miliseconds

SPHINCS_SHA2_128F_SIMPLE

Elapsed time: 7.59 miliseconds
Elapsed time: 2.27 miliseconds
Elapsed time: 4.51 miliseconds

Elapsed time: 7.71 miliseconds
Elapsed time: 1.40 miliseconds
Elapsed time: 2.74 miliseconds

Elapsed time: 8.73 miliseconds
Elapsed time: 2.24 miliseconds
Elapsed time: 4.54 miliseconds

SPHINCS_SHA2_128S_SIMPLE

Elapsed time: 2.58 miliseconds
Elapsed time: 1.44 miliseconds
Elapsed time: 0.97 miliseconds

Elapsed time: 2.45 miliseconds
Elapsed time: 4.38 miliseconds
Elapsed time: 3.43 miliseconds

Elapsed time: 2.48 miliseconds
Elapsed time: 2.30 miliseconds
Elapsed time: 2.39 miliseconds

SPHINCS_SHAKE_128F_SIMPLE

Elapsed time: 10.59 miliseconds
Elapsed time: 1.46 miliseconds
Elapsed time: 4.11 miliseconds

Elapsed time: 10.41 miliseconds
Elapsed time: 2.39 miliseconds

Elapsed time: 6.33 miliseconds

Elapsed time: 10.92 miliseconds
Elapsed time: 2.46 miliseconds
Elapsed time: 6.58 miliseconds