SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE

Fakulta elektrotechniky a informatiky

Evidenčné číslo: FEI-16607-115378

Edukačná aplikácia o Signal protokole

Diplomová práca

2026 **Bc. Bence Both**

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE

Fakulta elektrotechniky a informatiky

Evidenčné číslo: FEI-16607-115378

Edukačná aplikácia o Signal protokole

Diplomová práca

Študijný program:	aplikovaná informatika
Študijný odbor:	informatika
Školiace pracovisko:	Ústav informatiky a matematiky
Školiteľ:	Mgr. Tomáš Fabšič, PhD.

2026 **Bc. Bence Both**

Slovenská technická univerzita v Bratislave
Ústav informatiky a matematiky

Fakulta elektrotechniky a informatiky
Akademický rok: 2025/2026
Evidenčné číslo: FEI-16607-115378

ZADANIE DIPLOMOVEJ PRÁCE

Študent:	**Bc. Bence Both**
ID študenta:	115378
Študijný program:	aplikovaná informatika
Študijný odbor:	informatika
Vedúci práce:	Mgr. Tomáš Fabšič, PhD.
Vedúci pracoviska:	doc. Ing. Milan Vojvoda, PhD.

Názov práce: **Edukačná aplikácia o Signal protokole**

Jazyk, v ktorom sa práca vypracuje: slovenský jazyk

Špecifikácia zadania:

Cieľom tejto diplomovej práce je vytvoriť edukačnú aplikáciu, ktorá používateľom pomôže porozumieť vybraným kryptografickým mechanizmom používaným v Signal protokole.

Úlohy:
1. Dôkladne sa oboznámte s kryptografickými mechanizmami používanými v Signal protokole.
2. Vytvorte edukačnú aplikáciu, ktorá používateľovi pomôže porozumieť vybraným kryptografickým mechanizmom používaným v Signal protokole.
3. Otestujte aplikáciu a vyhodnoťte výsledky.

Zoznam odbornej literatúry:

1. Cohn-Gordon, K., Cremers, C., Dowling, B. et al. (2020). A Formal Security Analysis of the Signal Messaging Protocol. J Cryptol 33, 1914–1983. https://doi.org/10.1007/s00145-020-09360-1
2. Dodis, Y., Jost, D., Katsumata, S., Prest, T., Schmidt, R. (2025). Triple Ratchet: A Bandwidth Efficient Hybrid-Secure Signal Protocol. In: Fehr, S., Fouque, PA. (eds) Advances in Cryptology – EUROCRYPT 2025. EUROCRYPT 2025. Lecture Notes in Computer Science, vol 15608. Springer, Cham. https://doi.org/10.1007/978-3-031-91101-9_11
3. Kret, E., Schmidt, R. (2024). The PQXDH key agreement protocol, revision 3. https://signal.org/docs/specifications/pqxdh/
4. Marlinspike, M., Perrin, T. (2016). The X3DH Key Agreement Protocol, revision 1. https://signal.org/docs/specifications/x3dh/
5. Perrin, T., Marlinspike, M., Schmidt, R. (2025). The double ratchet algorithm, revision 4. https://signal.org/docs/specifications/doubleratchet/

Termín odovzdania diplomovej práce:	15. 05. 2026
Dátum schválenia zadania diplomovej práce:	27. 03. 2026
Zadanie diplomovej práce schválil:	prof. Dr. rer. nat. Martin Drozda – garant študijného programu

Abstrakt

Táto diplomová práca sa zaoberá analýzou a vizualizáciou protokolu Signal, ktorý predstavuje súčasný priemyselný štandard pre koncové šifrovanie (E2EE). Hlavným cieľom práce je vývoj interaktívnej edukačnej aplikácie, ktorá slúži na objasnenie komplexných kryptografických mechanizmov a umožňuje používateľom pochopiť jednotlivé kroky protokolu prostredníctvom vizualizácie procesov. Práca kladie osobitný dôraz na aktuálne výzvy v oblasti kryptografie, pričom podrobne rozoberá a demonštruje postkvantové varianty protokolu (PQXDH, SPQR, TripleRatchet). Softvérová implementácia bola vytvorená v jazyku Python s využitím frameworku Flet. Výsledná aplikácia poskytuje modulárny nástroj na názornú výučbu základných aj pokročilých kryptografických konceptov.

Kľúčové slová

Signal protokol, Python, Flet, Double Ratchet, Postkvantová kryptografia

Abstract

This master's thesis focuses on the analysis and visualization of the Signal protocol, the current industry standard for end-to-end encryption (E2EE). The primary objective of the work is to develop an interactive educational application designed to demystify complex cryptographic mechanisms and enable users to understand the protocol's operations through process-oriented visualization. The thesis places special emphasis on contemporary cryptographic challenges, providing a detailed discussion and demonstration of post-quantum secure variants, such as PQXDH, the SPQR and the TripleRatchet mechanisms. The software implementation was developed using Python and the Flet framework. The resulting application serves as a modular tool for the illustrative teaching of both fundamental and advanced cryptographic concepts.

Keywords

Signal protocol, Python, Flet, Double Ratchet, Post-quantum cryptography

Poďakovanie

Touto cestou by som sa chcel poďakovať svojmu vedúcemu práce Mgr. Tomášovi Fabšičovi, PhD. za odbornú podporu pri vypracovaní diplomovej práce, cenné pripomienky, užitočné rady, ako aj za jeho neustálu pomoc a usmernenie počas celého procesu jej tvorby.

Obsah

Zoznam značiek a skratiek

E2EE – e2e šifrovanie, z angl. *End-to-End Encryption*

CT – šifrovaný text, z angl. *Ciphertext*

pk – verejný kľúč, z angl. *Public Key*

sk – súkromný kľúč, z angl. *Secret/Private Key*

SK – zdieľaný kľúč, z angl. *Shared Key*

DH – Diffieho-Hellmanova výmena kľúčov, z angl. *Diffie-Hellman*

EC – eliptická krivka, z angl. *Elliptic Curve*

ECDH – Diffieho-Hellmanova výmena kľúčov na eliptických krivkách, z angl. *Elliptic Curve Diffie-Hellman*

ECC – kryptografia na eliptických krivkách, z angl. *Elliptic Curve Cryptography*

KDF – funkcia odvodenia kľúča, z angl. *Key Derivation Function*

CBC – reťazenie šifrových blokov, z angl. *Cipher Block Chaining*

AEAD – autentizované šifrovanie s pridanými údajmi, z angl. *Authenticated Encryption with Associated Data*

MAC – autentifikačný kód správy, z angl. *Message Authentication Code*

HMAC – autentifikačný kód správy založený na hašovaní, z angl. *Hash-based Message Authentication Code*

HKDF – funkcia odvodenia kľúča založená na HMAC, z angl. *HMAC-based Key Derivation Function*

PKI – infraštruktúra verejných kľúčov, z angl. *Public Key Infrastructure*

FS – dopredná bezpečnosť, z angl. *Forward Secrecy*

PFS – dokonalá dopredná bezpečnosť, z angl. *Perfect Forward Secrecy*

PCS – bezpečnosť po kompromitácii, z angl. *Post-Compromise Security*

mod – operácia modulo (zvyšok po delení), z angl. *Modulo*

DLP – problém diskrétneho logaritmu, z angl. *Discrete Logarithm Problem*

DHP – Diffieho-Hellmanov problém, z angl. *Diffie-Hellman Problem*

IKM – vstupný kľúčový materiál, z angl. *Input Keying Material*

OKM – výstupný kľúčový materiál, z angl. *Output Keying Material*

RFC – dokument so žiadosťou o komentáre, z angl. *Request for Comments*

AES – pokročilý štandard šifrovania, z angl. *Advanced Encryption Standard*

AES-GCM – pokročilý štandard šifrovania v režime Galois/Counter, z angl. *Advanced Encryption Standard – Galois/Counter Mode*

ctr – režim počítadla, z angl. *Counter Mode*

GMAC – Galoisov autentifikačný kód správy, z angl. *Galois Message Authentication Code*

AD – pridané údaje (asociované dáta), z angl. *Associated Data*

KEM – mechanizmus zapuzdrenia kľúča, z angl. *Key Encapsulation Mechanism*

ek – kľúč zapuzdrenia (šifrovací kľúč), z angl. *Encapsulation Key*

dk – kľúč odpuzdrenia (dešifrovací kľúč), z angl. *Decapsulation Key*

RSA – algoritmus verejného kľúča RSA, z angl. *Rivest-Shamir-Adleman*

ok – výstupný kľúč, z angl. *Output Key*

RK – koreňový kľúč, z angl. *Root Key*

ck – reťazový kľúč, z angl. *Chain Key*

CK_s – odosielací reťazový kľúč, z angl. *Sending Chain Key*

CK_r – prijímací reťazový kľúč, z angl. *Receiving Chain Key*

mk – kľúč správy, z angl. *Message Key*

SS – zdieľané tajomstvo, z angl. *Shared Secret*

encap – zapuzdrenie, z angl. *Encapsulation*

decap – odpuzdrenie, z angl. *Decapsulation*

X3DH – rozšírený trojitý Diffie-Hellman, z angl. *Extended Triple Diffie-Hellman*

PQ – postkvantový, z angl. *Post-Quantum*

PQXDH – postkvantový rozšírený Diffie-Hellman, z angl. *Post-Quantum Extended Diffie-Hellman*

IK – kľúč identity, z angl. *Identity Key*

SPK – podpísaný predbežný kľúč, z angl. *Signed Prekey*

OPK – jednorazový predbežný kľúč, z angl. *One-Time Prekey*

IK_x – kľúč identity strany X, z angl. *Identity Key of X*

SPK_x – podpísaný predbežný kľúč strany X, z angl. *Signed Prekey of X*

OPK_x – jednorazový predbežný kľúč strany X, z angl. *One-Time Prekey of X*

PQSPK – postkvantový podpísaný predbežný kľúč, z angl. *Post-Quantum Signed Prekey*

PQOPK – postkvantový jednorazový predbežný kľúč, z angl. *Post-Quantum One-Time Prekey*

$PQSPK_x$ – postkvantový podpísaný predbežný kľúč strany X, z angl. *Post-Quantum Signed Prekey of X*

$PQOPK_x$ – postkvantový jednorazový predbežný kľúč strany X, z angl. *Post-Quantum One-Time Prekey of X*

$PQPK_x$ – postkvantový verejný kľúč strany X, z angl. *Post-Quantum Public Key of X*

DH_{pub} – verejný kľúč Diffie-Hellman, z angl. *Diffie-Hellman Public Key*

DH_{priv} – súkromný kľúč Diffie-Hellman, z angl. *Diffie-Hellman Private Key*

DH_{pair} – pár kľúčov Diffie-Hellman, z angl. *Diffie-Hellman Key Pair*

ML-Kem – mechanizmus zapuzdrenia kľúča na báze modulových mriežok, z angl. *Module-Lattice-Based Key Encapsulation Mechanism*

CRQC – kvantový počítač relevantný pre kryptografiu, z angl. *Cryptographically Relevant Quantum Computer*

LWE – učenie s chybami, z angl. *Learning with Errors*

SPQR – riedka postkvantová rohatka, z angl. *Sparse Post-Quantum Ratchet*

SCKA – riedka kontinuálna dohoda na kľúči, z angl. *Sparse Continuous Key Agreement*

DR – double ratchet algoritmus, z angl. *Double Ratchet*

TR – triple ratchet algoritmus, z angl. *Triple Ratchet*

MVC – Model-View-Controller architektúra, z angl. *Model-View-Controller*

GUI – grafické používateľské rozhranie, z angl. *Graphical User Interface*

TLS – bezpečnosť transportnej vrstvy, z angl. *Transport Layer Security*

ot – otvorený text, z angl. *Plaintext*

pt – otvorený text, z angl. *Plaintext*

IT – informačné technológie, z angl. *Information Technology*

PQC – postkvantová kryptografia, z angl. *Post-Quantum Cryptography*

NIST – Národný inštitút štandardov a technológií, z angl. *National Institute of Standards and Technology*

Zoznam obrázkov

Zoznam výpisov kódov

Zoznam tabuliek

Úvod

V súčasnej digitálnej komunikácii je koncové šifrovanie (End-to-End Encryption) nevyhnutným predpokladom na zabezpečenie ochrany citlivých údajov prenášaných prostredníctvom sietí. Návrh moderného a bezpečného komunikačného protokolu sa však neobmedzuje iba na samotné šifrovanie prenášaných dát. Systém musí riešiť aj komplexnejšie problémy, akými sú spoľahlivá výmena kľúčov, zabezpečenie vlastnosti Perfect Forward Secrecy či ochrana pred budúcimi hrozbami, ktoré predstavujú kvantové počítače.

Odpoveď na tieto výzvy v súčasnosti predstavuje protokol Signal, z ktorého sa stal široko uznávaný a využívaný priemyselný štandard. Hoci je tento protokol postavený na pomerne jednoduchých a známych kryptografických primitívach, ich vzájomná integrácia vytvára vysoko komplexný systém. Pre budúcich informatikov je pochopenie týchto mechanizmov dôležité, samotné teoretické opisy však často nepostačujú na plné uchopenie širších súvislostí a fungovania v praxi.

Hlavným cieľom tejto diplomovej práce je preto vytvorenie interaktívnej edukačnej aplikácie, ktorá vizuálne a krok za krokom demonštruje fungovanie protokolu Signal. Práca predpokladá, že používateľ/čitateľ už ovláda základné kryptografické pojmy, preto sa ťažisko presúva na to, ako sú tieto primitíva zladené do jedného bezpečného celku. Okrem prezentácie základného fungovania sa aplikácia zameriava aj na to, ako klasický protokol Signal reaguje na prípadnú kompromitáciu kľúča, čím demonštruje takzvaný „samoopravný“ (self-healing) mechanizmus systému. Ďalej, v reakcii na najnovšie technologické výzvy, sa práca zaoberá aj skúmaním postkvantových inovácií tohto protokolu.

Úlohou výslednej edukačnej aplikácie je poskytnúť študentom praktickú pomôcku pri osvojovaní si fungovania protokolu, a tým prispieť k formovaniu prístupu k vývoju softvéru so zameraním na bezpečnosť.

1 Teoretická časť

Vzhľadom na to, že táto kapitola pokrýva rozsiahlu a komplexnú problematiku, v nasledujúcom texte stručne priblížime jej štruktúru a logickú nadväznosť jednotlivých tém.

V prvej časti kapitoly zadefinujeme pojem koncového šifrovania (End-to-End Encryption, E2EE) a predstavíme základné kryptografické operácie, ktoré sa v ňom využívajú. Následne sa budeme venovať dvom protokolom výmeny kľúčov používaným v rámci protokolu Signal, a to algoritmom X3DH a PQXDH, pričom poukážeme na ich hlavné rozdiely.

Nasledujúce podkapitoly sú zamerané na algoritmus Double Ratchet a jeho komponenty. Detailne v nich popíšeme jednotlivé kryptografické ratchety, z ktorých sa algoritmus skladá, ako aj samotný mechanizmus Double Ratchet – jeho fungovanie a spôsob, akým sa vyrovnáva so sieťovými oneskoreniami a doručením správ mimo poradia.

Ďalšie časti kapitoly sa zaoberajú novým postkvantovým ratchetom a mechanizmami, ktoré využíva. Rozoberieme ich fungovanie a identifikujeme paralely s klasickým algoritmom Double Ratchet. V poslednom bode kapitoly sa zameriame na kombináciu klasického a postkvantového ratchetu, ktorá je známa pod označením Triple Ratchet.

1.1 Šifrovanie typu end-to-end (End-to-end encryption, E2EE) a jeho bezpečnostný model

Šifrovanie typu end-to-end (E2EE) je model zabezpečenia komunikácie, ktorého hlavným cieľom je zabezpečiť, aby obsah správ bol prístupný a interpretovateľný len pre strany zapojené do komunikácie (koncové body)[1]. V modeli E2EE sú údaje šifrované na zariadení odosielateľa pred vstupom do siete a môžu byť dešifrované iba na zariadení (zariadeniach) príjemcu.

Počas komunikácie prechádza správa viacerými medziležiacimi sieťovými uzlami a vrstvami serverov, vrátane infraštruktúry poskytovateľa služieb. Žiadny z týchto prvkov však nemá schopnosť dekódovať prenášaný šifrovaný text. Architektúra E2EE poskytuje kryptografickú záruku, že ani poskytovateľ sieťových služieb, ani potenciálne kompromitovaný server, ani útočník odpočúvajúci sieť nemajú prístup k dôverným údajom [1].

Základné princípy bezpečnostného modelu E2EE sú nasledovné: procesy šifrovania a dešifrovania prebiehajú na koncových bodoch používateľov. Súkromné kľúče potrebné na dešifrovanie zostávajú po celý čas v zariadení používateľa a za žiadnych okolností ich poskytovateľ služby nikdy nezíska. Funkčnosť serverov je obmedzená na presmerovanie

alebo dočasné ukladanie šifrovaných paketov správ, známych aj ako šifrovaný text (ciphertext). Tieto servery však nemajú potrebné kľúče na interpretáciu obsahu paketov.

1.1.1 Všeobecný proces fungovania E2EE

Typický moderný systém E2EE je založený na nasledujúcej sekvencii logických krokov, ktoré zabezpečujú nadviazanie relácie a bezpečnú výmenu údajov:

1. **Generovanie a správa kľúčov (Key generation):** Každá zúčastnená strana (koncový bod) je zodpovedná za generovanie asymetrického páru kľúčov, ktorý pozostáva z verejného kľúča a súkromného kľúča. Verejný kľúč sa šíri ostatným stranám prostredníctvom dôveryhodného kanála, typicky infraštruktúry verejného kľúča (PKI), zatiaľ čo súkromný kľúč zostáva striktne v zariadení[2].

2. **Dohoda na kľúči (Key Agreement):** Začatie komunikácie medzi dvoma stranami je často sprevádzané implementáciou protokolu dohody o kľúči, ktorý je zvyčajne založený na Diffie-Hellman protokole nad eliptickou krivkou (ECDH). Protokol umožňuje stranám vypočítať zdieľané tajomstvo pomocou ich príslušných súkromných kľúčov a verejného kľúča druhej strany. Toto zdieľané tajomstvo je vytvorené bez toho, aby sa prejavilo v dátovom prevádzke.

3. **Odvodenie kľúča (Key Derivation):** Nie je bezpečné používať zdieľané tajomstvo vytvorené počas výmeny kľúčov priamo na kryptografické operácie. Namiesto toho sa používa funkcia odvodenia kľúča (KDF), ktorá odvodí jeden alebo viac kryptograficky silných a štatisticky nezávislých kľúčov relácie zo zdieľaného tajomstva. Vo väčšine prípadov sa na účely šifrovania autentizácie generujú samostatné kľúče [3].

4. **Autentizované šifrovanie (Authenticated encryption):** Odosielateľ používa schému autentizovaného šifrovania (napr. AEAD) na šifrovanie otvoreného textu, pričom relačný kľúč je generovaný funkciou KDF. Táto schéma zabezpečuje dôvernosť správy a generuje overovaciu značku (alebo MAC), ktorá overuje integritu a autentickosť správy.

5. **Prenos a dešifrovanie (Transmission and decryption):** Šifrovaný text a autentizačná značka sa prenášajú príjemcovi prostredníctvom serverov. Príjemca má tiež k dispozícii rovnaké zdieľané tajomstvo, čo je dôsledkom predtým vykonaného protokolu dohody o kľúči, čím je možné odvodiť rovnaké relačné kľúče. Tieto nástroje sa používajú na overenie platnosti autentizačnej značky a ak je overenie úspešné, správa sa dešifruje.

1.1.2 Záruky bezpečnosti E2EE

Robustná bezpečnosť poskytovaná šifrovaním typu end-to-end (E2EE) vyplýva z kombinácie niekoľkých základných kryptografických záruk:

- **Dôvernosť (Confidentiality):** Šifrovanie je proces, ktorý zabezpečuje, že obsah správy môže prečítať len určený príjemca (príjemcovia), ktorý disponuje príslušným kľúčom na jej dešifrovanie. V prípade, že by útočník monitoroval sieť alebo server, videl by len nečitateľný šifrovaný text.

- **Integrita (Integrity):** Autentizované šifrovanie (napr. AEAD) zabezpečuje integritu správy počas prenosu tým, že poskytuje mechanizmus na overenie pravosti správy a zabraňuje akýmkoľvek zmenám. V prípade, že útočník zmení aj len jediný bit v šifrovanom texte, autentizácia na strane príjemcu zlyhá a klient správu zahodí.

- **Autentizacia (Authentication):** Správa kľúčov je navrhnutá tak, aby zabezpečila, že strany zapojené do transakcie komunikujú s určenými príjemcami správy, za predpokladu, že výmena verejných kľúčov bola vykonaná bezpečným spôsobom.

- **Forward Secrecy (FS):** Táto záruka má v moderných šifrovacích schémach mimoriadny význam. Vybrané protokoly E2EE, akými sú napríklad protokoly Signal, implementujú FS na viacerých úrovniach. V prípade kompromitácie dlhodobých súkromných kľúčov účastníkov v budúcnosti je bezpečnosť minulej komunikácie zachovaná vďaka tomu, že v procese úvodnej dohody na kľúči boli použité krátkodobé (ephemerálne) kľúče. Samotný algoritmus Double Ratchet [4] následne poskytuje FS pre prípady kompromitácie kľúčov reťazca (chain keys) počas prebiehajúcej relácie. V dôsledku toho útočník nie je schopný spätne dešifrovať predchádzajúcu komunikáciu, pretože sa používajú krátkodobé kľúče, ktoré boli medzičasom zničené.

- **Post-Compromise Security (PCS):** V prípade kompromitácie tajomstiev zdieľaných medzi komunikujúcimi stranami boli vyvinuté pokročilé protokoly, ktoré zabezpečujú, že systém sa dokáže „vyliečiť“. To znamená, že po úspešnej kompromitácii budú budúce správy opäť bezpečné a dôverné, akonáhle sa strany v rámci protokolu dohodnú na nových zdieľaných tajomstvách [5].

1.2 Základné kryptografické primitíva používané v systémoch E2EE

Moderné protokoly E2EE, ako napríklad široko auditovaný protokol Signal, sú založené na dobre známych a dôkladne analyzovaných kryptografických stavebných blokoch, známych ako primitíva[1]. Tieto komponenty majú samostatne robustné matematické vlastnosti, avšak práve ich strategická kombinácia poskytuje zložité bezpečnostné záruky, ktoré ponúka E2EE.

1.2.1 Výmena kľúčov Diffie–Hellman

Výmena kľúčov Diffie–Hellman (DH) je základný kryptografický protokol, ktorý umožňuje dvom stranám vytvoriť zdieľané tajomstvo cez nezabezpečený kanál bez toho, aby sa samotné tajomstvo prenášalo cez sieť [6].

Princíp fungovania klasického protokolu DH (nad konečnými poľami) je nasledovný:

1. Strany (Alice a Bob) sa verejne dohodnú na veľkom prvočísle p a generátore g (ktorý je generátorom multiplikatívnej grupy modulo p).
2. Alice si zvolí náhodné, tajné číslo a, a Bob si zvolí náhodné, tajné číslo b.
3. Alice vypočíta svoj verejný kľúč: $A = g^a \pmod{p}$.
4. Bob vypočíta svoj verejný kľúč: $B = g^b \pmod{p}$.
5. Strany si navzájom pošlú svoje verejné kľúče (A a B) cez nezabezpečený kanál.
6. Alice z prijatej hodnoty B a svojho tajného a vypočíta spoločné tajomstvo: $s = B^a \pmod{p} = (g^b)^a \pmod{p} = g^{ab} \pmod{p}$.
7. Bob z prijatej hodnoty A a svojho tajného b vypočíta spoločné tajomstvo: $s = A^b \pmod{p} = (g^a)^b \pmod{p} = g^{ab} \pmod{p}$.

Je zrejmé, že obe strany majú prístup k rovnakému zdieľanému tajomstvu, označenému ako $s = g^{ab} \pmod{p}$. Bezpečnosť Diffieho-Hellmanovho protokolu je primárne založená na výpočtovej náročnosti Diffieho-Hellmanovho problému (DHP). Tento problém spočíva v tom, že ak sú dané verejné hodnoty $g^a \pmod{p}$ a $g^b \pmod{p}$, pre útočníka je prakticky nemožné vypočítať hodnotu zdieľaného tajomstva $s = g^{ab} \pmod{p}$ bez znalosti súkromných exponentov a alebo b.

DHP je úzko spojený s problémom diskrétneho logaritmu (DLP). DLP definuje náročnosť získania exponentu a z verejnej hodnoty $g^a \pmod{p}$. Pre veľké prvočísla p a

vhodne zvolené generátory g sa predpokladá, že efektívne riešenie DLP je prakticky nevykonateľné.

Existuje pritom jasný vzťah medzi týmito problémami: ak by bolo možné efektívne riešiť DLP, bolo by možné ľahko vyriešiť aj DH problém, pretože by stačilo vypočítať a a následne získať $g^{ab} = (g^b)^a \pmod{p}$. Naopak, riešenie DH problému neznamená automaticky, že vieme získať jednotlivé exponenty a alebo b, teda DH riešiteľné *neimplikuje* riešiteľnosť DLP. Tento rozdiel je kľúčový pre formálne chápanie bezpečnosti DH protokolu: jeho bezpečnosť je redukovateľná na obtiažnosť DLP, ale naopak nie je známe, že by riešenie DH umožnilo efektívne riešiť DLP [7].

Je pozoruhodné, že súčasné systémy E2EE takmer bez výnimky používajú variantu protokolu DH s eliptickou krivkou (ECDH). ECDH poskytuje rovnakú bezpečnosť s podstatne kratšou dĺžkou kľúča, čo vedie k značným úsporám výpočtového výkonu a šírky pásma, najmä na zariadeniach s obmedzenými zdrojmi (napr. mobilných zariadeniach) [8, 9].

1.2.2 Funkcia odvodenia kľúča (Key Derivation Function, KDF)

Zdieľané tajomstvo vyplývajúce z dohody o kľúči ECDH nie je vhodné na použitie ako priamy šifrovací kľúč. Je to spôsobené tým, že takéto zdielané tajomstva často nemajú potrebné štatistické vlastnosti, ako je rovnomerné rozloženie. V dôsledku toho sa jedno zdielané tajomstvo používa na generovanie viacerých kryptograficky nezávislých kľúčov na rôzne účely, vrátane šifrovania a autentizácie.

Túto funkciu vykonávajú funkcie odvodenia kľúča (KDF). KDF je deterministická pseudonáhodná funkcia, ktorá generuje jeden alebo viacero bezpečných kľúčov zo vstupného kľúčového materiálu (Input Keying Material, IKM) a voliteľných parametrov (napr. „salt“ a kontextové informácie) [3].

Najrozšírenejšou používanou KDF je HKDF (KDF založená na HMAC, RFC 5869 [10]), ktorá využíva primitív HMAC (Hash-based Message Authentication Code) založený na kryptografickej hašovacej funkcii. Proces HKDF sa skladá z dvoch odlišných krokov:

1. **Extrakcia (Extraction):** Proces generovania pseudonáhodného kľúča (pseudorandom key,PRK) s pevnou dĺžkou a silnou kryptografiou zahŕňa použitie vstupného kľúčového materiálu (IKM) a voliteľnej soli. Tento krok sa označuje ako „koncentrácia“ entropie vstupného tajomstva.

2. **Expanzia (Expand):** Systém je schopný generovať ľubovoľný počet a dĺžku výstupného kľúčového materiálu (output keying material, OKM) z PRK, ako aj informačný parameter opisujúci kontext.

Kľúčová funkcia KDF v protokoloch E2EE zabezpečuje kryptografickú separáciu kľúčov. Tým sa zabezpečuje absencia akéhokoľvek využiteľného matematického vzťahu medzi kľúčmi používanými na odlišné účely (napr. šifrovanie Alice-Bob, šifrovanie Bob-Alice alebo odvodenie budúcich kľúčov).

1.2.3 Autentizované šifrovanie s pridruženými údajmi (Authenticated Encryption with Associated Data, AEAD)

V moderných kryptografických systémoch už nestačí zabezpečiť len dôvernosť správ, musí byť zaručená aj integrita a autentickosť údajov, t. j. je potrebné overiť, či správa nebola počas prenosu (náhodne alebo úmyselne) pozmenená a či skutočne pochádza od predpokladaného odosielateľa.

V minulosti sa tieto dva ciele riešili oddelene, napríklad prostredníctvom implementácie šifrovania CBC a overovania HMAC. Tento prístup je však zložitý a náchylný na chyby (napr. nesprávna implementácia schémy „Encrypt-then-MAC“) [11].

Ako riešenie tohto problému sa navrhujú konštrukcie AEAD. AEAD je šifrovací systém, ktorý zabezpečuje dôvernosť, integritu a autentickosť v jednej operácii [12]. Dve bežné implementácie systémov AEAD sú:

1. **AES-GCM (Advanced Encryption Standard - Galois/Counter Mode)**:Uvažovaná bloková šifra využíva AES v režime počítadla (CTR) na účely šifrovania a GMAC založený na násobení nad Galoisovými poľami na účely autentizácie. Kapacita platformy na rýchle vykonávanie závisí od prítomnosti špecializovanej hardvérovej akcelerácie, ako je napríklad sada inštrukcií AES-NI [12].

2. **ChaCha20-Poly1305:** Moderná konštrukcia AEAD založená na streamovej šifre (RFC 8439). ChaCha20 vykonáva šifrovanie, zatiaľ čo Poly1305 vykonáva autentizáciu. Ponúka vynikajúci výkon v softvérových implementáciách, najmä na architektúrach bez hardvérovej podpory AES [13].

Súvisiace údaje (Associated Data, AD) sú definované ako informácie (napr. hlavičky, metadáta, čísla verzií protokolov), ktoré nevyžadujú šifrovanie, ale vyžadujú autentizáciu. AEAD zaručuje, že v prípade, ak útočník manipuluje s šifrovaným textom alebo súvisiacimi údajmi, overenie na strane príjemcu zlyhá. Schémy AEAD preto slúžia ako základný rámec na zabezpečenie bezpečnosti prenosu údajov v rámci systémov E2EE.

1.2.4 Mechanizmus zapuzdrenia kľúča (Key Encapsulation Mechanism, KEM)

Mechanizmus zapuzdrenia kľúča (Key Encapsulation Mechanism, KEM) predstavuje moderný kryptografický prístup určený na bezpečnú výmenu symetrického kľúčového

materiálu medzi dvoma stranami prostredníctvom asymetrickej kryptografie. V kontexte systémov E2EE slúži KEM ako robustná alternatíva alebo doplnok k tradičným metódam dohody na kľúči, akou je napríklad protokol Diffie-Hellman (DH).

Zatiaľ čo pri protokole DH obe strany aktívne prispievajú k vytvoreniu zdieľaného tajomstva vlastnými podielmi (výmenou verejných hodnôt), KEM je navrhnutý tak, že jedna strana (odosielateľ) priamo vygeneruje symetrický kľúč a následne ho bezpečne „zapuzdrí“ (zašifruje) pomocou verejného kľúča príjemcu. Tento prístup je matematicky čistejší a eliminuje mnohé zraniteľnosti spojené s priamym šifrovaním kľúčov pomocou tradičných asymetrických algoritmov (napr. útoky na výplň, tzv. padding oracles, v algoritme RSA) [14].

Formálne sa mechanizmus KEM skladá z troch základných pravdepodobnostných algoritmov:

1. **Generovanie kľúčov (Key Generation, KeyGen):** Algoritmus, ktorý vytvorí asymetrický pár kľúčov pozostávajúci z verejného kľúča (pk) a súkromného kľúča (sk):

   $$(pk, sk) \leftarrow \text{KeyGen}()$$

   Verejný kľúč pk je šírený prostredníctvom dôveryhodného kanála (napr. PKI), zatiaľ čo súkromný kľúč sk zostáva striktne chránený v zariadení príjemcu.

2. **Zapuzdrenie (Encapsulation, Encap):** Odosielateľ použije verejný kľúč príjemcu na vygenerovanie silného, náhodného symetrického kľúča K a k nemu prislúchajúceho šifrovaného textu c (zapuzdreného kľúča):

   $$(K, c) \leftarrow \text{Encap}(pk)$$

   Šifrovaný text c sa následne prenesie cez nezabezpečenú sieť k príjemcovi. Samotný kľúč K sa nikdy neprenáša.

3. **Odpuzdrenie (Decapsulation, Decap):** Príjemca po prijatí šifrovaného textu c použije svoj súkromný kľúč sk na extrakciu pôvodného symetrického kľúča K:

   $$K \leftarrow \text{Decap}(sk, c)$$

Ak nedošlo k modifikácii šifrovaného textu c počas prenosu a operácia je úspešná, zaručuje sa, že extrahovaný kľúč sa identicky zhoduje s kľúčom vygenerovaným odosielateľom. Rovnako ako v prípade DH protokolu platí, že výsledný symetrický kľúč K sa spravidla nepoužíva priamo na šifrovanie obsahu, ale slúži ako vstupný kľúčový materiál (IKM) pre funkciu odvodenia kľúča (KDF), čím sa odvodia finálne relačné kľúče pre schému AEAD.

1.3 Úvod do protokolov X3DH a PQXDH

Kľúčovým bodom systémov šifrovania typu end-to-end (E2EE) je implementácia bezpečnej výmeny kľúčov medzi stranami v asynchrónnom prostredí, kde komunikujúce strany nie sú nutne online v rovnakom čase. Protokol Signal bol vyvinutý s cieľom riešiť tento problém pomocou hierarchického kľúčového systému a protokolu Extended Triple Diffie-Hellman (X3DH) v spojení s jeho postkvantovým rozšírením Post-Quantum Extended Diffie-Hellman (PQXDH). Táto kapitola je venovaná úvodu do klasického X3DH a jeho moderného rozšírenia, PQXDH, pričom sa primárne opiera o oficiálnu dokumentáciu protokolu Signal[15, 16].

1.3.1 Počiatočná dohoda o kľúči: X3DH

Cieľom protokolu X3DH je bezpečne a asynchrónne vytvoriť počiatočné zdieľané tajomstvo (SK) s vysokou entropiou. V kontexte komunikačného procesu Alica ako iniciátorka odošle správu Bobovi, ktorý preberá úlohu respondenta.

Typy kľúčov a hierarchia kľúčov Protokol Signal používa kryptografické kľúče s rôznou životnosťou a účelom, aby zmiernil riziko. Systém je založený na ECDH protokole na výmenu kľúča využívajúcom eliptickú krivku Curve25519. Kľúče používané v systéme je možné rozdeliť do 3 kategórií:[17]

1. **Identity Key Pair (IK):** Ide o najdlhšie trvajúci statický pár kľúčov, ktorý sa používa na autentizáciu digitálnej identity používateľa. Verejná časť (IK_{Pub}), ktorá je registrovaná na serveri, slúži ako základ pre verifikáciu uvedeného „bezpečnostného čísla“ (Safety Number) [1].

2. **Signed Prekey Pair (SPK):** Kľúč so strednou životnosťou, ktorý bol digitálne podpísaný pomocou súkromného identifikačného kľúča. Tento jav má dvojakú úlohu:

   (a) zodpovedá za autentifikáciu SPK voči klientovi

   (b) umožňuje neaktívnej strane zúčastniť sa na počiatočnej výmene Diffie-Hellman

   SPK sa zvyčajne menia každý týždeň alebo každý mesiac [15].

[1] Bezpečnostné číslo je 60-miestny kód reprezentujúci odtlačok verejných kľúčov identity oboch strán. Slúži na manuálnu verifikáciu integrity kanála a ochranu pred útokmi typu Man-in-the-Middle. V aplikácii Signal je túto verifikáciu možné vykonať manuálnym vizuálnym porovnaním číselného kódu alebo pohodlnejšie naskenovaním QR kódu z obrazovky druhého používateľa[18].

3. **One-Time Prekeys (OPK):**Termín „jednorazové predkľúče“ (OPK) sa vzťahuje na špecifický typ digitálneho kľúča, ktorý je určený na použitie pri jednej príležitosti. Na rozdiel od opakovane použiteľných kľúčov, ktoré možno použiť viackrát, jednorazové predkľúče sú navrhnuté na jedno použitie a po jednom použití sa majú odstrániť. Pri registrácii klient nahraje na server podstatný počet týchto verejných kľúčov. Pri inicializácii novej konverzácie poskytne server volajúcej strane jeden z dostupných OPK a následne ho zo svojej databázy odstráni. Ich jednorazové použitie a okamžité zmazanie po nadviazaní relácie garantujú, že aj v prípade kompromitácie iných kľúčov (napríklad strednodobého SPK) zostáva počiatočná komunikácia v bezpečí. Perfektná dopredná bezpečnosť (PFS) neustálej komunikácie sa však dosahuje až následnou algoritmickou rotáciou kľúčov pomocou algoritmov.

Zverejnenie Bobových kľúčov Aby s ním mohla Alica začať komunikovať, Bob najprv nahrá na server svoj balík kľúčov (Prekey bundle), ktorý obsahuje nasledujúce kľúče eliptickej krivky (ECDH):

- Identifikačný kľúč $\mathbf{IK_B}$ („Identity Key“).
- Podpísaný predbežný kľúč $\mathbf{SPK_B}$ („Signed Prekey“) a jeho digitálny podpis na overenie pravosti.
- Sekvencia jednorazových predbežných kľúčov $\mathbf{OPK_B}$ („One-Time Prekeys“).

Iniciatíva Alice a výpočet zdieľaného tajomstva Alica získa Bobov balík kľúčov zo servera. Následne vygeneruje svoj vlastný dočasný (efemérny) pár kľúčov, označený ako $\mathbf{EK_A}$. Spoločný tajný kľúč (SK) sa odvodí kombináciou štyroch (alebo troch) operácií Diffie-Hellman (DH):

$$DH1 = DH(IK_A, SPK_B)$$
$$DH2 = DH(EK_A, IK_B)$$
$$DH3 = DH(EK_A, SPK_B)$$
$$DH4 = DH(EK_A, OPK_B)$$

Komponenty $DH1$ a $DH2$ slúžia na vzájomnú autentizáciu, ich bezpečnosť však závisí od nekompromitovania dlhodobých kľúčov (IK). $DH3$ poskytuje doprednú bezpečnosť (forward secrecy) pomocou dočasného a podpísaného predbežného kľúča. $DH4$ pridáva dodatočnú vrstvu bezpečnosti prostredníctvom jednorazového predbežného kľúča (OPK).

Konečný kľúč sa odvodí pomocou funkcie **HKDF** (funkcia odvodenia kľúča založená na HMAC). Protokol je flexibilný a prispôsobuje sa dostupnosti jednorazových kľúčov:

a) Štandardný postup so 4 DH (s prítomným $\mathbf{OPK_B}$):

$$SK = KDF(\ DH1 \ ||\ DH2 \ ||\ DH3 \ ||\ DH4)$$

b) Riešenie absencie $\mathbf{OPK_B}$ (náhradný mechanizmus s 3 DH): Ak je sada OPK na serveri vyčerpaná, komponent $DH4$ sa vynechá:

$$SK = KDF(\ DH1 \ ||\ DH2 \ ||\ DH3)$$

Tento postup stále zabezpečuje šifrovanie, avšak neposkytuje FS pre počiatočnú správu kvôli dlhšej životnosti SPK_B. Kompromitácia súkromného kľúča SPK_B by mohla viesť k spätnému dešifrovaniu.

Po odvodení SK Alica zostaví počiatočnú správu pre Boba, ktorá obsahuje jej verejné kľúče (IK_A, EK_A), identifikátory Bobových použitých predkľúčov a samotný šifrovaný text chránený pomocou AEAD schémy. Alica následne vymaže svoj dočasný súkromný kľúč EK_A a výsledky DH výpočtov.

Prijatie správy Bobom Bob prijme správu a na základe identifikátorov zistí, ktoré z jeho predkľúčov Alica použila. Načíta svoje príslušné súkromné kľúče a zopakuje operácie $DH1$ až $DH4$ (prípadne len po $DH3$) s využitím Aliciných verejných kľúčov priložených v správe.

Následne Bob aplikuje rovnakú funkciu KDF na odvodenie zdieľaného tajomstva SK. Týmto kľúčom overí autenticitu a dešifruje počiatočný šifrovaný text. Po úspešnom ukončení protokolu Bob bezpečne vymaže svoj použitý jednorazový súkromný kľúč OPK_B, aby zabránil jeho opätovnému použitiu.

1.3.2 Kvantovo odolná ochrana: Post-Quantum Extended Diffie--Hellman (PQXDH)

Protokol PQXDH stavia na základoch protokolu X3DH, pričom klasický ECDH integruje s postkvantovým (PQ) mechanizmom zapuzdrenia kľúča (KEM). Vzniká tak hybridné riešenie, pre ktoré Signal využíva algoritmus CRYSTALS-Kyber-1024.

Postkvantové typy kľúčov a ich úloha Protokol PQXDH je v porovnaní s tradičným X3DH doplnený o nové, postkvantovo bezpečné kľúče, ktoré slúžia pre mechanizmus hybridnej výmeny kľúčov. Tieto kľúče sú založené na algoritme CRYSTALS-Kyber (ML-KEM) a možno ich rozdeliť do nasledujúcich kategórií:

1. **Signed Last-Resort PQKEM Prekey (PQSPK):** Ide o digitálne podpísaný, postkvantový kľúč „poslednej šance“, ktorý má stredne dlhú životnosť. Podobne ako podpísaný predkľúč (SPK) založený na ECC, aj tento umožňuje asynchrónny

prenos správ v prípade, že sa na serveri minú jednorazové postkvantové kľúče (PQOPK); protokol sa vtedy spolieha na tento kľúč, aby zachoval ochranu odolnú voči kvantovým počítačom. Kľúč je podpísaný identifikačným kľúčom používateľa (IK), čo klientovi zaručuje jeho integritu a pôvod.

2. **Signed One-Time PQKEM Prekeys (PQOPK):** Termín „postkvantové jednorazové predkľúče“ (PQOPK) označuje sadu digitálnych kľúčov, ktoré sú navrhnuté na jedno použitie pri spustení novej relácie (Session). Server ich poskytuje iniciátorovi po jednom, pričom po nadviazaní spojenia sa kľúč vymaže, čo zaručuje doprednú bezpečnosť (FS) aj voči budúcemu kvantovému útoku. Na rozdiel od OPK v PQXDH klient pred nahraním podpíše kľúče PQOPK svojím identifikačným kľúčom, čím zabráni neoprávnenému zásahu do výmeny kľúčov.

Zverejnenie Bobových kľúčov Podobne ako pri klasickom X3DH, Bob nahráva svoj balík kľúčov na server. Tento balík obsahuje už popísané kľúče eliptickej krivky (IK_B, SPK_B, OPK_B), avšak je navyše rozšírený o kľúče PQ KEM:

- **Podpísaný pre-key poslednej inštancie $\mathbf{PQSPK_B}$** vrátane jeho podpisu: $\mathbf{Sig(IK_B, PQSPK_B, Z_{PQSPK})}$
- **Sekvenciu podpísaných jednorazových predkľúčov $\mathbf{PQOPK_B}$** a ich jedinečné podpisy vytvorené pomocou $\mathbf{IK_B}$: $\mathbf{Sig(IK_B, PQOPK_{Bn}, Z_n)}$

Hodnoty **Z** sú 64-bajtové náhodné sekvencie (nonce), ktoré slúžia na kryptografické oddelenie podpisov.

Iniciatíva Alice a výpočet zdieľaného tajomstva Alica získa rozšírený balík, overí podpisy a vygeneruje dočasný pár kľúčov EK_A presne tak, ako v štandardnom protokole. Zásadným rozdielom je pridaný krok, v ktorom Alica vytvorí zapuzdrenie PQKEM pre postkvantový kľúč prijatý od Boba, označený ako $PQPK_B$. Je potrebné poznamenať, že tento kľúč môže byť buď jednorazový predkľúč ($PQOPK_n$), alebo podpísaný predkľúč ($PQSPK_B$). Kľúč $PQSPK_B$ sa použije ako záložné riešenie v situácii, keď server vyčerpá všetky Bobove jednorazové predkľúče ($PQOPK_n$) a Bob zatiaľ nestihol nahrať nové:

$$(CT, SS) = \text{PQKEM-ENC}(PQPK_B)$$

Kde CT je ciphertext a SS je postkvantové zdieľané tajomstvo. Výpočet konečného zdieľaného kľúča SK prebieha analogicky s výpočtami v X3DH, avšak funkcia KDF je na konci rozšírená o komponent SS:

a) Ak je v pakete OPK_B (4 DH):

$$SK = KDF(\ DH1 \ ||\ DH2 \ ||\ DH3 \ ||\ DH4 \ ||\ SS)$$

b) Ak v pakete nie je OPK_B (3 DH):

$$SK = KDF(\ DH1 \ ||\ DH2 \ ||\ DH3 \ ||\ SS)$$

Alica pred šifrovaním navyše vypočíta sekvenciu asociovaných dátových bajtov (AD), ktorá obsahuje identifikačné informácie pre obe strany:

$$AD = EncodeEC(IK_A) \ ||\ EncodeEC(IK_B)$$

Inicializačná správa odoslaná Bobovi je štrukturálne rovnaká ako pri X3DH, no navyše obsahuje zapuzdrený kľúč CT (výstup z PQKEM). Zdieľané tajomstvo SS sa po odvodení SK vymaže spolu s ostatnými dočasnými dátami.

Prijatie správy Bobom Postup prijatia správy je identický s X3DH, s jedným podstatným medzikrokom pred samotným výpočtom zdieľaného kľúča. Bob pomocou identifikátorov zistí použité kľúče, načíta svoje súkromné kľúče a navyše musí dekapsulovať postkvantové tajomstvo pomocou súkromného kľúča prislúchajúceho k $PQPK_B$:

$$SS = \text{PQKEM-DEC}(PQPK_B, CT)$$

S dekapsulovaným tajomstvom SS Bob zopakuje výpočet kľúča SK podľa vzorcov z predchádzajúcej podkapitoly, vypočíta asociované dáta AD a pokúsi sa dešifrovať počiatočný šifrovaný text. Pri úspešnom ukončení Bob bezpečne vymaže všetky použité jednorazové súkromné kľúče (klasické OPK_B aj postkvantové $PQOPK_n$).

1.4 KDF reťazce

1.4.1 Úloha a štruktúra reťazcov KDF

Základná štruktúra algoritmu Double Ratchet sa skladá z reťazcov KDF (Key Derivation Function Chains), ako je podrobne opísané v oficiálnej dokumentácii protokolu Signal[4]. Zatiaľ čo samotná funkcia odvodenia kľúča (KDF) predstavuje iba základný kryptografický stavebný blok (primitívum), reťazce KDF definujú systém a štrukturálne usporiadanie, ktoré toto primitívum iteratívne využíva. Práve toto usporiadanie umožňuje bezpečné a nepretržité obnovovanie kľúčov počas celej komunikačnej relácie.

Reťazec KDF je definovaný ako iteratívny proces, v ktorom sa časť výstupu funkcie odvodenia kľúča používa ako „tajomstvo“ na generovanie nasledujúceho prvku reťazca.

Krok reťazca KDF je možné opísať pomocou nasledujúcich parametrov:

Vstup:

- Tajný a náhodný kľúč KDF: **Chain Key** (CK).
- Arbitrárne vstupné údaje (*input data*).

Výstup: Funkcia produkuje dvojicu kľúčov, ktoré sú výpočtovo nerozlišiteľné od náhodných hodnôt.:

$$(OK, CK_{next}) = \text{KDF}(CK, \text{Vstup})$$

- **Výstupný kľúč (OK):** Kľúč používaný na skutočnú úlohu (napr. šifrovanie správy).
- **Ďalší reťazový kľúč (CK_{next}):** Vstupný kľúč pre ďalší krok v reťazci.

Tento rekurzívny mechanizmus zabezpečuje priebežnú aktualizáciu stavu protokolu a zároveň garantuje dodržanie týchto kľúčových bezpečnostných vlastností:

- **Odolnosť**(Resilience): Výstupné kľúče sa javia ako náhodné pre každého útočníka, ktorý nevlastní reťazový kľúč. Táto vlastnosť zostáva zachovaná aj v prípade, že útočník je schopný ovplyvniť vstupné údaje KDF.
- **Forward Secrecy:** Vzhľadom na správanie KDF ako jednosmernej funkcie nie je možné odvodiť predchádzajúce výstupné kľúče z reťazového kľúča získaného v neskoršom bode reťazca. Toto opatrenie je implementované s cieľom zabezpečiť, aby potenciálna kompromitácia reťazového kľúča neohrozila dôvernosť minulých správ.
- **Obnova po narušení**(Break-in Recovery): V prípade, že vstupné údaje reťazca KDF obsahujú novú entropiu, ktorá je útočníkovi neznáma, reťazec je schopný „samoliečenia". V dôsledku toho je obnovená bezpečnosť budúcich výstupných kľúčov.

1.4.2 Synchronizácia reťazca v komunikácii

V implementácii Double Ratchet každá komunikujúca strana udržiava tri paralelné reťazce KDF: **Root Chain**, **Sending Chain** a **Receiving Chain**.

Porozumenie synchronizácie medzi stranami je nevyhnutné pre správne fungovanie systému. V bilaterálnej komunikácii (napr. medzi Alicou a Bobom) sú reťazce strán zrkadlovými obrazmi jeden druhého:

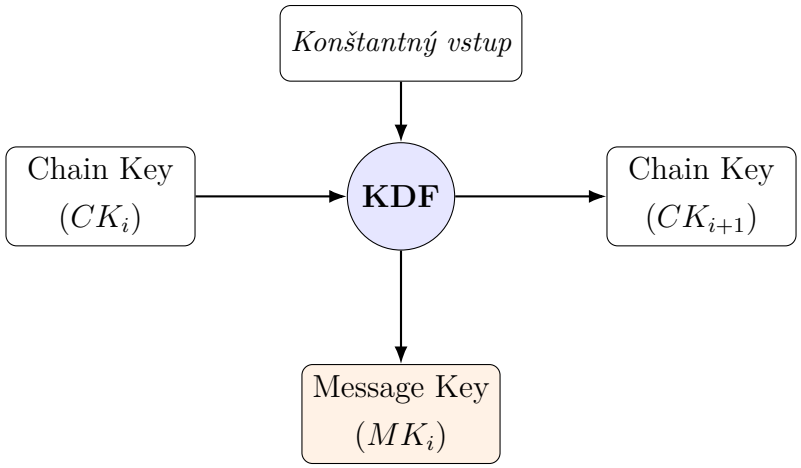
1. *Sending Chain* Alice je identická s *Receiving Chain* Boba.
2. Naopak, *Sending Chain* Boba je identický s *Receiving Chain* Alice.

1.5 Symmetric-key Ratchet

Zatiaľ čo reťazce KDF poskytujú základnú štruktúru, symetrický kľúč ratchet je mechanizmus, ktorý skutočne posúva reťazce odosielania a prijímania pre každú odoslanú alebo prijatú správu. Táto zložka je zodpovedná za generovanie kľúčov správ (message keys),je v tejto sekcii podrobne preskúmaný na základe oficiálnej špecifikácie protokolu Signal [4].

1.5.1 Prevádzkový mechanizmus

Základný princíp symetrickej račny je založený na využití konštantnej hodnoty ako vstupu reťazca KDF. Ak v tejto fáze nie je vo vstupoch reťazca prítomný nový zdroj entropie (napríklad výstup z Diffie-Hellman výmeny), proces je deterministický vzhľadom na aktuálny kľúč reťazca. Logika tohto procesu je znázornená na obrázku 1.



Obr. 1: Mechanizmus krokovania symetrického kľúča Ratchet.

Jednotlivé fázy kľúčovania (*krok*) prebiehajú nasledovne:

- **Vstup:** Aktuálny reťazový kľúč (CK) a konštantný KDF vstup.
- **Transformácia:** Systém aplikuje funkciu KDF.
- **Výstup:**
  - *Kľúč správy (MK):* Pre šifrovanie obsahu správy pomocou AEAD.
  - *Ďalší reťazový kľúč (CK_{next}):* Prepíše predchádzajúci reťazový kľúč.

1.5.2 Bezpečnostné vlastnosti a vymazanie kľúča

Symetrická ratchet je primárne určená na zabezpečenie doprednej bezpečnosti na úrovni jednotlivých správ. Ak by totiž protivník v určitom momente získal aktuálny reťazový

kľúč, nebol by schopný dešifrovať predchádzajúcu komunikáciu. Funkcia odvodzovania kľúčov (KDF) je jednosmerná, čo technicky znemožňuje proces označovaný ako „vrátenie reťaze“ (rolling back the chain), a teda nemožno z nej odvodiť predtým odstránené kľúče správ.

> *Dôležité:* Po použití kľúča správy musí byť tento kľúč okamžite vymazaný z pamäte. Rovnako sa po každom posunutí ratchetu (t. j. po kroku, kedy sa z aktuálneho reťazového kľúča odvodí nový reťazový kľúč a kľúč pre správu) predchádzajúci reťazový kľúč trvalo odstráni.

Symetrická ratchet sama o sebe nezabezpečuje bezpečnosť po kompromitácii, pretože nepridáva novú náhodnosť. Táto nevýhoda je odstránená ratchetom Diffie-Hellman.

1.5.3 Podpora asynchrónnosti

Konštrukcia symetrickej ratchet poskytuje technický základ pre riešenie oneskorení a neusporiadaných správ, ktoré sú bežné v sieťovej komunikácii. Keďže v KDF reťazcoch nie je možné ísť späť, prijímajúca strana musí pri prijatí správy mimo poradia (napríklad s vyšším poradovým číslom) svoj prijímací reťazec „vopred posunúť dopredu“ (pre-roll).

Prijímateľ tak postupne odvodí všetky kľúče pre chýbajúce správy až po kľúč potrebný pre aktuálne prijatú správu. Kľúče vygenerované pre tieto preskočené správy si dočasne uloží do pamäte a doručenú správu plynulo dešifruje. Keď oneskorená chýbajúca správa neskôr dorazí, systém ju dešifruje pomocou príslušného vopred uloženého kľúča. Aby sa však striktne zachovala dopredná bezpečnosť protokolu, tento kľúč sa ihneď po svojom jednorazovom použití (dešifrovaní správy) okamžite a natrvalo vymaže z pamäte.

Tento proces sa deje výhradne na úrovni prijímacieho reťazca (receiving chain). Neovplyvňuje tak stav nadradeného koreňového reťazca (root chain), z ktorého sú odosielací (sending) a prijímací reťazec odvodené, vďaka čomu zostáva celková štruktúra protokolu synchronizovaná aj pri výpadkoch správ.

1.6 Diffie-Hellman Ratchet

Druhým a kľúčovým pilierom „dvojitej“ povahy algoritmu Double Ratchet je Diffie-Hellman Ratchet. Ako už bolo spomenuté, symetrický ratchet je zodpovedný za zabezpečenie doprednej bezpečnosti v prípade kompromitácie reťazového kľúča. Naopak, DH ratchet umožňuje obnovenie bezpečnosti po kompromitácii (PCS). V tejto podkapitole spracované s využitím oficiálnej dokumentácie ako hlavného zdroja [4].

Táto zložka je navrhnutá tak, aby v prípade kompromitácie aktuálneho stavu strán (získaním tajných kľúčov) útočníkom systém dokázal „sa sám vyliečiť“. Dôvernosť sa

obnoví po krátkom, dočasnom období, počas ktorého si strany opäť vymieňajú správy, pričom útočník je vylúčený z budúcej komunikácie.

1.6.1 Teoretické základy a injekcie entropie

Vzhľadom na deterministickú povahu symetrickej kryptografie by útočník po kompromitácii reťazového kľúča teoreticky mohol vypočítať všetky budúce kľúče. Aby sa tomu zabránilo, je nevyhnutné, aby systém do procesu zaviedol externú, novú náhodnosť (entropiu).

DH ratchet na dosiahnutie tohto cieľa využíva nepretržité používanie výmeny kľúčov Diffie-Hellman (typicky ECDH). Strany počas komunikácie nepoužívajú statické kľúče, ale nepretržite generujú nové páry kľúčov (DH_{pair}) a prenášajú verejné kľúče (DH_{pub}) v hlavičke správy (*header*).

Keď strana dostane nový verejný kľúč od svojho partnera, vykoná operáciu DH so svojím aktuálnym súkromným kľúčom a prijatým verejným kľúčom. Výsledné zdieľané tajomstvo slúži ako nový zdroj entropie.

1.6.2 Koreňový reťazec a integrácia KDF

DH ratchet pracuje priamo na koreňovom reťazci. Tento reťazec možno považovať za „chrbtovú kosť“("backbone") protokolu. Na rozdiel od symetrických reťazcov, pod ktorými sa v kontexte tohto protokolu rozumejú odosielací a prijímací reťazec a ktoré pracujú s konštantným vstupom, koreňový reťazec prijíma ako vstup v každom kroku výsledok operácií DH.

Matematicky možno tento proces rozložiť na nasledujúce kroky, keď strana (napr. Bob) dostane nový kľúč DH_{pub}:

1. **Výpočet výstupu DH:** Bob vykoná výpočet:

$$DH_{out} = \text{DH}(\text{Bob}_{priv}, \text{Alice}_{pub})$$

2. **Aplikácia KDF (KDF_RK):** Systém aplikuje funkciu Root KDF, ktorá prijíma dva vstupy: predchádzajúci koreňový kľúč (RK_{old}) a nový výstup DH (DH_{out}).

$$(RK_{new}, CK_{new}) = \text{KDF\_RK}(RK_{old}, DH_{out})$$

Funkcia generuje dva nové kryptograficky silné kľúče:

- RK_{new} (**Nový koreňový kľúč**): Tento kľúč zabezpečuje kontinuitu koreňového reťazca pre ďalší krok DH.

- CK_{new} (**Nový Chain Key**): Tento kľúč inicializuje úplne nový reťazec *Sending* alebo *Receiving* pre symetrický ratchet.

Táto hierarchická štruktúra zabezpečuje, že výsledok výmeny DH „presakuje“ do reťazcov odosielania správ. Keďže CK_{new} nemožno vypočítať bez znalosti DH_{out}, nové kľúče správ budú nezávislé od predchádzajúcich.

1.6.3 Mechanizmus „ping-pong“ a prechody stavov

V praxi fungovanie DH ratchet vykazuje akési striedanie „ping-pong“ alebo „krok za krokom“ vyvolané výmenou správ. Proces je asynchrónny, čo znamená, že strany neaktualizujú súčasne, ale v reakcii na odpovede.

Podrobný popis procesu na príklade Alice a Boba:

Počiatočný stav: Alice má Bobov verejný kľúč (B_{pub0}) a Bob má Alicin verejný kľúč (A_{pub0}).

Alice odosiela: Alice vygeneruje nový pár kľúčov (A_1).

- Vykoná krok DH (medzi svojím novým kľúčom A_{priv1} a Bobovým verejným kľúčom B_{pub0}).
- Aktualizuje koreňový reťazec (čím priamo mení svoj pôvodný koreňový kľúč).
- Vytvorí nový odosielací reťazec.
- Odošle správu, ktorej súčasťou je jej nový verejný kľúč A_{pub1}.

Na ďalší posun v koreňovom reťazci a vytvorenie nového prijímacieho reťazca už musí čakať na Bobovu odpoveď obsahujúcu jeho nový verejný kľúč.

Bob prijíma (krok DH 1): Bob prijíma správu s A_{pub1}. Keďže ide o nový kľúč:

- Vykoná krok DH (medzi B_{priv0} a A_{pub1}).
- Aktualizuje koreňový reťazec.
- Vytvorí nový *prijímací reťazec* na prijímanie správ od Alice.

Bob odpovie (krok DH 2): Bob vygeneruje nový pár kľúčov (B_1).

- Vykoná druhý krok DH (teraz medzi novým kľúčom A_{pub1} a svojím novým kľúčom B_{priv1}).
- Vytvorí nový *reťazec odosielania*.
- Odosiela odpoveď s verejným kľúčom B_{pub1}.

Alice prijíma: Alice prijíma B_{pub1} a tiež vykonáva príslušné kroky DH, synchronizujúc sa s Bobom.

Tento mechanizmus zabezpečuje, že po každej kompletnej výmene správ sú šifrovacie kľúče matematicky úplne obnovené.

1.7 Algoritmus Double Ratchet

1.7.1 Architektúra a fungovanie

Kryptografické konštrukcie podrobne opísané v predchádzajúcich častiach – reťazce KDF, symetrické ratchety a ratchety Diffie-Hellman – sú samy osebe iba stavebnými blokmi. Inovácia algoritmu Double Ratchet spočíva v integrácii týchto komponentov do jedného koherentného stavového automatu. Názov protokolu odkazuje na dva úzko spolupracujúce mechanizmy:

- **Symetrická ratchet** „klikne“ raz za každú správu, čím zabezpečí jedinečné kľúče pre správy a doprednu bezpečnosť komunikacie.
- **Diffie-Hellmanova (DH) ratchet** „klikne“ pri každom prepnutí (keď sa smer komunikácie obráti a príde nový kľúč), čím zabezpečí vloženie novej entropie a obnovenie bezpečnosti po narušení.

1.7.2 Stav integrovaného systému

Počas prevádzky Double Ratchet si strany (Alice a Bob) udržiavajú synchronizovaný stav. Stav danej strany (P) obsahuje nasledujúce hlavné premenné:

DH_{pair}**:** Vlastný aktuálny pár kľúčov Diffie-Hellman strany.

DH_{remote}**:** Najnovšie prijatý verejný kľúč partnera.

RK **(koreňový kľúč(Root Key):** Aktuálny 32-bajtový kľúč koreňového reťazca.

CK_s, CK_r**:** Aktuálne kľúče odosielacieho a prijímajúceho reťazca.

N_s, N_r**:** Počítadlá odoslaných a prijatých správ v aktuálnych reťazcoch.

1.7.3 Ratchet Flow

Algoritmus je riadený udalosťami: odoslanie alebo prijatie správy spustí zmenu stavu.

Odosielanie Keď Alice odošle správu Bobovi, proces ovplyvní iba Symmetric Ratchet (pokiaľ práve neprebieha inicializácia alebo prepínanie DH).

1. **Symetrický krok:** Alice vykoná krok na *reťazci odosielania* s kľúčom CK_s.
   - Vygeneruje nový kľúč správy (MK).
   - Aktualizuje CK_s na nasledujúcu hodnotu.
2. **Zostavenie hlavičky:** Vytvorí hlavičku obsahujúcu jej aktuálny verejný kľúč ($DH_{.pub}$), číslo reťazca (N_s) a dĺžku predchádzajúceho reťazca (PN).
3. **Šifrovanie:** Zašifruje správu pomocou MK (AEAD).
4. **Zvýšenie počítadla:** Zvýši hodnotu N_s o jeden.

Tento proces sa opakuje, kým Alice nepretržite odosiela správy bez toho, aby dostala odpoveď. Každá správa dostane nový kľúč, ale pár kľúčov DH a koreňový reťazec zostávajú nezmenené.

Prijímanie správ Keď Alice prijme správu od Boba, skontroluje hlavičku, aby sa rozhodla, ktorý mechanizmus ratchet aktivovať.

A) Iba symetrický krok (v rámci rovnakého reťazca): Ak verejný kľúč v hlavičke zodpovedá už uloženému kľúču DH_{remote}, znamená to, že Bob ešte nezmenil „smer“, ale len poslal ďalšiu správu v sekvencii.

- Alice vykonáva symetrické kroky na *Receiving Chain*, kým nedosiahne poradové číslo (N) prijatého správu.

B) DH Ratchet krok (Začatie nového reťazca): Ak sa verejný kľúč v hlavičke líši od uloženého DH_{remote}, znamená to „zmenu éry“. Bob vygeneroval nový pár kľúčov a odpovedal. V tomto bode sa vykoná nasledujúca zložitá sekvencia:

1. **Uzavretie reťazca KDF:** Ak v predchádzajúcom reťazci chýbajú správy (na základe hodnoty hlavičky PN), Alica pre ne najskôr vygeneruje a bezpečne uloží kľúče správ (message keys). To jej umožní tieto správy úspešne dešifrovať neskôr, keď dorazia oneskorene (*Out-of-Order* spracovanie).
2. **Operácia DH Ratchet:**
   - Vypočíta tajný kľúč DH: $DH_{out} = \mathrm{DH}(DH_{pair}.priv, \mathrm{Header}.pub)$.
   - Aktualizuje koreňový reťazec: $(RK, CK_r) = \mathrm{KDF\_RK}(RK, DH_{out})$.

Tým sa vytvorí nový prijímajúci reťazec (CK_r), pomocou ktorého Alica spracuje nové správy.

3. **Aktualizácia kľúčového páru:** Alice vygeneruje nový súkromný DH_{pair} pre svoju ďalšiu odpoveď a vykonáva druhý krok DH na koreňovom reťazci, čím vytvorí nový reťazec odosielateľa (CK_s).

4. **Spracovanie správy:** Nakoniec odvodí kľúč správy z nového CK_r pomocou symetrického ratchetu a dešifruje správu.

1.7.4 Súhrn bezpečnostných vlastnosti

Vysoká miera bezpečnosti protokolu Double Ratchet je dosiahnutá modulárnou kombináciou viacerých kryptografických mechanizmov:

Hrozba	**Obranný mechanizmus**	**Vysvetlenie**
Dešifrovanie minulých správ	Symetric Ratchet (Forward Secrecy)	Okamžité vymazanie reťazových kľúčov a kľúčov správ zabraňuje retroaktívnemu prístupu.
Dešifrovanie budúcich správ	DH Ratchet (Post-Compromise Security)	Ak sú kľúče ukradnuté, útočník môže čítať len do ďalšieho prepnutia DH. Nová výmena kľúčov DH útočníka vylúči.
Manipulácia so správami	Šifrovanie AEAD	Autentifikované šifrovanie zaručuje integritu.
Výmena/vyhadzovanie správ	Sekvencie a reťazce	Vďaka prísnej štruktúre reťazca je možné manipuláciu s poradím okamžite odhaliť alebo opraviť.

1.8 Spracovanie neusporiadaných a oneskorených správ

Podpora pre nespoľahlivé sieťové vrstvy bola základnou požiadavkou pri návrhu algoritmu Double Ratchet, čo táto podkapitola analyzuje s odvolaním sa na oficiálnu špecifikáciu[4]. Moderné systémy zasielania správ často fungujú asynchrónne, pričom nie je zaručené, že správy dorazia k príjemcovi v poradí, v akom boli odoslané, alebo že vôbec dorazia. Spracovanie „nesprávne zoradených správ“ zabezpečuje, že stav protokolu nebude ohrozený, ak neskoršia správa dorazí skôr ako skoršia.

1.8.1 Úloha hlavičky v synchronizácii

Kľúčom k vyriešeniu problému sú metadáta. Hlavička každej šifrovanej správy obsahuje dve kritické celé čísla, ktoré umožňujú príjemcovi presne umiestniť správu do kryptografického reťazca:

- N **- číslo správy:** Poradové číslo správy v aktuálnom *reťazci odosielania/prijímania*. Prvá správa má číslo 0, druhá 1 atď. Toto číslo sa na začiatku každého nového reťazca (v kroku DH ratchet) vynuluje.
- PN **- dĺžka predchádzajúceho reťazca:** koľko správ bolo odoslaných v predchádzajúcom reťazci pred kolom DH).

S touto informáciou môže prijímajúca strana deterministický určiť, koľko krokov ratchet musí vykonať na svojich reťazcoch, aby dosiahla kľúč pre prijatú správu.

1.8.2 Mechanizmus „preskočených kľúčov správ“

Keď je správa prijatá, systém skontroluje, či zapadá do očakávanej sekvencie. Existujú dva možné prípady:

Synchrónny prípad: Hodnota prijatej správy N presne zodpovedá ďalšiemu očakávanému poradovému číslu prijímajúceho reťazca. V tomto prípade spracovanie pokračuje normálne.

Asynchrónny (medzera) prípad: Hodnota prichádzajúcej správy N je väčšia ako očakávané poradové číslo alebo správa patrí do nového reťazca DH, pričom v predchádzajúcom reťazci sú stále nespracované prvky (označené hodnotou PN).

V tejto situácii protokol používa stratégiu **„fast-forward“**:

1. **Krok reťazca:** Systém začne spúšťať symetrický krok (kroky KDF) pre chýbajúce indexy.
2. **Ukladanie kľúčov:** Pre každý „preskočený“ krok systém vygeneruje príslušný kľúč správy, ale namiesto toho, aby ho zahodil, uloží ho do dočasného úložiska (zoznamu alebo hash mapy) spolu s indexom správy.
3. **Aktualizácia stavu:** Kľúč reťazca sa priebežne prepisuje nasledujúcou hodnotou, aby stav reťazca „dohnal“ prijatú správu.
4. **Spracovanie aktuálnej správy:** Akonáhle reťaz dosiahne požadovaný stav, vygeneruje sa kľúč pre prijatú správu a vykoná sa dešifrovanie.

Príklad fungovania Predpokladajme, že Bob čaká na správu číslo 3 ($N = 3$), ale kvôli oneskoreniu siete najskôr dorazí správa číslo 5 ($N = 5$).

1. Systém zistí „medzeru“ medzi správami 3 a 4.
2. Vyvolá funkciu KDF s aktuálnym kľúčom reťazca, aby vygeneroval MK_3. Uloží ho do úložiska *SkippedKeys*. Kľúč reťazca sa aktualizuje.
3. Funkcia KDF sa vyvolá znova a vygeneruje MK_4. Ten sa tiež uloží. Kľúč reťazca sa aktualizuje.
4. Funkcia KDF je volaná tretíkrát, čím sa vytvorí MK_5. S týmto kľúčom je možné správu dešifrovať a zobraziť používateľovi.

Keď neskôr (s oneskorením) dorazí tretia správa, systém najskôr vyhľadá kľúč v úložisku *SkippedKeys*. Keďže ho nájde, môže ho okamžite dešifrovať bez vykonania akýchkoľvek KDF operácií.

1.8.3 Bezpečnostné a implementačné aspekty

Počas procesu sa ukladajú iba konečné *kľúče správ*. Medziprodukty *reťazové kľúče* sa nikdy neukladajú, pretože by mohli byť použité na odvodenie následných kľúčov, čo by porušilo zásadu Forward Secrecy.

- **Prahové hodnoty pre vymazanie(Deletion Threshold):** Uložené, ale nepoužité kľúče predstavujú bezpečnostné riziko. Implementácie ukladajú obmedzenia:
  - *Časové obmedzenie(Time limit):* Napr. ak sa kľúč nepoužíva X dní, je vymazaný.
  - *Množstevný limit(Quantity limit):* Napr. maximálne 2000 krokov môže byť posunutých dopredu, aby sa zabránilo útokom typu Denial-of-Service (DoS).
- **Usporiadané dešifrovanie:** Aplikačná vrstva (UI) je zodpovedná za zobrazenie oneskorených správ používateľovi v správnom poradí.

1.9 ML-KEM Braid Protokoll

Jednou z kľúčových otázok v modernom kryptografickom výskume je integrácia postkvantovej (PQ) bezpečnosti do existujúcich komunikačných kanálov. Protokol **ML-KEM Braid** reaguje na túto výzvu, pričom podrobnosti jeho fungovania v tejto časti čerpáme primárne z oficiálnej technickej dokumentácie[19]. Tento mechanizmus je takzvaný

mechanizmus *Sparse Continuous Key Agreement* (SCKA), ktorý používa algoritmus ML-KEM (predtým Kyber) štandardizovaný organizáciou NIST.

Hlavným cieľom protokolu je generovať sériu postkvantových bezpečných zdieľaných tajomstiev pre dve strany. Tieto tajomstvá zabezpečujú Forward Secrecy a bezpečnosť po kompromitácii (PCS).

1.9.1 Sparse Continuous Key Agreement (SCKA)

Termín „Sparse“ („riedky“) odkazuje na skutočnosť, že nove zdieľané tajomstvo (*SS*) nie je generované v každom cykle. Veľké kľúče a šifrované texty sú rozdelené na časti a prenášané vo viacerých správach pomocou samoopravných kódov(erasure codes).

Protokol SCKA generuje prísne usporiadanú sekvenciu zdieľaných tajomstiev. Pojem **epocha** (alebo identifikátor epochy) označuje poradovú pozíciu konkrétneho zdieľaného tajomstva v tejto sekvencii, pričom je reprezentovaný nezáporným celým číslom [19]. Každá epocha je tak spätá s unikátnym zdieľaným tajomstvom, ktoré zostáva v platnosti, kým protokol neinicializuje prechod do nasledujúcej epochy v poradí.

1.9.2 Obmedzenia veľkosti: ML-KEM vs. ECDH

Priame („naivné“) použitie klasického ML-KEM v moderných systémoch zasielania správ naráža na vážne prekážky kvôli požiadavkám na šírku pásma. Porovnajme veľkosti kľúčov s aktuálne populárnymi riešeniami založenými na eliptických krivkách:

- **ECDH:**
  - Verejný kľúč: 32 bajtov(Toto je jediný údaj, ktorý sa pri výmene kľúčov posiela po sieti. Obe strany odošlú len 32 bajtov).
- **ML-KEM-768:**
  - Zapuzdrovací kľúč(Encapsulation key): 1184 bajtov.
  - Šifrovaný text: 1088 bajtov.

Je zrejmé, že správy ML-KEM sú o niekoľko rádov väčšie. V prípade protokolu typu „ping-pong“, ktorý generuje nový kľúč pre každú výmenu správ (napríklad klasický Double Ratchet), by tento nárast veľkosti spôsobil neprijateľné oneskorenia a nárast prenesených dát. To odôvodňuje špeciálnu štruktúru ML-KEM Braid.

1.9.3 Rola erasure kódov(Erasure Codes)

Kľúčovým prvkom protokolu je erasure code, ktorý sa používa na zvýšenie odolnosti siete a na zabezpečenie spoľahlivého doručenia dát aj v prípade straty častí prenosu.

Vymazávacie kódy (napr. Reed-Solomon) umožňujú odosielateľovi rozdeliť pôvodnú správu na N častí a vygenerovať M redundantných častí. Príjemca môže rekonštruovať správu bez strát, aj keď prijme ľubovoľných N z $N + M$ odoslaných častí. Tým sa eliminuje potreba pomalých požiadaviek na opätovné odoslanie na úrovni siete.

1.9.4 Paralelizácia a fragmentácia dát

Ako už bolo spomenuté, priame použitie klasického ML-KEM by kvôli obrovskej veľkosti kľúčov spôsobilo v asynchrónnom komunikačnom protokole neprijateľné zdržania. ML-KEM Braid tento problém rieši pomocou inkrementálneho rozhrania a prísnej fragmentácie. Veľké kryptografické objekty sú rozdelené na menšie časti a prenášané postupne vo viacerých správach.

Zásadnou výhodou tohto prístupu je, že odpovedajúca strana nepotrebuje čakať na prijatie celého zapuzdrovacieho kľúča (EK) na to, aby mohla začať generovať a odosielať svoju odpoveď. Umožňuje to najmä oddelenie krátkej hlavičky. Po prijatí tejto hlavičky môže strana spustiť proces zapuzdrenia a začať obojsmerný paralelný prenos najväčších dátových celkov. Konkrétne rozdelenie komponentov a ich priradenie k sekvenčnému alebo paralelnému režimu prenosu detailne popisuje tabuľka 1.

Komponent	**Odosielateľ**	**Režim prenosu a paralelizácia**
ek_header	Iniciátor	**Sekvenčne**. Umožňuje respondentovi predčasne začať generovať ct_1.
ek_vector	Iniciátor	**Paralelne**. Najväčšia časť zapuzdrovacieho kľúča (EK), odosiela sa súčasne s prijímaním ct_1.
ct_1	Respondent	**Paralelne**. Najväčšia časť šifrovaného textu (CT), odosiela sa súčasne s prijímaním *ek_vector*.
ct_2	Respondent	**Sekvenčne**. Odosiela sa až po skompletizovaní ct_1. Obsahuje dáta potrebné na finálnu dekapsuláciu tajomstva.

Tabuľka 1: Rozdelenie a paralelizácia prenosu dát v protokole ML-KEM Braid

1.9.5 Stavy v ML-KEM Braid protokole

Protokol SCKA pracuje ako deterministický stavový automat. Proces neustáleho generovania zdieľaných tajomstiev prebieha v epochách, pričom kľúčovým mechanizmom je

striedanie rolí (*Role Reversal*). Strana, ktorá v aktuálnej epoche vystupuje ako Iniciátor (odosiela zapuzdrovací kľúč), sa v nasledujúcej automaticky stáva Respondentom (generuje šifrovaný text).

Pre lepšiu zrozumiteľnosť môžeme životný cyklus jednej epochy z hľadiska logiky protokolu rozdeliť do troch konceptuálnych fáz. Tieto fázy predstavujú logický nadstavec nad detailnými stavmi implementovaného automatu:

1. **Iniciácia (Odosielanie EK):** Iniciátor vygeneruje kľúčový pár (EK, DK) a postupne odosiela fragmenty EK.
2. **Reakcia (Prijímanie EK / Odosielanie CT):** Respondent po prijatí hlavičky generuje zdieľané tajomstvo a spätne odosiela fragmenty šifrovaného textu CT.
3. **Finalizácia (Prijímanie CT a dekapsulácia):** Iniciátor skompletizuje CT, vykoná dekapsuláciu pomocou DK a obe strany prechádzajú do novej epochy s obrátenými rolami.

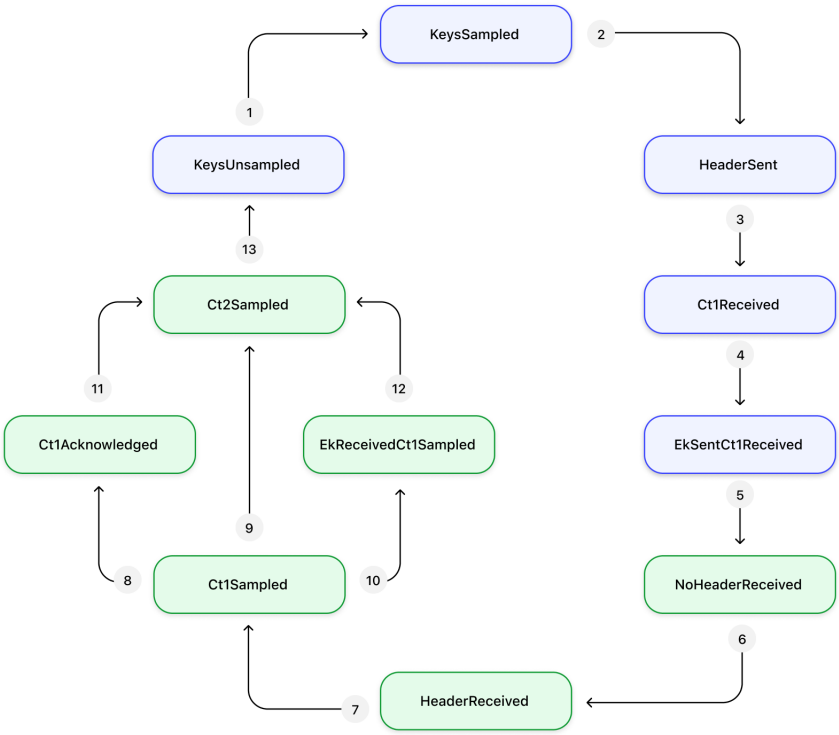
Samotná implementácia protokolu však vyžaduje jemnejšie delenie stavov kvôli asynchrónnej povahe a fragmentácii dát. Tieto technické stavy a prechody medzi nimi sú znázornené na Obrázku 2. Modrou farbou sú vyznačené stavy prislúchajúce roli Iniciátora, zatiaľ čo zelená farba reprezentuje stavy Respondenta.

`KeysUnsampled`: Počiatočný stav. Agent je pripravený vygenerovať úplne nový pár kľúčov KEM pri ďalšej udalosti odosielania. V tomto stave ignoruje všetky prichádzajúce správy. Pri odoslaní vygeneruje kľúčový pár, začne odosielať hlavičku a prechádza do stavu `KeysSampled`.

`KeysSampled`: Agent úspešne vygeneroval kľúčový pár a nachádza sa vo fáze postupného odosielania fragmentov hlavičky (*chunks*). Vo svojom stave si uchováva dekapsulačný kľúč (`dk`) a vektorovú časť (`ek_vector`). Akonáhle prijme správu typu `Ct1`, s istotou vie, že druhá strana prijala celú hlavičku, a prechádza do stavu `HeaderSent`.

`HeaderSent`: Agent dokončil prenos hlavičky. V tomto stave začína prenášať fragmenty kľúča `ek_vector` a zároveň prijíma prvé fragmenty šifrovaného textu (`ct1`). Keď úspešne prijme a dekóduje celé `ct1`, prechádza do stavu `Ct1Received`.

`Ct1Received`: Agent má už kompletne prijaté `ct1`, no proces odosielania fragmentov `ek_vector` z jeho strany ešte prebieha. Vo svojich správach zároveň posiela potvrdenia o prijatí (`EkCt1Ack`). Keď obdrží správu typu `Ct2`, vie, že druhá strana úspešne prijala jeho `ek_vector`, a prechádza do stavu `EkSentCt1Received`.



Obr. 2: Stavový automat aktéra v protokole SCKA [19]

`EkSentCt1Received`: Agent kompletne odoslal `ek_vector`, prijal `ct1` a momentálne len pasívne prijíma zostávajúce fragmenty `ct2`. Neodosiela už žiadne dáta. Keď je `ct2` kompletne prijaté, agent overí MAC, dekapsuluje zdialane tajomstvo (`ss`), exportuje výstupný kľúč a prechádza do stavu `NoHeaderReceived`, čím sa pripravuje na začiatok ďalšej epochy v obrátenej role.

Nasledujúce stavy opisujú účastníka (respondenta), ktorý prijíma zapuzdrovací kľúč a ako odpoveď generuje a odosiela šifrovaný text.

`NoHeaderReceived`: Agent očakáva a postupne prijíma fragmenty hlavičky novej epochy. Neodosiela žiadne dáta. Po úspešnom zložení hlavičky a overení jej integrity (MAC) extrahuje zárodok kľúča (`ek_seed`) a prechádza do stavu `HeaderReceived`.

`HeaderReceived`: Hlavička je kompletne prijatá. Pri najbližšej udalosti odoslania agent vygeneruje zdieľané tajomstvo a prvú časť šifrovaného textu (`ct1`). Okamžite získava a exportuje zdieľané tajomstvo na ďalšie použitie. Následne odošle prvý fragment `ct1` a prechádza do stavu `Ct1Sampled`.

Ct1Sampled: Tento stav zahŕňa najkomplexnejšie vetvenie. Agent odosiela fragmenty `ct1` a súbežne prijíma fragmenty `ek_vector`. Vývoj situácie závisí od poradia sieťových udalostí:

- Ak prijme celý `ek_vector` skôr, než dostane potvrdenie o prijatí `ct1`, prejde do `EkReceivedCt1Sampled`.
- Ak dostane potvrdenie o prijatí `ct1` skôr, než má skompletizovaný `ek_vector`, prejde do `Ct1Acknowledged`.
- Ak v jedinom prijímacom cykle získa posledný fragment `ek_vector` aj potvrdenie, vypočíta `ct2` a prechádza priamo do `Ct2Sampled`.

EkReceivedCt1Sampled: Agent úspešne prijal celý zapuzdrovací kľúč, no stále odosiela `ct1` a čaká na potvrdenie (`EkCt1Ack`). Keď toto potvrdenie dorazí, agent vypočíta finálne `ct2` a prechádza do stavu `Ct2Sampled`.

Ct1Acknowledged: Situácia je opačná k predchádzajúcej: agent dokončil odosielanie `ct1` a vie, že bolo doručené, no stále nedisponuje kompletným kľúčom `ek_vector`. Po jeho kompletnom stiahnutí a overení integrity agent vypočíta `ct2` a prechádza do stavu `Ct2Sampled`.

Ct2Sampled: Agent dokončil odosielanie `ct1`, plne prijal `ek_vector` a momentálne odosiela posledné fragmenty `ct2`. Zároveň čaká na príchod správy s identifikátorom nasledujúcej epochy (`epoch + 1`). Keď ju obdrží, znamená to, že epocha sa úspešne skončila, prichádza k výmene rolí (Role Reversal) a agent prechádza do počiatočného stavu `KeysUnsampled`.

1.10 Sparse Post-Quantum Ratchet (SPQR)

V predchádzajúcej kapitole opísaný protokol SCKA slúži na to, aby zúčastnené strany po určitom počte asynchrónnych výmen správ získali nové zdieľané tajomstvo. Mechanizmus Sparse Post-Quantum Ratchet (SPQR) je postavený na tomto protokole analogickým spôsobom, akým je algoritmus Double Ratchet (DR) postavený na výmene kľúčov Diffie-Hellman. Mechanizmus SPQR technicky zabezpečuje, že každá jednotlivá správa je šifrovaná iným kľúčom, čím sa garantuje Forward Secrecy (FS). Zároveň prostredníctvom aplikácie nového zdieľaného tajomstva úspešne obnovuje bezpečnosť po kompromitácii (PCS). Informácie v tejto kapitole sú spracované s využitím oficiálnej technickej dokumentácie ako hlavného zdroja [20].

Ako už z názvu vyplýva, tento mechanizmus využíva proces ratchetovania rovnako ako mechanizmus DR. Nachádzajú sa tu dva reťazce správ (*message chains*), ktoré tvoria

symetrický ratchet plniaci presne rovnakú funkciu ako v kontexte DR. Odosielací reťazec (*sending chain*) jednej strany priamo korešponduje s prijímacím reťazcom (*receiving chain*) druhej strany a naopak. Ku každému reťazcu prislúcha inkrementálne počítadlo, čím sa zachováva rovnaká štruktúra ako pri Double Ratchet.

Okrem symetrických reťazcov systém operuje s koreňovým kľúčom (RK). Tento kľúč sa priebežne kombinuje s novým zdieľaným tajomstvom, ktoré poskytuje protokol SCKA. Výsledkom tejto kombinácie je odvodenie nového koreňového kľúča a inicializácia úplne nových kľúčov pre odosielací a prijímací reťazec. Vytvorenie nových reťazcov predstavuje začiatok novej *epochy*. Jednotlivé epochy sú striktne zoradené a rozlišujú sa na základe svojho poradového čísla.

Zavedenie epoch a ich identifikátorov je nevyhnutné, pretože na rozdiel od klasického DR nie je možné začať používať novovytvorené reťazce okamžite po ich odvodení. Druhá strana totiž v danom momente ešte nemusí disponovať všetkými fragmentmi dát potrebnými na získanie nového zdieľaného tajomstva, a teda nedokáže paralelne odvodiť nové reťazce na svojej strane. Z tohto dôvodu musí protokol SCKA, ktorý spravuje asynchrónny stav oboch aktérov, poskytovať mechanizmu SPQR presnú spätnú väzbu o tom, reťazce ktorej konkrétnej epochy sa majú použiť na šifrovanie alebo dešifrovanie danej správy. Na dosiahnutie tejto synchronizácie vyžaduje protokol SCKA implementáciu nasledujúcich rozhraní[20]:

- `SCKASend(state)` → `(msg, sending_epoch, output_key)`
  - `msg`: Nepriehľadné (opaque) dátové fragmenty určené pre samotný protokol SCKA.
  - `sending_epoch`: Najnovšia epocha, o ktorej je kryptograficky zaručené, že ju prijímateľ bude po spracovaní parametra `msg` poznať.
  - `output_key`: Nadobúda hodnotu `None` alebo `(key_epoch, key)`, pričom `key_epoch` identifikuje epochu, ku ktorej získaný kľúč (`key`) patrí.
- `SCKAReceive(state, msg)` → `(receiving_epoch, output_key)`
  - `receiving_epoch`: Epocha, ktorá bola odosielateľom nastavená ako `sending_epoch` v čase generovania danej správy.
  - `output_key`: Nadobúda hodnotu `None` alebo `(key_epoch, key)`, analogicky k procesu odosielania.

Hoci oficiálna špecifikácia označuje jeden z parametrov ako `msg`, v tomto špecifickom kontexte to nereprezentuje samotný zašifrovaný text používateľskej komunikácie (payload). Ide o metadáta a kryptografické údaje obsiahnuté v hlavičke správy (napríklad

indikátor typu správy, fragmenty EK alebo fragmenty CT), ktoré sú nevyhnutné pre správne fungovanie protokolu SCKA.

Jedným z výstupných parametrov je identifikačné číslo epochy (`sending_epoch` alebo `receiving_epoch`). Tento parameter presne definuje, ku ktorej epoche patrí reťazec, z ktorého sa má odvodiť kľúč správy (Message Key) na jej bezpečné zašifrovanie alebo dešifrovanie. Ďalším výstupným parametrom je `output_key`, čo je v praxi postkvantové zdieľané tajomstvo pochádzajúce z protokolu SCKA. Toto tajomstvo sa následne posúva do algoritmu SPQR na odvodenie spomínaných nových reťazcov. Ako bolo detailne opísané v predchádzajúcej sekcii, účastník môže získať nové zdieľané tajomstvo buď pri prijímaní, alebo pri odosielaní dát v závislosti od toho, či v aktuálnej epoche plní funkciu iniciátora alebo respondenta (pričom tieto roly sa neustále striedajú).

1.10.1 Úloha SCKA (Sparse Continuous Key Agreement)

Logickým základom vrstvy SPQR je princíp Continuous Key Agreement (CKA), ktorý bol prispôsobený postkvantovému prostrediu. V podstate strany počas komunikácie nepretržite generujú a vymieňajú nové kľúče KEM (*Shared Secrets*) asynchrónne. SCKA slúži aj ako abstrakčná vrstva, ktorá skrýva zložitosť konkrétnej implementácie KEM (napr. enkapsuláciu, dekapsuláciu) pred hlavným protokolom.

Mechanizmus SPQR je abstraktne navrhnutý tak, aby dokázal pracovať ako nadstavba pre ľubovoľný protokol SCKA. Najznámejším praktickým príkladom takéhoto nasadenia je práve ML-KEM Braid, aktuálne využívaný v protokole Signal. V teoretickej rovine je však implementácia SPQR mechanizmu plne kompatibilná s akýmkoľvek iným SCKA protokolom, za predpokladu, že bezchybne podporuje definované programové rozhrania.

1.11 Triple Ratchet Protokoll

Hoci praktická implementácia kvantových počítačov relevantných pre kryptografiu (cryptographically relevant quantum computers - CRQC) ešte nie je hotová, táto hrozba je už dnes relevantná vďaka stratégii útoku „Harvest Now, Decrypt Later“ („Zbieraj teraz, dešifruj neskôr“). Podstatou tejto stratégie je, že útočníci zaznamenávajú aktuálny šifrovaný dátový tok, aby ho mohli dešifrovať o niekoľko rokov neskôr, keď budú mať k dispozícii vhodnú technológiu.

Najnovší vývoj algoritmu Double Ratchet, **Triple Ratchet**, ponúka riešenie tohto problému, pričom všetky technické detaily sú prevzaté z jeho oficiálnej dokumentácie[4]. Protokol používa hybridný prístup, ktorý kombinuje rýchlosť, malú veľkosť a desaťročia

overenú bezpečnosť klasických metód s odolnou, ale často náročnejšou na zdroje ochranou algoritmov postkvantovej kryptografie (PQC).

1.11.1 Synergia PQXDH a Triple Ratchet

Často kladenou otázkou v literatúre a v technickom dizajne je, prečo, ak už používame postkvantovú metódu v počiatočnej fáze komunikácie (*Initial Key Agreement*) – napríklad PQXDH (*Post-Quantum Extended Diffie-Hellman*) – prečo je potrebné zaviesť Triple Ratchet v neskoršej fáze relácie? Odpoveď spočíva v časovom predĺžení bezpečnostných záruk a koncepcii „Break-in Recovery“ (obnova po narušení).

Oba protokoly hrajú počas životného cyklu odlišné, ale komplementárne úlohy:

PQXDH (inicializácia): Zodpovedá za nadviazanie relácie. Zabezpečuje, že počiatočné zdieľané tajomstvo dohodnuté medzi stranami je chránené pred útokmi typu „Harvest Now, Decrypt Later“, aj keď útočník disponuje kvantovým počítačom. Ak by však mechanizmus ratchet používal iba klasické (nekvantovo zabezpečené) algoritmy po spustení PQXDH, kompromitácia zariadenia v neskoršom čase by bola fatálna.

Triple Ratchet (relácia): Zodpovedá za nepretržitú ochranu relácie. Schopnosť „samoliečenia“ klasického Double Ratchet je založená na probléme diskrétneho logaritmu, ktorý je možné prelomiť pomocou kvantového počítača. To by znamenalo, že budúci útočník, ktorý získa stav zariadenia, by mohol dešifrovať nielen aktuálne správy, ale aj všetky budúce správy, pretože by bol schopný matematicky zvrátiť „liečivé“ kroky ratchet (ktoré majú útočníka zablokovať).

Zavedenie Triple Ratchet je preto nevyhnutné na zabezpečenie obnovy po kvantovom prelomení. Tak aj v prípade, že sú kľúče zariadenia kompromitované, systém sa môže vrátiť do bezpečného stavu odolného voči kvantovým počítačom.

1.11.2 Technická integrácia

PQXDH je „predchodcom“ protokolu v systéme. Úspešné vykonanie PQXDH vedie k kryptograficky silnej hodnote SK (*Shared Key*), ktorá slúži ako vstup pre inicializáciu Triple Ratchet. Tento SK sa používa na odvodenie hodnôt Triple Ratchet *Root Key* a počiatočných hodnôt *Chain Key*, čím sa zabezpečuje kontinuita medzi fázami handshake a výmeny správ.

1.11.3 Architektúra hybridného modelu

Názov Triple Ratchet odkazuje na skutočnosť, že architektúra protokolu nezahŕňa nahradenie jediného mechanizmu, ale skôr paralelné vykonanie dvoch matematicky

nezávislých ratchetových systémov a zlúčenie ich výstupov. Dve hlavné zložky systému sú:

- **Tradičný Double Ratchet (EC):** Mechanizmus založený na eliptických krivkách (napr. Curve25519) opísaný v predchádzajúcich častiach. Jeho úlohou je umožniť extrémne rýchlu a efektívnu výmenu kľúčov a chrániť pred útočníkmi s tradičným výpočtovým výkonom. Táto vrstva má nízku latenciu a malú veľkosť metadát.
- **Sparse Post-Quantum Ratchet (PQ):** Vnáša do systému postkvantovú odolnosť. SPQR nie je samostatný kryptografický algoritmus, ale protokol vyššej úrovne, ktorý využíva mechanizmus Sparse Continuous Key Agreement (SCKA) na generovanie zdieľaných tajomstiev.

Termín „Sparse“ v názve SPQR vyplýva z technickej nevyhnutnosti. SPQR nemusí nutne „kliknúť“ pri každej výmene správ, hustota (*frekvencia*) je parametrizovateľná, čo umožňuje rovnováhu medzi požiadavkami na šírku pásma a frekvenciou aktualizácií zabezpečenia (napr. výmena kľúčov PQC sa uskutočňuje len každých N správ).

1.11.4 Kombinácia kľúčov a ochrana hĺbky

Najkritickejším bodom Triple Ratchet je bezpečná kombinácia kľúčov z dvoch nezávislých systémov. Cieľom je zabezpečiť, aby bezpečnosť konečného výsledku bola aspoň taká silná ako bezpečnosť silnejšej z dvoch komponentov. Premenné stavu systému sú rozdelené do dvoch samostatných častí:

- ec_{state}: Stav klasického Double Ratchet („Root“, „Sending“, „Receiving“ reťazce).
- pq_{state}: Stav postkvantového ratchet („SPQR“ stav).

Počas šifrovania systém načítava jeden kľúč správy z každého podprotokolu:

$$ec_{mk} \leftarrow \mathsf{RatchetSendKey}(\texttt{ec\_state})$$
$$pq_{mk} \leftarrow \mathsf{SCKARatchetSendKey}(\texttt{spqr\_state})$$

Potom sa na generovanie konečného šifrovacieho kľúča použije špeciálna hybridná funkcia KDF_{HYBRID}:

$$mk = \mathsf{KDF}_{\mathsf{HYBRID}}(ec_{mk}, pq_{mk})$$

Táto konštrukcia zabezpečuje princíp „Defense in Depth“ (hlboká obrana). Vďaka vlastnostiam KDF, ak je aspoň jeden zo vstupov kryptograficky silný (tajný a náhodný), potom bude silný aj výstup (mk). To pokrýva dva scenáre:

1. **Kvantové prelomenie:** Ak je klasický algoritmus (ec_{mk}) kompromitovaný, entropia pq_{mk} bude naďalej chrániť správu.

2. **Algoritmická chyba:** Ak sa v ešte stále mladej implementácii PQC nájde chyba alebo sa preukáže, že jej matematické predpoklady sú nesprávne, klasický ec_{mk} bude naďalej poskytovať obvyklú úroveň ochrany.

1.11.5 Komunikačná hlavička a synchronizácia

Zavedenie Triple Ratchet má významný vplyv na protokol prenosu dát, najmä pokiaľ ide o metadáta. Hlavička správy sa stáva komplexnou štruktúrou, ktorá musí obsahovať synchronizačné dáta oboch vrstiev:

Hlavička EC:

- **Aktuálny verejný kľúč odosielateľa** (DH): Efemérny kľúč pre Diffie-Hellman ratchet, ktorý umožňuje post-compromise security v klasickej rovine.
- **Počet správ v predchádzajúcom reťazci** (PN): Táto hodnota informuje prijímateľa, koľko správ bolo odoslaných v predošlom reťazci eliptického ratchetu, čo umožňuje detekciu stratených správ.
- **Poradové číslo správy** (n): Index správy v rámci aktuálneho symetrického reťazca pre daný eliptický kľúč.

Hlavička SCKA:

- **SCKA správa** (msg): Dáta samotného SCKA protokolu (napr. ML-KEM Braid). V praxi sú to fragmenty alebo „chunky“ postkvantového verejného kľúča (EK) alebo šifrovaného textu (CT), ktoré sa postupne prenášajú naprieč viacerými správami.
- **Číslo správy v SPQR** (n): Poradové číslo správy v rámci aktuálnej postkvantovej epochy. Toto číslo je kľúčové pre odvodenie správneho kľúča správy (pq_mk) zo symetrického reťazca SPQR a pre identifikáciu správ, ktoré prišli mimo poradia.

Po analýze hlavičky prijímajúca strana prechádza obe stavové stroje samostatne. Táto modularita zvyšuje odolnosť systému a umožňuje, aby bola v budúcnosti nahradená akákoľvek zložka (napr. algoritmus PQC novším štandardom NIST) bez nutnosti prepracovať celú architektúru protokolu. Zároveň je počas implementácie potrebné venovať osobitnú pozornosť spracovaniu zvýšenej veľkosti hlavičky, aby nespôsobovala neprijateľné spomalenie v mobilných sieťach s obmedzenou šírkou pásma.

2 Aktuálny riešenie na trhu

Porozumenie moderným kryptografickým protokolom, najmä protokolu Signal, ktorý poskytuje E2EE, môže byť náročnou úlohou pre vysokoškolských študentov aj IT odborníkov. Cieľom tejto kapitoly je kriticky analyzovať aktuálne dostupné vzdelávacie materiály a technické zdroje a preukázať, že v súčasnosti neexistuje na trhu žiadna integrovaná aplikácia, ktorá by vizuálne a interaktívne prezentovala plnú funkčnosť protokolu, vrátane najnovších postkvantových mechanizmov.

2.1 Teoretické zdroje a oficiálne špecifikácie

Primárnym zdrojom informácií o protokole Signal sú technické špecifikácie a oficiálne blogové príspevky nadácie Signal Foundation. Tieto zdroje vynikajú v rámci odvetvia svojou čitateľnosťou a matematickou presnosťou pri definovaní komplexných vzťahov. Dokumentácia je modulárna, čo umožňuje samostatne študovať jednotlivé komponenty, ako sú podanie kľúčov X3DH, algoritmus Double Ratchet alebo ML-KEM Braid Protocol.

Napriek ich vysokej kvalite sú tieto materiály určené primárne pre odborníkov a pokročilých študentov. Pre študentov bez predchádzajúcich vedomostí v oblasti kryptografie (napríklad ak ešte neabsolvovali relevantný predmet) môžu byť tieto zdroje príliš abstraktné. Hustota informácií a chýbajúca vizualizácia dynamických zmien stavov môžu viesť k nesprávnym interpretáciám a chybným záverom.

2.2 Existujúce interaktívne a vizuálne riešenia

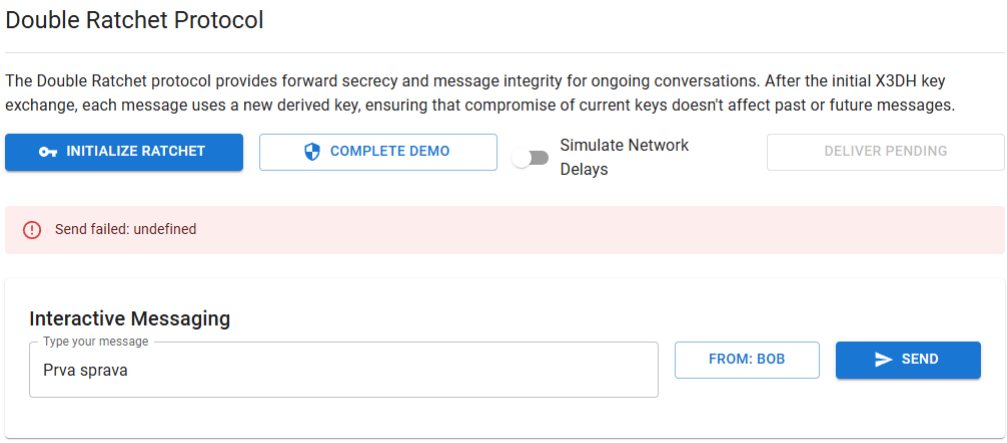
Na trhu v súčasnosti existujú len čiastkové riešenia, ktoré sa zvyčajne zameriavajú na izolované časti protokolu

2.2.1 Signal Storybook[21]

Táto webová aplikácia predstavuje v súčasnosti jeden z najrelevantnejších dostupných nástrojov, ktorý ponúka interaktívne prostredie na štúdium fungovania protokolu Double Ratchet. Rozhranie umožňuje používateľom sledovať proces výmeny kľúčov X3DH krok za krokom, zatiaľ čo modul Double Ratchet poskytuje možnosť odosielania vlastných správ a simuláciu sieťového oneskorenia. Podľa oficiálneho GitHub repozitára je aplikácia predmetom aktívneho vývoja, preto zistenia a popisy uvedené v tejto práci reflektujú stav platný k momentu jej písania (marec 2026)[22].

Pri používaní stránky možno v jej fungovaní postrehnúť niekoľko nedostatkov. Ak napríklad v module Double Ratchet iniciujeme komunikáciu zo strany Boba odoslaním

prvej správy, systém zobrazí chybové hlásenie bez akéhokoľvek kontextu či bližších podrobností, to môžme vidieť na obrázku 3. Podobné implementačné obmedzenia sa objavujú aj pri simulácii sieťového oneskorenia. Hoci nástroj túto funkciu ponúka, v praxi to znamená, že odoslané správy sa dočasne ukladajú do frontu. Následne sú po stlačení tlačidla doručené všetky čakajúce správy v pôvodnom poradí. Rozhranie okrem času oneskorenia uvedeného pri správe neposkytuje žiadnu vizuálnu spätnú väzbu o tom, v čom sa toto prijímanie líši od bežnej prevádzky, viditeľné na obrázku 4. Výrazným obmedzením je aj to, že simuláciu oneskorenia nefunguje vždy, napriek tomu, že rozhranie indikuje, že je zapnúta. Ďalšou anomáliou, ktorá však nesúvisí s oneskorením, je chyba pri zobrazovaní poradového čísla správy (Message number) – to sa v každom kroku Double Ratchet algoritmu pri prvej (nultej) správe nezobrazuje správne a nadobúda hodnotu „Message #N/A“, viditeľné tiež na obrázku 4.

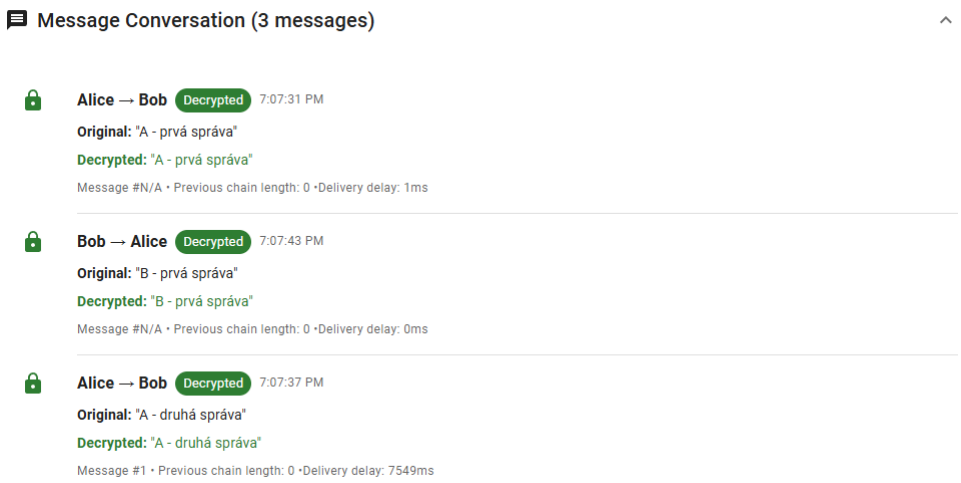


Obr. 3: Chyba pri odosielaní prvej správy zo strany Boba

Pri posudzovaní vyššie uvedených anomálií je však potrebné vziať do úvahy popis projektu, podľa ktorého bolo primárnym cieľom vytvoriť funkčnú online implementáciu protokolu Signal, a nie aplikáciu určenú výhradne na vzdelávacie účely. Možno konštatovať, že tento nástroj pôvodný cieľ v zásade spĺňa, ak odhliadneme od spomínaných nedostatkov v používateľskom rozhraní a simulácii, aplikácia v pozadí realizuje funkčnú implementáciu protokolu Signal.

2.2.2 Naratívne vizualizácie (Vas3k blog)

Tento blogový príspevok[23] pomáha budovať základnú intuíciu naratívnym spôsobom pomocou kreatívnych analógií (napríklad „kombinačné zámky“ a „ratchet stromy“) [23]. Tento prístup je vynikajúci na pochopenie na úrovni konceptov, ale chýba mu technická hĺbka, presnosť a interaktivita, ktorá sa od vzdelávacej aplikácie očakáva.



Obr. 4: Oneskorená správa a chyba v poradových číslach

2.2.3 Video zdroje

Mnohé YouTube kanály (napr. Computerphile[24]) a ďalšie video zdroje sa zaoberajú vysvetlením protokolu Signal. Tieto zdroje sa spravidla zameriavajú na algoritmus „Double Ratchet“ a koncept „forward secrecy“. Hoci pomáhajú pri popularizácii témy, zvyčajne predstavujú len vybrané časti protokolu a nie sú vhodné na podrobnú demonštráciu celej architektúry alebo najnovšieho vývoja.

2.2.4 Metodická analógia: The Illustrated TLS Connection

Hoci nesúvisí priamo s protokolom Signal, je dôležité spomenúť stránky „The Illustrated TLS Connection“ ako existujúce metodické príklady. Tieto zdroje podrobne vysvetľujú nadviazanie spojenia (handshake) TLS 1.2[25] a 1.3[26], kde je možné interaktívne prehliadať a analyzovať obsah každého odoslaného dátového balíka. Jedným z cieľov tejto práce je dosiahnuť podobnú hĺbku vizualizácie prispôsobenú špecifikám protokolu Signal.

2.3 Postkvantový prechod a mechanizmus SPQR

Najvýznamnejším aktuálnym vylepšením protokolu Signal je Sparse Post-Quantum Ratchet (SPQR), ktorý dopĺňa existujúci algoritmus Double Ratchet o kvantovo bezpečnú vrstvu. Táto technológia je postavená na mechanizme ML-KEM, ktorý ako hybridné riešenie zabezpečuje, aby dôvernosť správ zostala zachovaná aj v prípade, že by klasické šifrovanie založené na eliptických krivkách bolo v budúcnosti prelomené kvantovým počítačom.

Technická komplexnosť SPQR je momentálne dostupná takmer výlučne v akademických publikáciách a v surovom zdrojovom kóde. Neexistuje zrozumiteľná, interaktívna vizualizácia, ktorá by demonštrovala, ako je zostavený stavový automat „Triple Ratchet“ alebo ako v praxi prebieha výmena kľúčov založená na epochách.

2.4 Medzera na trhu a dôvody pre vývoj aplikácie

Na základe analýzy môžeme konštatovať, že vo výučbe protokolu Signal existuje značná trhová a vzdelávacia medzera. Súčasné zdroje sú buď príliš teoretické (špecifikácie), alebo sa sústredia len na izolované detaily protokolu. Chýba najmä riešenie, ktoré by bolo:

- **Integrované**: Na jednom rozhraní prezentuje proces od registrácie a X3DH/PQXDH handshake až po priebežné operácie Triple Ratchet.
- **Moderné**: Ako prvé zahŕňa vizuálne vysvetlenie mechanizmov SPQR a SCKA.
- **Vizuálne interaktívne**: Umožňuje používateľovi manuálne krokovať procesy, sledovať zmeny kľúčového materiálu a triky na optimalizáciu šírky pásma (napr. delenie kľúčov).

Na základe týchto skutočností môžeme vyhlásiť, že aplikácia plánovaná ako cieľ tejto práce zapĺňa existujúcu medzeru. Nie je len ďalším teoretickým zhrnutím, ale technologickým mostom, ktorý premieňa precízne opisy Signal Foundation a výsledky akademického výskumu na hmatateľné, prehľadné procesy, čím pripravuje budúcich inžinierov na éru postkvantovej bezpečnosti.

3 Požiadavky na aplikaciu

3.1 Funkcionálne požiadavky

Tieto požiadavky definujú, aké funkcie musí systém poskytovať a ako má reagovať na vstupy používateľa.

Všeobecné požiadavky aplikácie

1. **Výber edukačného modulu:** Systém musí používateľovi umožniť výber z nasledujúcich kryptografických modulov: X3DH, PQXDH, Double Ratchet, SPQR (SCKA) a Triple Ratchet.

2. **Správa stavu (Scenáre):** Systém musí umožniť uloženie aktuálneho stavu celej simulácie do súboru a jeho následné načítanie. Táto funkcia slúži na prípravu a demonštráciu predpripravených edukačných scenárov.

Modul Double Ratchet (Hlavné zameranie)

1. **Riadenie komunikácie:** Používateľ musí mať možnosť manuálne vytvárať správy (vo forme otváraného textu), zvoliť odosielateľa správy a manipulovať s doručovaním správ (napr. doručenie mimo poradia, zahodenie správy).

2. **Zobrazenie stavu komunikácie:** Systém musí v reálnom čase zobrazovať aktuálny stav každého účastníka komunikácie. To zahŕňa zoznam správ (s obsahom a hlavičkou), aktuálne kryptografické kľúče, reťazce (chains) a počítadlá. Na začiatku simulácie musí systém poskytnúť sumárne zobrazenie úvodného stavu inicializovaného pomocou X3DH.

3. **Detailná vizualizácia výpočtov:** Po kliknutí na príslušné tlačidlo musí systém otvoriť nové okno (vyskakovacie okno), ktoré detailne a krok za krokom vizualizuje kryptografické operácie pre danú správu na strane odosielateľa alebo príjemcu. Vizualizácia musí obsahovať reálne dáta a matematické kroky (napr. prijatie verejného DH kľúča, odvodenie kľúča odosielacieho reťazca, generovanie nového DH páru, odvodenie koreňového kľúča).

4. **Správa histórie kľúčov:** Systém musí uchovávať a zobrazovať históriu kryptografických kľúčov. Viditeľnosť týchto kľúčov sa musí dynamicky meniť na základe zvolenej perspektívy.

5. **Perspektívy používateľa (Úrovne viditeľnosti):** Systém musí podporovať prepínanie medzi nasledujúcimi perspektívami, ktoré menia viditeľnosť dát a možnosti interakcie:

(a) **Globálna perspektíva:** Systém zobrazí všetky verejné aj privátne dáta oboch účastníkov. Všetky správy sú dešifrovateľné. Používateľ môže plne manipulovať s odosielaním aj prijímaním za obe strany a vidí kompletnú históriu kľúčov.

(b) **Perspektíva účastníka (Alica alebo Bob):** Systém zobrazí iba verejné dáta z hlavičiek, dáta z nich odvodené a privátne dáta zvoleného účastníka. Systém nesmie zobraziť privátne dáta (kľúče) druhej strany, hoci ich existencia musí byť logicky indikovaná na základe dostupného stavu. Používateľ môže odosielať správy za obe strany, ale manuálne prijímať a krokovo vizualizovať správy smie len z pohľadu zvoleného účastníka.

(c) **Perspektíva útočníka (Attacker Dashboard):** V module DoubleRatchet systém poskytne rozhranie na výber kompromitovaných kľúčov (aktuálnych aj historických – koreňové, reťazcové, spracovávacie, DH kľúče).

- Ak nie je zvolený žiadny kompromitovaný kľúč, systém zobrazí iba verejné dáta.
- Ak je kľúč kompromitovaný, systém musí zobraziť dešifrovaný obsah príslušných správ a automaticky odvodiť všetky ďalšie kompromitovateľné kľúče (napr. znalosť koreňového a privátneho DH kľúča odhalí prijímací reťazec).
- Systém musí vizuálne zvýrazniť tie kľúče, ktoré neboli kompromitované priamo, ale boli vypočítané na základe iných kompromitovaných dát.
- Systém musí byť schopný prostredníctvom tejto perspektívy jasne demonštrovať vlastnosť „post-compromise security“ (bezpečnosť po kompromitácii).

Ostatné moduly

1. **Modul X3DH:** Systém musí detailne krokovať protokol X3DH a vizualizovať špecifické hraničné situácie (napr. absencia jednorazových kľúčov - OPK na serveri, generovanie nového SPK balíčka stranou B).

2. **Modul PQXDH:** Systém musí poskytovať funkcionalitu analogickú modulu X3DH, pričom sa zameriava primárne na správanie a štruktúru nových postkvantových kľúčov.

3. **Modul SPQR (SCKA):** Systém musí ilustrovať fungovanie protokolu a poskytovať vizualizáciu výpočtov podobne ako modul Double Ratchet. Tento modul nemusí obsahovať perspektívu útočníka.

4. **Modul Triple Ratchet:** Systém musí demonštrovať synergiu protokolov Double Ratchet a SPQR. Vizualizácia sa zameriava na proces odvodzovania kľúčov správ, ktoré vznikajú kombináciou čiastkových kľúčov vygenerovaných oboma ratchetingovými mechanizmami súčasne.

3.2 Nefunkcionálne požiadavky

Tieto požiadavky definujú kvalitatívne atribúty, obmedzenia a celkové vlastnosti softvérového systému.

1. **Kryptografická exaktnosť a presnosť:** Všetky simulované kryptografické výpočty (derivácie kľúčov, ratcheting) musia logicky a matematicky zodpovedať oficiálnej špecifikácii Signal protokolu. Aj keď ide o edukačnú aplikáciu, zobrazené závislosti medzi kľúčmi nesmú obsahovať faktické odchýlky od štandardu.

2. **Zameranie na používateľský zážitok (Usability) a ergonómiu:** Keďže ide o edukačnú aplikáciu, používateľské rozhranie musí byť navrhnuté tak, aby nedochádzalo k informačnému preťaženiu používateľa. Prechody medzi perspektívami (Globálna, A, B, Útočník) a vizualizácia výpočtov vo vyskakovacích oknách musia byť jasne oddelené a intuitívne navigovateľné.

3. **Vizuálna odozva (Feedback):** Systém musí pri zmene stavu (napr. kompromitácia kľúča útočníkom v čase) okamžite a bez nutnosti obnovovania aplikácie (real-time) vizuálne aktualizovať závislé komponenty (zvýraznenie odvodených kľúčov, odkrytie plaintextu).

4. **Obmedzenie účelu aplikácie (Bezpečnostný štandard):** Aplikácia má výhradne demonštračný a edukačný charakter. Nemusí byť navrhnutá ani implementovaná na prenos skutočných citlivých dát. Simulované kľúče a náhodné generátory nemusia spĺňať štandardy pre kryptograficky bezpečné pseudonáhodné generátory (CSPRNG), nakoľko slúžia len na vizualizáciu konceptov.

5. **Modulárna architektúra:** Systém musí byť navrhnutý modulárne, aby bolo v budúcnosti možné plynulo integrovať ďalšie kryptografické mechanizmy (napríklad plnohodnotné rozpracovanie modulu Triple Ratchet) bez nutnosti prepisovania jadra pre správu perspektív a stavov.

6. **Platformová dostupnosť a architektúra:** Primárnym cieľovým prostredím pre nasadenie a beh aplikácie je desktopové prostredie. Systém však musí byť navrhnutý s ohľadom na vysokú mieru prenositeľnosti a modularity (napríklad striktné oddelenie používateľského rozhrania od aplikačnej logiky simulujúcej

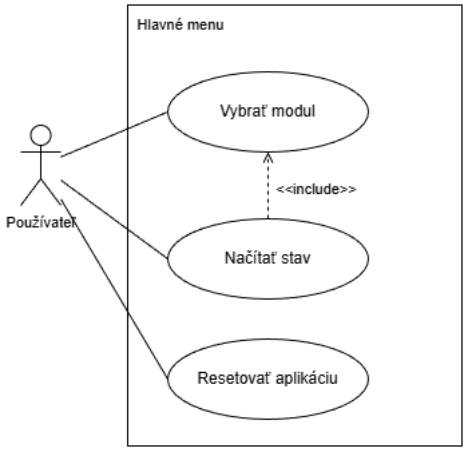
kryptografické operácie). Tento návrh by mal umožňovať alebo výrazne uľahčovať alternatívne nasadenie systému aj vo forme webovej aplikácie bez nutnosti rozsiahleho prepisovania jadra systému.

4 Návrh riešenia

4.1 Návrh prípad použitia

Na základe požiadaviek sme stanovili minimálne funkcie, ktoré musí aplikácia používateľovi poskytovať. Zo všetkých požiadaviek jasne vyplýva, že systém obsahuje viacero voliteľných modulov. Z tohto dôvodu sa používateľovi pri spustení aplikácie ako prvé zobrazí rozhranie na výber modulu.

Tu môže používateľ nielen spustiť nové relacie, ale aj načítať vopred pripravený stav (napríklad scenár zostavený učiteľom). V prípade modulu Double Ratchet môže takýmto scenárom byť prezentácia bezpečnosti po kompromitácii (PCS). To dobre ilustruje, ktorý krok pomáha obnoviť bezpečnosť protokolu po úniku kľúča, aby účastníci mohli opäť posielať bezpečné, šifrované správy. Podobne užitočné môže byť v module X3DH načítanie uloženého stavu, ktorý simuluje, že na serveri už nie sú k dispozícii žiadne kľúče OPK. V záujme dobrého používateľského zážitku aplikácia po načítaní stavu automaticky otvorí príslušný modul. Tieto hlavné prípady použitia ilustruje obrázok č. 5.



Obr. 5: Diagram prípad použitia pre hlavné menu

Moduly môžeme rozdeliť do dvoch hlavných skupín:

- **Protokoly na dohodnutie kľúča (X3DH a PQXDH):** Ich cieľom je vytvorenie spoločného tajného kľúča medzi účastníkmi.
- **Protokoly na odosielanie správ (Double Ratchet, SPQR, Triple Ratchet):** Tu si strany neustále odosielajú a prijímajú správy.

Hoci v praxi je na spustenie protokolov na odosielanie správ potrebná dohoda o kľúči (napr. X3DH), pre lepšiu zrozumiteľnosť je ich podrobný popis uvedený v samostatných

moduloch. V moduloch na dohodu o kľúči môže používateľ krok za krokom sledovať výpočty a kroky. V hlavných moduloch pre odosielanie správ však ešte pred odoslaním alebo prijatím prvých správ uvádzame len kratšiu, všeobecnú vizualizáciu počiatočnej výmeny kľúčov. Tento súhrn neumožňuje zásah používateľa, keďže na podrobné štúdium slúžia samostatné moduly.

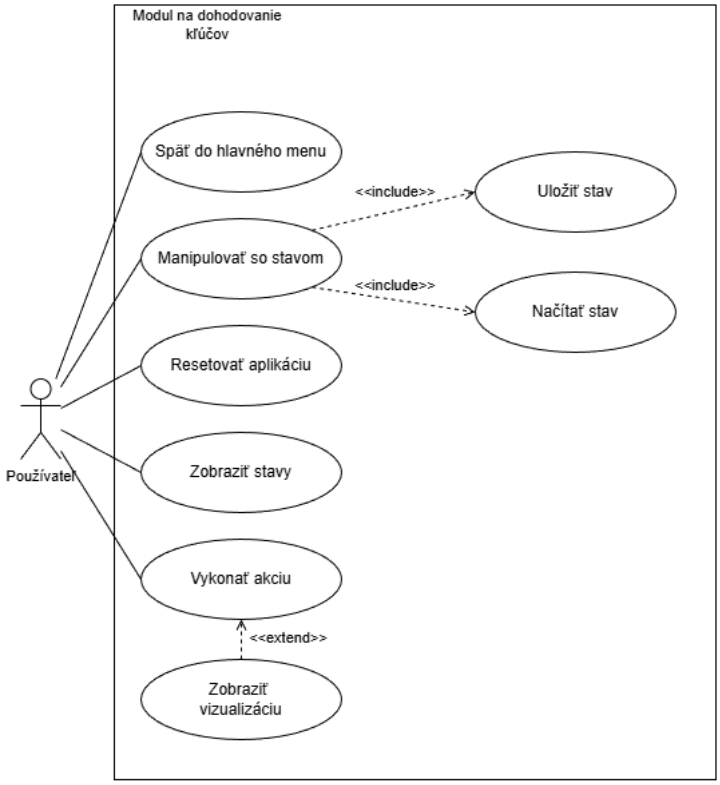
Každý modul musí zabezpečovať štyri základné funkcie:

1. **Návrat do hlavného menu:** Používateľ sa môže kedykoľvek vrátiť späť, aby si vybral iný modul.
2. **Uloženie stavu:** Uloženie aktuálneho procesu do súboru, pričom používateľ si môže vybrať miesto uloženia a zadať názov súboru.
3. **Načítanie stavu:** Načítanie predtým uloženého stavu (scenára).
4. **Obnovenie stavu:** Vymazanie aktuálneho stavu a reštartovanie modulu s čistým štítom.

Moduly **prvej skupiny** (X3DH a PQXDH) musia okrem vyššie uvedených štyroch funkcií spracovávať aj špecifické údaje účastníkov. Konkrétne prípady použitia týchto protokolov na dohodu kľúčov sú znázornené na obrázku č. 6. V týchto protokoloch počítame s tromi účastníkmi: so serverom, na ktorý účastníci nahrajú kľúče; s iniciátorom, ktorý odošle prvú správu; a s príjemcom, ktorý si vopred nahral svoje kľúčové sady a čaká na požiadavku. Aplikácia musí jasne zobrazovať aktuálny stav všetkých troch účastníkov. Používateľ musí mať možnosť ručne prechádzať procesom a aplikácia musí umožňovať vizuálne zobrazenie presných výpočtov príslušných k danému kroku.

Moduly na odosielanie správ patriace do **druhej skupiny** (Double Ratchet, SPQR, Triple Ratchet) je podstatne komplexnejšia. Hoci v týchto prípadoch modelujeme výlučne komunikáciu medzi dvoma účastníkmi (Alice a Bob), v dôsledku asynchrónnej výmeny správ môžu vykonateľné operácie a poradie vykonávania spôsobiť veľkú variabilitu. Okrem základných systémových funkcií tvorí chrbticu týchto modulov interaktívne spracovanie správ: používateľ má možnosť ručne zadať otvorený text (plaintext payload), vybrať aktuálneho odosielateľa a ovplyvniť prijímanie. Spracovanie prichádzajúcich správ môže prebiehať automaticky, v poradí odoslania (in-order delivery), alebo manuálne, pričom používateľ môže simulovať sieťové oneskorenia a správy prichádzajúce mimo poradia (out-of-order).

Aplikácia musí na strane odosielateľa aj príjemcu prehľadne a podrobne vizualizovať prebiehajúce zmeny kryptografického stavu. To má mimoriadny význam na strane príjemcu. Pri spracovaní prichádzajúcej správy musí systém znázorniť fungovanie funkcií

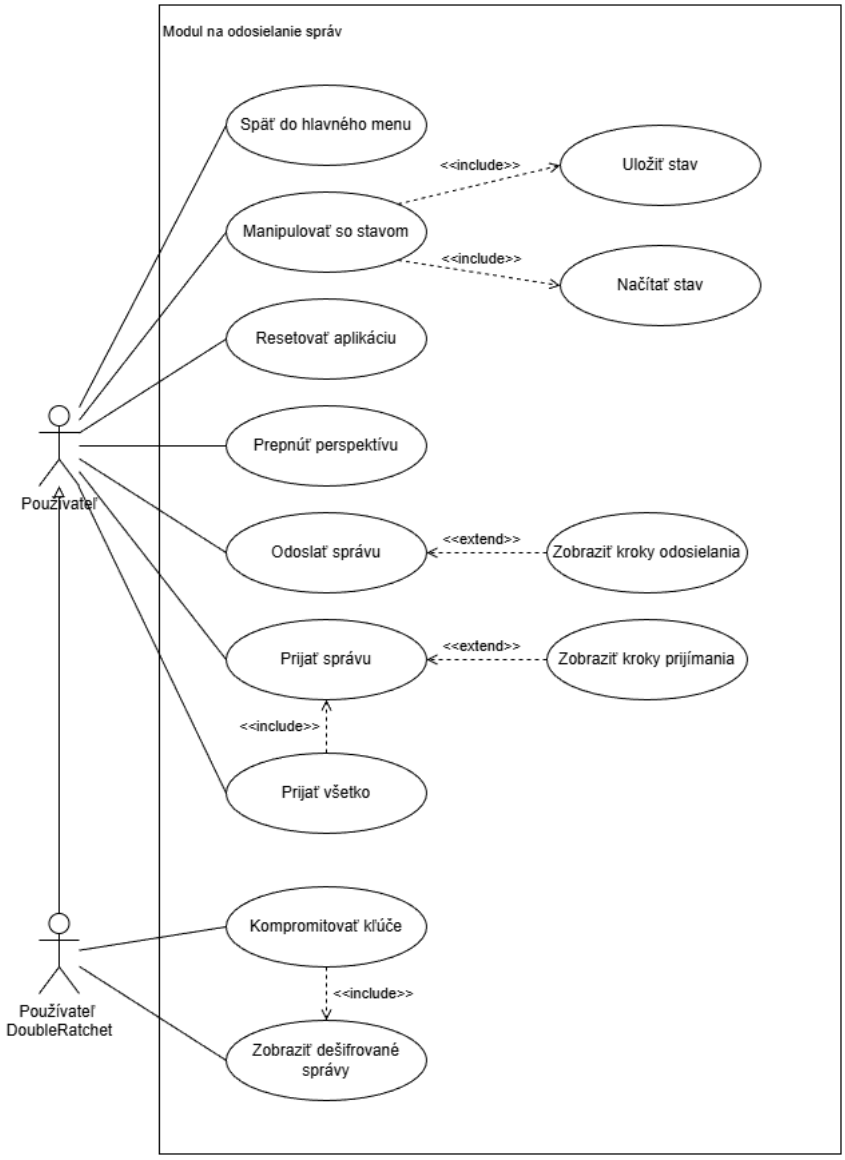


Obr. 6: Diagram prípadov použitia protokolov na výmenu kľúčov

odvodenia kľúča (KDF), aktualizáciu reťazcov a proces posúvania Diffie-Hellmanovej západky (DH ratcheting) v prípade hlavičky obsahujúcej nový verejný kľúč odosielateľa. Okrem toho musí systém vizuálne predviesť a vysvetliť problém so stratenými alebo oneskorenými správami a príslušné riešenie.

Na základe požiadaviek je potrebné zabezpečiť aj zmenu perspektívy: v takomto prípade môže používateľ vidieť len tie údaje, ktoré patria danému účastníkovi, napríklad vlastnené súkromné kľúče a verejné hlavičky viditeľné pre všetkých

Jedným z najdôležitejších prvkov modulu Double Ratchet je perspektíva útočníka (Attacker Dashboard). Pri jej výbere môže používateľ určiť, ktorý kľúč ktorého účastníka útočník kompromitoval. Týmto spôsobom je možné simulovať situácie, keď útočník získa prístup k zariadeniu jednej zo zúčastnených strán, získa súkromné kľúče a pomocou nich sa pokúsi dešifrovať správy odoslané v minulosti alebo v budúcnosti (po ich kompromitácii). Ako je vidieť aj na obrázku č. 7, táto funkcia sa nachádza len v module Double Ratchet, keďže hlavným zameraním diplomovej práce a aplikácie je prezentácia fungovania a mechanizmov tradičného (bez postkvantového) protokolu Signal.



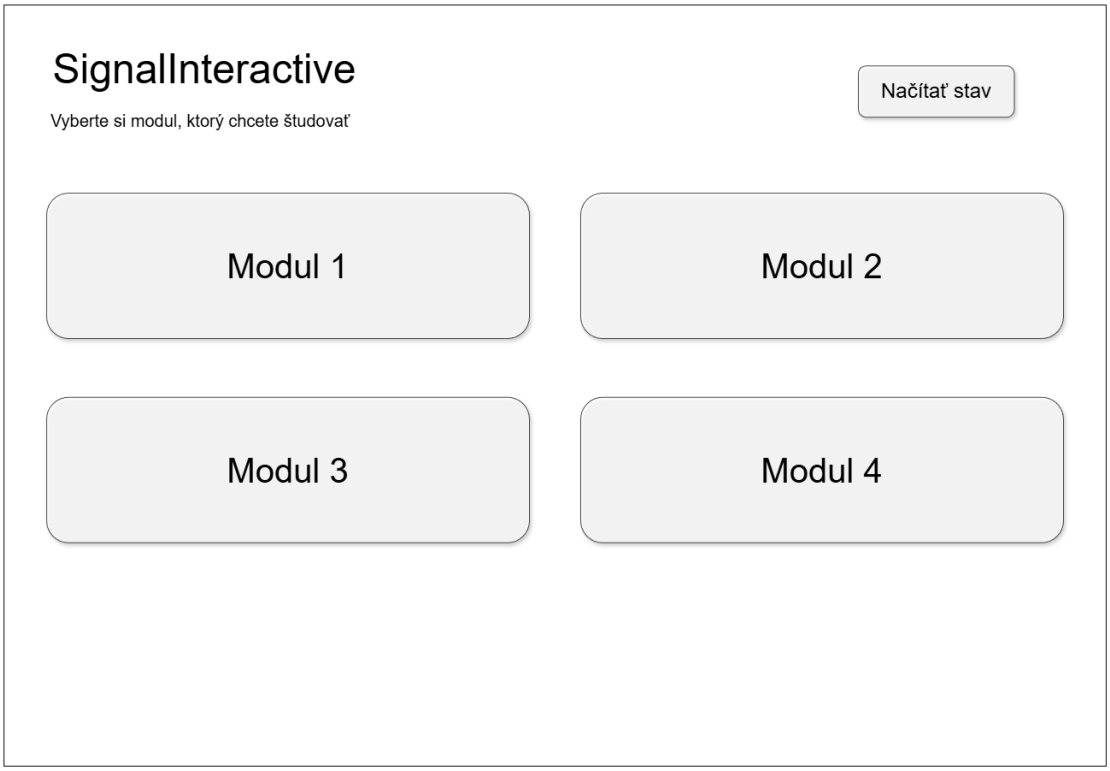
Obr. 7: Diagram prípadov použitia komunikačných protokolov

4.2 Návrh grafického používateľského rozhrania

Z predtým špecifikovaných požiadaviek a prípadov použitia priamo vyplýva logická štruktúra grafického používateľského rozhrania (GUI). Architektúra systému je rozdelená do troch hlavných zobrazení: hlavného menu (výberu modulov), rozhrania protokolov na dohodu kľúčov a rozhrania protokolov na odosielanie správ. Tieto dopĺňajú vyskakujúce okná (pop-up window), ktoré zobrazujú podrobné procesy. A tiež špeciálny pohľad útočníka patriaci k modulu Double Ratchet a s ním súvisiace rozhranie na výber kompromitovaných kľúčov (Attacker Dashboard).

4.2.1 Rozhranie na výber modulov

Toto rozhranie je vstupným bodom aplikácie a predstavuje prehľadné, ľahko orientovateľné grafické rozhranie. Jeho hlavnými prvkami sú interaktívne tlačidlá reprezentujúce jednotlivé moduly. Po kliknutí na vybraný modul systém načíta rozhranie daného protokolu. Ak bol modul už použitý počas aktuálnej relácie (session), systém obnoví posledný známy stav; v opačnom prípade sa spustí úplne nová simulačná relácia. Na tomto rozhraní sa nachádza aj funkcia slúžiaca na načítanie predchádzajúcich stavov (Load State), ktorej fungovanie je podrobne opísané v predchádzajúcej podkapitole. V záujme primeranej užívateľskej skúsenosti (UX) a prehľadnej navigácie obsahuje úvodná stránka aj krátky informatívny pomocný text (výzvu): *Vyberte si modul, ktorý chcete študovať(Select the module you want to examine)* . Vizuálny návrh rozhrania na výber modulov znázorňuje obrázok č. 8.



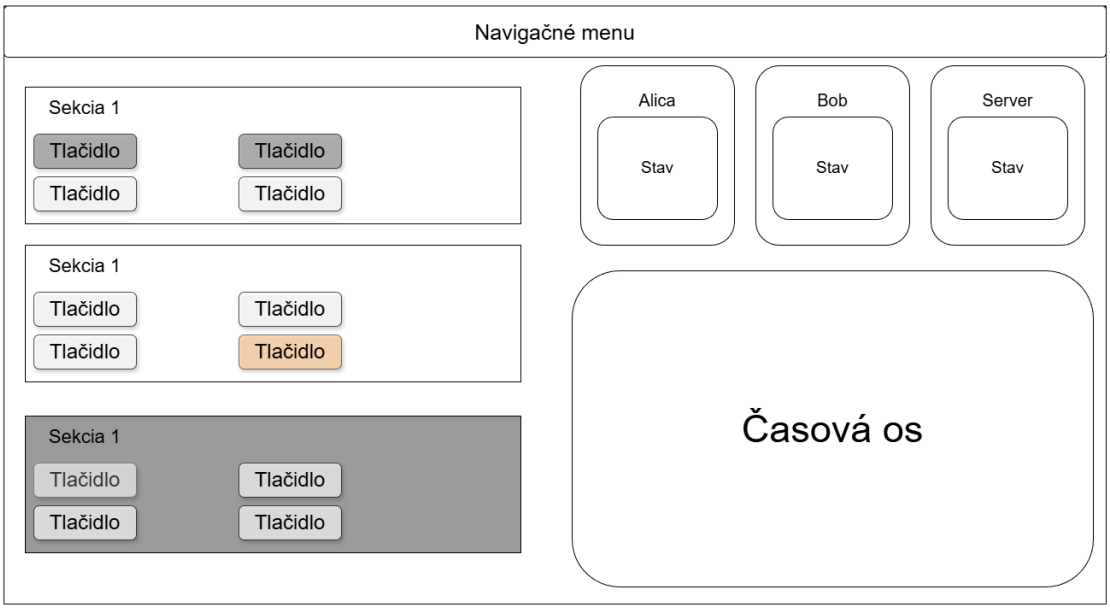
Obr. 8: Náčrt grafického používateľského rozhrania hlavného menu

4.2.2 Rozhranie protokolov na dohodu kľúčov

Vykonávanie protokolov na tomto rozhraní je rozdelené na logické etapy (fázy). Táto štruktúra výrazne uľahčuje sledovanie procesov. Na začiatku simulácie je aktívna iba prvá etapa; prvky nasledujúcich fáz sa vizuálne zobrazujú, ale sú v neaktívnom (disabled) stave. Toto vynútené poradie znemožňuje používateľovi náhodne preskočiť kritické

kroky. Podobne, operácie vyžadujúce jednorazové vykonanie (napríklad generovanie identifikačného kľúča, teda Identity Key, alebo nahratie počiatočného balíka kľúčov na server) sa ihneď po prvom kliknutí deaktivujú. Aby bola interakcia intuitívna aj pre menej skúsených používateľov alebo študentov, aplikácia vizuálne zvýrazňuje nasledujúci logicky očakávaný krok, čím používateľa vedie.

Rozloženie rozhrania je nasledovné: na ľavej strane obrazovky sa nachádzajú vykonateľné operácie (akcie). V hornej časti pravého panela sa zobrazuje aktuálny stav a údaje troch hlavných entít (Alice, Bob a Server). Pod týmto panelom je umiestnená časová os (timeline), ktorá zaznamenáva udalosti v chronologickom poradí, čím umožňuje retrospektívnu analýzu vykonaných operácií. Kostrou rozhrania (wireframe) je znázornená na obrázku č. 9.

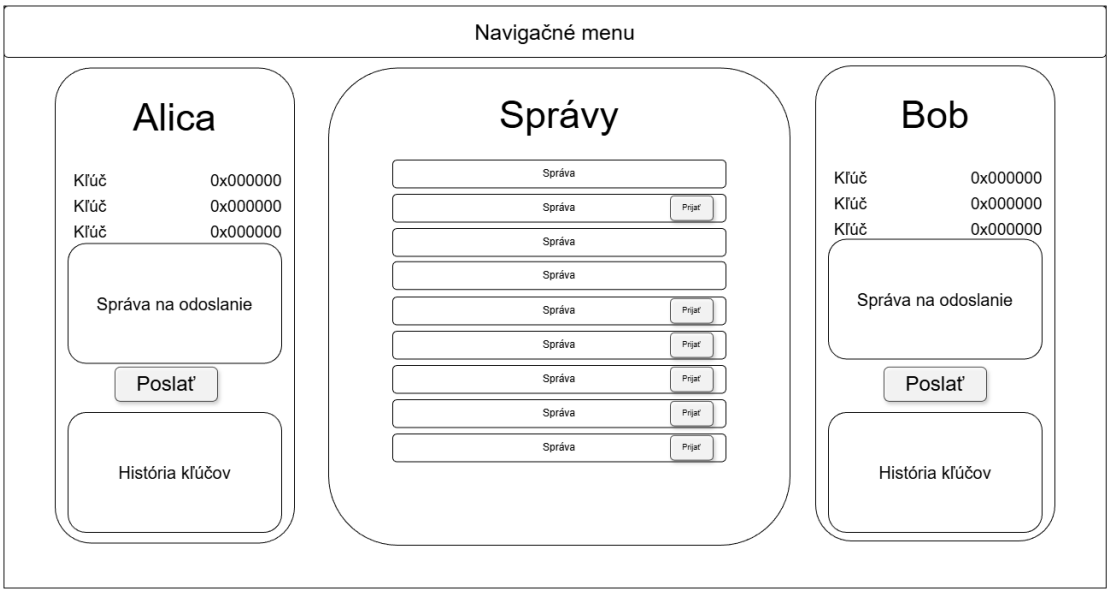


Obr. 9: Náčrt grafického používateľského rozhrania modulu dohodou kľúčov

4.2.3 Rozhranie protokolov na odosielanie správ

Keďže tieto moduly sa zameriavajú na neustálu asynchrónnu komunikáciu, hlavným prvkom grafického rozhrania sú samotné správy. Ako je vidieť na obrázku č. 10, na oboch okrajoch obrazovky sa nachádzajú vyhradené panely komunikujúcich strán (Alice a Bob).

Horná časť panelov účastníkov obsahuje aktuálne kľúče a počítadlá danej entity. Keďže kryptografické kľúče by boli príliš dlhé na užívateľsky prívetivé zobrazenie, systém zobrazuje iba posledných 8 znakov hexadecimálnej reprezentácie kľúčov. Tento prístup sleduje vizuálnu logiku známu z moderných verziových systémov; v prípade Git je



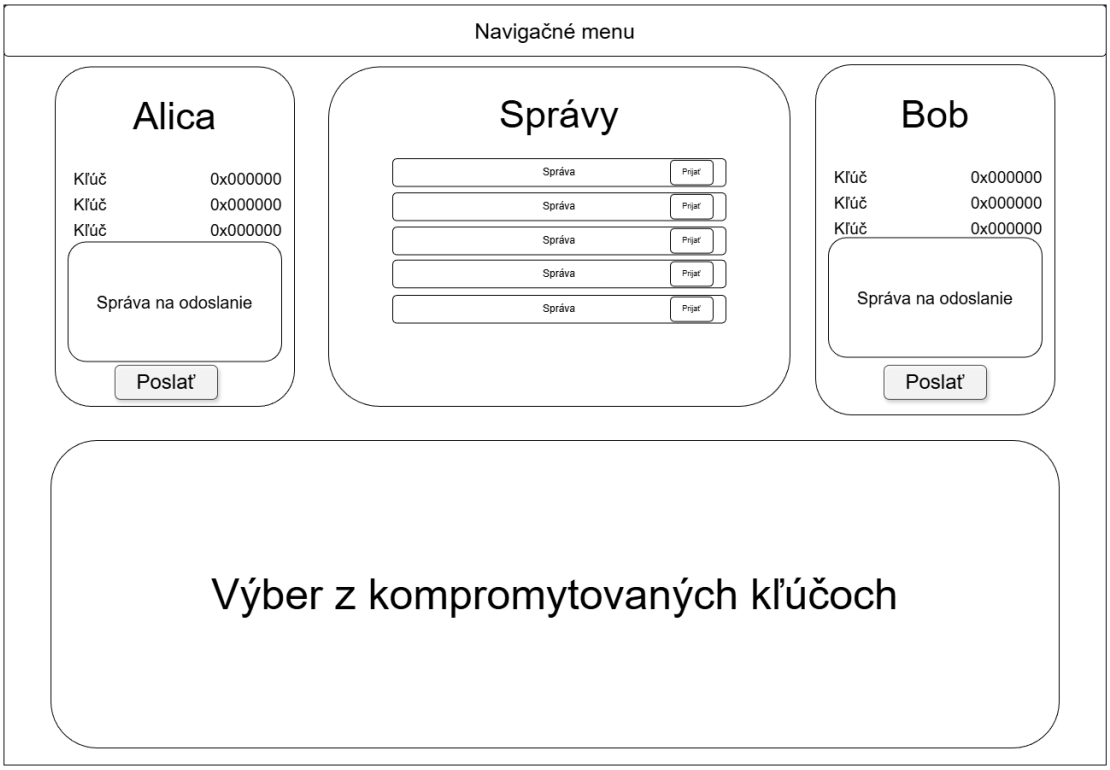
Obr. 10: Náčrt grafického používateľského rozhrania modulu odosielanie kľúčov

osvedčenou praxou skracovanie dlhých identifikátorov SHA-1 (short SHA), kde sa pre ľudskú čitateľnosť štandardne zobrazuje len prvých 7 znakov [27]. Toto dizajnové rozhodnutie slúži na zníženie kognitívnej zaťaženosti (cognitive load) [28]. Kognitívna psychológia, najmä klasické výskumy Millera [29], poukazujú na to, že kapacita ľudskej pracovnej pamäti je prísne obmedzená. V súlade s tým sú používatelia schopní oveľa rýchlejšie a efektívnejšie identifikovať, vizuálne spracovať a porovnať krátke reťazce znakov ako celú dĺžku kryptografických kľúčov. Pod nimi sa nachádza pole na zadávanie správ (input field), kde môže používateľ ručne zadať otvorený text (plaintext payload), ktorý chce odoslať, alebo môže nechať systémom generovanú, očíslovanú správu. Tlačidlo na spustenie odosielania je umiestnené priamo pod vstupným poľom. Zvyšnú vertikálnu časť panelov v spodnej časti vypĺňuje história kľúčov (key history). Toto dizajnové rozhodnutie nielenže zabezpečuje optimálne využitie plochy obrazovky, ale aj úplne spĺňa funkčnú požiadavku, podľa ktorej musí aplikácia transparentne prezentovať v minulosti použité kľúče.

Stredná časť obrazovky slúži na zobrazenie správ, ktoré prechádzajú sieťou a boli odoslané. Vedľa odoslaných, ale ešte nedoručených správ sa zobrazuje špeciálne tlačidlo. Jeho stlačením príjemca manuálne prijme a spracuje vybranú správu. Tento mechanizmus poskytuje veľmi jednoduchý a intuitívny spôsob simulácie a prezentácie sieťových anomálií (napríklad správ prichádzajúcich mimo poradia (out-of-order) alebo výrazne oneskorených správ).

Pri výbere perspektívy útočníka integrovanej do modulu Double Ratchet sa grafické rozhranie dynamicky preusporiada. V takomto prípade sa stredný panel obsahujúci

správy vertikálne zmenší a z panelov účastníkov zmizne história kľúčov. Ich miesto zaujíma rozhranie kompromitovaného výberu kľúčov , ako je vidieť na obrázku č. 11. Na tomto rozhraní sa používateľ, ktorý prevzal úlohu útočníka, dostane k všetkým minulým a súčasným kľúčom vygenerovaným počas relácie. Tento zoznam sa neobmedzuje len na správové kľúče (message keys), ale zahŕňa aj reťazové kľúče (chain keys) slúžiace na ich odvodenie, ako aj asymetrické kľúčové páry patriace k Diffie-Hellmanovmu račňu.

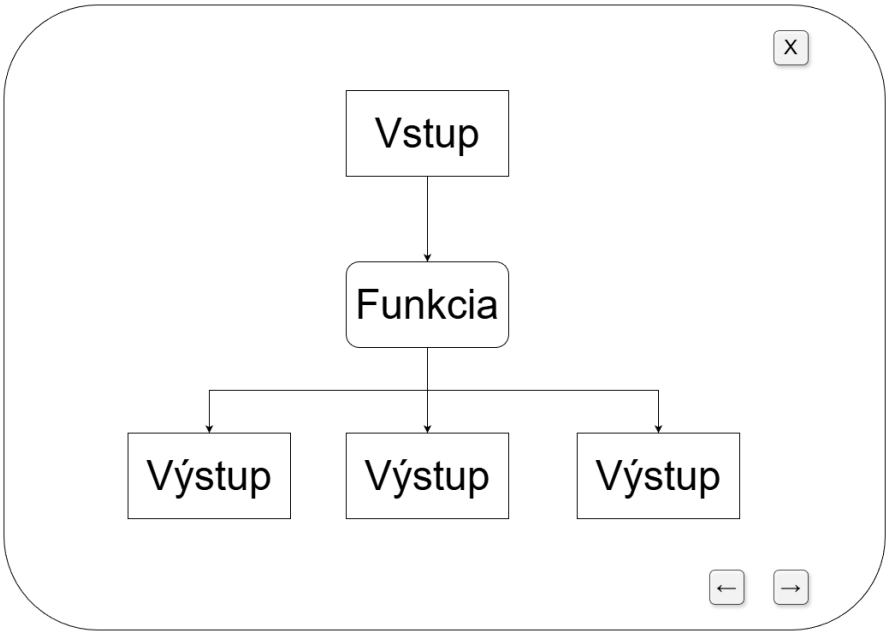


Obr. 11: Náčrt grafického používateľského rozhrania panela útočníka

4.2.4 Vyskakovacie okná na vizualizáciu procesov

Jednou z najdôležitejších funkcií edukačnej aplikácie je detailná vizualizácia kryptografických krokov, ktorá zrozumiteľne ukazuje, aké presné výpočty entít prebiehajú v pozadí a v akom poradí. Prevažná väčšina krokov, ktoré sa majú vizualizovať, predstavuje volanie špecifických kryptografických primitív a funkcií (napr. funkcií odvodenia kľúča alebo šifrovania), ktoré majú definované vstupné parametre (input) a produkujú konkrétne výstupné hodnoty (output). V záujme vizuálnej jednoznačnosti a rýchlej rozlíšiteľnosti (ako je to vidieť aj na obrázku č. 12) používame špecifický systém označovania (notáciu): funkcie a akcie symbolizujú obdĺžniky so zakrútenými rohmi, zatiaľ čo vstupné a výstupné údaje symbolizujú tradičné obdĺžniky. Smer toku dát jednoznačne označujú šípky. Tieto vizualizačné okná obsahujú tri ovládacie tlačidlá na zabezpečenie

interaktivity. Dve z nich slúžia na navigáciu po jednotlivých krokoch v rámci procesu (Predchádzajúci/Ďalší – Previous/Next). Tretie tlačidlo slúži na zatvorenie okna (Close); po jeho stlačení vizualizácia okamžite zmizne a používateľ môže bez prerušenia pokračovať v interakcii s hlavným modulom na odosielanie správ, napríklad môže iniciovať odoslanie alebo prijatie nasledujúcej správy.



Obr. 12: Náčrt grafického používateľského rozhrania vizualizácie

5 Implementácia

5.1 Technológie a nástroje použité pri implementácii

Po navrhnutí architektúry systému a grafického rozhrania bol kľúčovým krokom výber vhodnej technológie. Keďže primárnym cieľom aplikácie je demonštrácia a vizualizácia komplexného protokolu Signal na vzdelávacie účely, boli potrebné vývojárske nástroje, ktoré umožňujú priehľadnú implementáciu zložitej logiky na pozadí a zároveň zabezpečujú vytvorenie moderného, interaktívneho a platformovo nezávislého používateľského rozhrania (UI). Na základe týchto kritérií sme pre vývoj zvolili programovací jazyk Python a grafický framework/knižnicu Flet.

5.1.1 Programovací jazyk Python

Implementácia logiky tvoriacej jadro systému prebehla v programovacom jazyku Python. Python je vysokoúrovňový, všeobecného účelu, dynamicky typizovaný a objektovo orientovaný jazyk, ktorý sa v poslednom desaťročí stal jedným z najrozšírenejších programovacích jazykov [30].

Výber jazyka Python v tomto projekte odôvodňovalo viacero faktorov:

- **Čitateľnosť a prehľadná syntax:** V prípade softvéru určeného na vzdelávacie účely, resp. demonštračných nástrojov s otvoreným zdrojovým kódom je veľmi dôležité, aby bol samotný zdrojový kód študovateľný. Pseudokódová, prehľadná syntax jazyka Python výrazne znižuje kognitívnu záťaž [31], vďaka čomu študenti v budúcnosti ľahšie pochopia aj implementáciu algoritmov.
- **Rýchly vývoj prototypov:** Iteratívny vývoj a testovanie ratcheting mechanizmov a stavov protokolu Signal je možné v jazyku Python realizovať oveľa rýchlejšie ako v jazykoch nižšej úrovne so statickým typizovaním (napr. C++ alebo Java) [32].
- **Podpora kryptografických knižníc:** Aj keď edukačný softvér implementuje mnoho algoritmov v simulovanej forme vo vlastnej implementácii, rozsiahly ekosystém jazyka Python (napríklad balíky *cryptography* alebo *PyCryptodome*) poskytuje stabilný základ pre bezpečné a štandardné hašovacie funkcie, ako aj operácie založené na eliptických krivkách (ECC) [33].

5.1.2 Rámec Flet (Flet Framework)

Zatiaľ čo je Python vynikajúcou voľbou na napísanie backendovej logiky, tradičné knižnice Python GUI (ako Tkinter alebo PyQt) majú často zastaraný vizuálny vzhľad

alebo vyžadujú komplexnú kódovú bázu na vytvorenie moderných reaktívnych rozhraní (reactive UI). Na prekonanie tohto problému sme na implementáciu grafického rozhrania aplikácie použili framework **Flet**. Flet je relatívne nový open-source UI framework, ktorý umožňuje vývojárom vytvárať interaktívne, multiplatformové (cross-platform) desktopové, webové a mobilné aplikácie výlučne pomocou Python kódu. Pod kapotou Flet beží robustný framework **Flutter** vyvinutý spoločnosťou Google [34]. Flet v podstate tvorí most (wrapper) medzi logikou Pythonu a renderovacím jadrom Flutteru.

Architektúra Fletu využíva model klient-server (client-server model), a to aj v prípade, že aplikácia beží lokálne v desktopovom prostredí. Skript v jazyku Python (backend) spravuje stavy a udalosti, zatiaľ čo vizuálne vykresľovanie vykonáva klient Flutter bežiaci na pozadí prostredníctvom pripojenia cez WebSocket [34]. Toto riešenie prinieslo počas vývoja aplikácie viacero výhod:

1. **Deklaratívne a komponentovo orientované používateľské rozhranie:** Flet umožnil implementáciu vopred navrhnutých grafických prvkov (panely účastníkov, bloky správ, časové osy) ako samostatné, opakovane použiteľné triedy (vlastné ovládacie prvky). Komponenty sú usporiadané do hierarchického stromu (control tree), čo výrazne zvýšilo prehľadnosť kódu grafického rozhrania.

2. **Reaktívne riadenie stavu (Reactive State Management):** Pri simulácii kryptografických modulov sa údaje neustále menia (napríklad pri prijatí správy). Vyvolaním metódy `update()` v Flet sa rozhranie automaticky aktualizuje v reálnom čase.

3. **Jednoduchosť zmeny pohľadu a perspektívy:** Ako bolo špecifikované v kapitole o návrhu, systém musí podporovať prechod medzi Attacker Dashboardom a rôznymi perspektívami účastníkov. Pomocou Fletu sa tieto komplexné zmeny rozloženia používateľského rozhrania (napríklad skrytie histórie kľúčov a zobrazenie výberu kompromitovaných kľúčov) mohli realizovať plynule manipuláciou atribútov viditeľnosti príslušných kontajnerov rozloženia (Rows, Columns, Stacks) bez toho, aby došlo k „zamrznutiu“ rozhrania (UI blocking) alebo bolo potrebné stránku znovu načítať.

4. **Jednotná kódová základňa (Single Codebase):** Vďaka použitiu Flet sa dalo vyhnúť zapojeniu klasických webových technológií (HTML, CSS, JavaScript) [35]. Vďaka tomu môže logika frontendu a backendu aplikácie bežať v tom istom jazyku a byť úzko integrovaná, čo výrazne skrátilo čas vývoja a znížilo počet použitých technológií (a tým aj zdrojov chýb).

Na zhrnutie možno povedať, že flexibilita jazyka Python v oblasti spracovania údajov a výpočtov v kombinácii s modernými deklaratívnymi zobrazovacími schopnosťami Fletu založeného na Flutteri poskytla optimálne technické riešenie na vytvorenie komplexnej aplikácie určené na vzdelávacie účely, ktorá nielenže je funkčne presná, ale aj z ergonomického a vizuálneho hľadiska spĺňa súčasné očakávania v oblasti vývoja softvéru.

5.2 Softvérová architektúra a štruktúra aplikácie

Po výbere vhodných technológií bol ďalším krokom implementácie logická a fyzická štrukturalizácia kodovej bázy. V prípade komplexnej edukačnej aplikácie, ktorá zahŕňa viacero nezávislých modulov, je nevyhnutná vysoká úroveň modularity a znovu použiteľnosť kódu (reusability). Za týmto účelom sme pri návrhu systému postupovali podľa návrhového vzoru **Model-View-Controller (MVC)** a zásad objektovo orientovaného programovania (Object-Oriented Programming, OOP) [36].

V koreňovom adresári aplikácie sa nachádza súbor `main.py`, ktorý predstavuje vstupný bod, a súbor `common.py`, ktorý obsahuje globálne konštanty a pomocné funkcie. Chrbticu bázy kódu tvoria tri hlavné adresáre, ktoré funkčne oddeľujú rôzne komponenty systému:

- **`assets/`:** Tento adresár zodpovedá za ukladanie statických zdrojov, napríklad ikon, obrázkov a externých konfiguračných súborov. S cieľom zvýšiť udržateľnosť (maintainability) a znížiť redundanciu zdrojového kódu boli dlhšie textové obsahy (napríklad vyskakujúce vysvetľujúce texty) presunuté do vyhradeného súboru. Tento prístup zaručuje, že súbory kódu nebudú zbytočne dlhé a úprava textového obsahu nebude vyžadovať zásah do samotnej programovej logiky.

- **`components/`:** Tento modul obsahuje všeobecné, na protokoloch nezávislé stavebné prvky systému. Patrí sem súbor `router.py`, zodpovedný za navigáciu a prepínanie medzi zobrazeniami, súbor `persistence.py`, ktorý zabezpečuje správu súborov, a súbor `module_menu.py`, generujúci hlavné menu. Okrem toho súbor `data_classes.py` definuje globálne dátové štruktúry, prostredníctvom ktorých medzi sebou komunikujú rôzne moduly.

- **`modules/`:** Tento adresár obsahuje samotnú kryptografickú biznis logiku a špecifické užívateľské rozhrania. V záujme maximálneho uplatnenia modularity a princípu DRY (Don't Repeat Yourself – Neopakuj sa) [37] je táto sekcia postavená na prísnej objektovo orientovanej dedičskej hierarchii (inheritance hierarchy).

5.2.1 Vzor MVC a dedičná hierarchia (Inheritance Model)

Architektúra jednotlivých kryptografických modulov jednotne sleduje vzor MVC. Každý modul je rozdelený do troch hlavných súborov: súbor `logic.py` zodpovedá za správu kryptografických stavov a algoritmov (Model), súbor `view.py` zodpovedá za renderovanie grafických prvkov pomocou Flet (View), súbor `step_visualization.py` zhromažďuje UI elementy určené pre vizualizáciu krokov algoritmu, zatiaľ čo súbor `module.py` je kontrolér (Controller), ktorý spája logiku s rozhraním a spravuje interakcie používateľa.

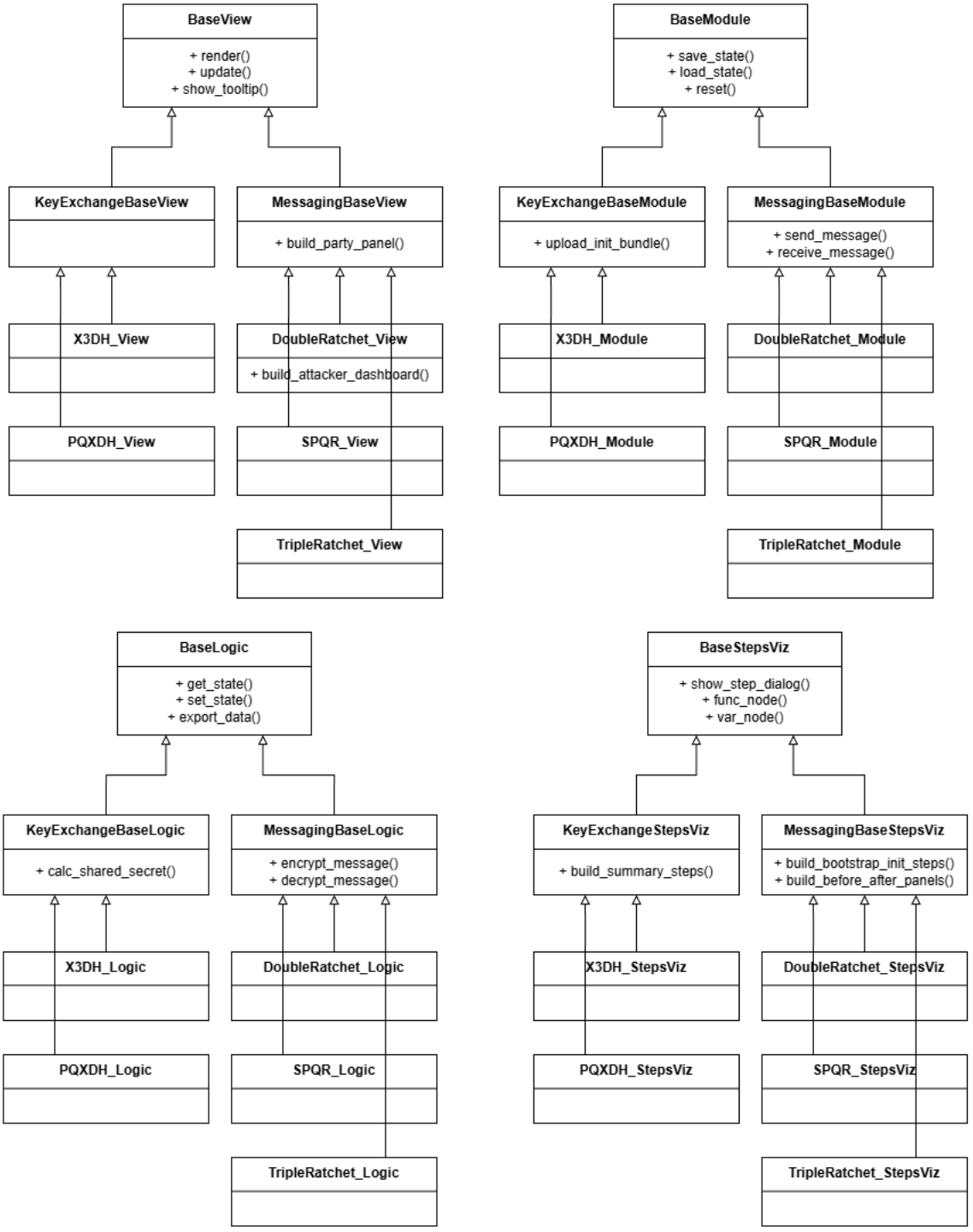
Keďže prezentované protokoly (napríklad tradičný X3DH a postkvantový PQXDH, alebo Double a Triple Ratchet) vykazujú funkčné aj vizuálne významné prekrývania, implementácia je postavená na trojvrstvovej (three-tier) hierarchii tried, aby sa zabránilo duplikácii kódu:

1. **Základné triedy (Base Level):** V koreňovom adresári `modules/` sa nachádzajú najvšeobecnejšie základné triedy (`base_module.py`, `base_view.py`). Tieto triedy implementujú základné funkcie platné pre všetky moduly.

2. **Stredná abstrakčná vrstva (Intermediate Group Level):** Pre dve hlavné skupiny, ktoré sme už spomenuli v kapitole o návrhu, sme vytvorili po jednej abstrakčnej strednej triede. Nadradené triedy *Protokoly dohodnutia kľúčov* (Key Exchange) a *Protokoly odosielania správ* (Messaging) rozširujú podtriedy o špecifickú logiku charakteristickú pre danú skupinu. Napríklad nadradená trieda Messaging (`base_messaging_view`) už implementuje panely účastníkov, vstupné polia pre odosielanie správ a vizuálne kontajnery histórie kľúčov, keďže tieto sú vo všetkých protokoloch odosielania správ skoro identické.

3. **Konkrétne implementácie (Concrete Implementations):** Na najnižšej úrovni hierarchie sa nachádzajú samotné moduly (X3DH, PQXDH, Double Ratchet, SPQR, Triple Ratchet). Tieto triedy sa dedia (inheritujú) z príslušných medzitried, takže už musia implementovať len špecifickú kryptografickú matematiku, ktorá je pre ne charakteristická (napríklad výpočty Diffie-Hellman alebo posúvanie reťazcov KDF) a prípadné jedinečné vizuálne prvky (napríklad Attacker Dashboard v prípade Double Ratchet).

Tento architektonický prístup zaručuje, že zdrojový kód zostane prehľadný, ľahko testovateľný a v budúcnosti bude možné ho plynule rozširovať o nové protokoly (napríklad ďalšie postkvantové algoritmy). Štruktúru systému môžeme pozorovať aj na diagramu triedy č. 13.



Obr. 13: Diagram tried

5.3 Modul X3DH

Pri implementácii protokolu X3DH sme sa primárne opierali o oficiálnu špecifikáciu Signal [15]. Pri implementácii sme kládli hlavný dôraz nie na abstraktnú kryptografickú eleganciu kódu, ale na vizualizovateľnosť pre vzdelávacie účely. V súlade s tým bola

vnútorná logika funkcií navrhnutá tak, aby jednotlivé kroky boli na grafickom rozhraní ľahko sledovateľné, odlíšiteľné a reprodukovateľné.

Štruktúru modulu sme rozdelili do troch hlavných fáz v súlade s plánmi opísanými v kapitole 4.2.2.

Prvá fáza simuluje spojenie medzi Alice ako iniciátorkou komunikácie a sprostredkovateľským serverom. V tejto fáze môže používateľ vykonať počiatočné kroky, ako je generovanie počiatočného balíka kľúčov a jeho nahranie na server.

Aplikácia používa logické obmedzenia na zachovanie konzistencie procesu: počiatočný balík obsahujúci identifikačný kľúč sa môže vygenerovať len raz a operácie sú viazané na prísne poradie (napríklad nie je možné iniciovať nahratie bez existujúcich kľúčov). Softvér venuje zvýšenú pozornosť správe súboru OPK. Ak počet OPK uložených na serveri klesne na nízku úroveň, systém v okne stavu servera žltým zvýraznením a v prípade úplného nedostatku červeným zvýraznením upozorní používateľa na potrebu obnovenia kľúčov. Súčasne s tým dostane podobné vizuálne upozornenie aj ovládacie tlačidlo zodpovedné za generovanie nových OPK.

Ako súčasť simulácie sme zabudovali aj dve funkčné tlačidlá, pomocou ktorých je možné manuálne použiť jeden OPK zo zásoby na serveri Alice alebo Bob. To umožňuje preskúmať špeciálny prípad, keď na serveri nie je k dispozícii žiadny OPK zo strany Boba, takže Alice musí podľa protokolu vykonať len tri výpočty DH namiesto obvyklých štyroch, ako sme podrobne opísali v kapitole 1.3.1.

Druhá fáza obsahuje operácie, ktoré Alice vykonáva s využitím Bobovho kľúčového balíka na výpočet zdieľaného tajomstva. Proces začína vyžiadaním Bobovho súboru kľúčov (*Key Bundle*) od servera. Aplikácia spoľahlivo spracováva odpovede obsahujúce aj neobsahujúce OPK a vizualizácia presne odzrkadľuje oba prípady.

Po prijatí kľúčového zväzku Alice overí Bobov digitálny podpis a následne vygeneruje nový dočasný pár kľúčov DH (*Ephemeral Key*, EK). S týmito kľúčmi vykoná operácie DH podľa protokolu na určenie zdieľaného tajomstva (*Shared Secret*, SK). Ako posledný krok tejto fázy Alice vypočíta asociované dáta (*Associated Data* – AD), ktoré sa použijú v neskorších protokoloch (napríklad pri šifrovaní v rámci Double Ratchet).

V poslednej fáze Alice odošle inicializačnú správu, ktorá obsahuje metadáta potrebné na spustenie protokolu. Bob po prijatí správy vykoná svoje vlastné výpočty. Ak sú kľúče a podpisy platné, Bob získa rovnaké zdieľané tajomstvo (SK) a asociované dáta (AD) ako Alice, čím sa vytvoria základy bezpečného kanála.

V záujme užívateľského zážitku a sledovateľnosti zaznamenáva časová os na pravej strane rozhrania každú vykonanú akciu. To umožňuje užívateľovi získať aj na konci procesu komplexný prehľad o presnom poradí udalostí a priebehu protokolu.

5.4 Modul DoubleRatchet

Základom biznisovej logiky modulu Double Ratchet sú algoritmy a referenčné kódové príklady uverejnené v oficiálnej špecifikácii Signal, ktoré tvoria kostru kryptografických operácií. Pre správne fungovanie algoritmov je nevyhnutné vedenie záznamov o kryptografickom stave účastníkov, na tento účel slúžia triedy stavov definované v už spomínanom súbore `data_classes.py`, ktoré sme implementovali pomocou dekorátora `@dataclass` jazyka Python, čím sme zabezpečili transparentnú a efektívnu správu dátových štruktúr. Kryptografické pomocné funkcie, ktoré sú nevyhnutné pre fungovanie protokolu a v oficiálnej špecifikácii sú označené ako „externé“ funkcie, sme implementovali v súbore `modules/external.py`.

Okrem predvolených inicializačných metód uvedených v oficiálnej dokumentácii bola implementácia modulu rozšírená aj o vlastnú inicializačnú funkciu. Táto funkcia nastavuje počiatočný stav komunikujúcich strán na základe výsledkov modulu X3DH (Extended Triple Diffie-Hellman). Vďaka tejto integrácii prezentované riešenie nielen demonštruje fungovanie izolovaného algoritmu Double Ratchet, ale zahŕňa aj počiatočnú výmenu kľúčov a spoločných tajných kľúčov. Modul tak v praxi modeluje komplexné fungovanie celého klasického protokolu Signálu.

Na praktickú implementáciu kryptografických primitív sme použili už spomínanú knižnicu PyCryptodome. Na implementáciu schémy AEAD (Authenticated Encryption with Associated Data) sme zvolili algoritmus ChaCha20-Poly1305, ktorý je tiež natívne dostupný v balíku `Crypto`.

Jednou z hlavných zvláštností a vzdelávacích hodnôt modulu je integrovaná perspektíva útočníka (Attacker perspective) a s ňou organicky prepojený grafický rozhranie, tzv. „Attacker dashboard“ (panel útočníka). S jeho pomocou je možné vizuálne demonštrovať odolnosť algoritmu a analyzovať, aký vplyv má prípadné kompromitovanie kľúča (napríklad únik aktuálnych stavových kľúčov) na bezpečnosť budúcej a minulej komunikácie (Forward Secrecy a Post-Compromise Security).

5.4.1 Logika a architektúra perspektívy útočníka

Perspektíva útočníka slúži na interaktívnu analýzu a simuláciu bezpečnostných vlastností kryptografického protokolu Double Ratchet. Jej primárnym cieľom nie je len statické zobrazenie stavu, ale aj demonštrovanie toho, ktoré správy je útočník schopný úspešne dešifrovať v prípade kompromitácie rôznych tajných kľúčov.

Analytická logika modulu sa opiera o tri hlavné sady vstupných údajov:

- Aktuálny stav relácie (`DoubleRatchetState`) obsahujúci kompletnú históriu kľúčov a protokol správ komunikujúcich strán

- Zoznam správ odoslaných v sieti
- Slovník kľúčov, ktoré používateľ označil za kompromitované (známe) z pohľadu útočníka

Počas simulácie funkcia prechádza históriou kľúčov oboch účastníkov a vymenúva potenciálne kompromitovateľné tajomstvá. Tieto kryptografické prvky sú nasledujúce:

- Koreňové kľúče (Root Keys, RK) z histórie kľúčov účastníkov
- Reťazové kľúče odosielateľa a príjemcu (Chain Keys, CKs/CKr)
- Súkromné kľúče Diffie-Hellman (DH) a verejné kľúče, ktoré sú s nimi spojené
- Jedinečné kľúče správ (Message Keys, MK) pochádzajúce zo záznamu správ

Všetky zozbierané kľúče systém doplní o kontextové metadáta, napríklad o entitu vlastníka a smer siete. Tieto anotácie sú kľúčové pre neskoršie filtrovanie a zapojenie derivácií.

Jadrom analýzy dešifrovania je nasledujúci algoritmus, ktorý deterministicky skúma kompromitované dáta z hľadiska dešifrovateľnosti správ. Pokusy o odvodenie prebiehajú s cieľom optimalizácie zdrojov v troch vrstvách s postupne rastúcou zložitosťou:

1. **Dešifrovanie na základe priameho kľúča správy (MK):** Najmenšia a najjednoduchšia vetva z hľadiska výpočtových nárokov. Ak útočník priamo vlastní kľúč správy patriaci k testovanej správe, systém okamžite dešifruje šifrovaný text a správa dostane status *decryptable (dešifrovateľná)*.
2. **Derivácia založená na reťazovom kľúči (CK):** V priebehu tohto procesu, ak došlo k kompromitácii reťazového kľúča, systém vygeneruje správové kľúče patriace k reťazcu opakovaným spustením kľúčovej derivácie `KDF_CK`. Počet krokov derivácie určuje rozdiel medzi počítadlom v hlavičke a počiatočným indexom kľúča. S cieľom zachovať vysoký výpočtový výkon systém ukladá do vyrovnávacej pamäte (caching) nadbytočné iterácie `KDF_CK`.
3. **Iteratívny model kompromitácie DH:** Najzložitejšia časť modulu modeluje prípad, keď útočník získa prístup k koreňovému kľúču (RK) a k sérii súkromných kľúčov DH. Systém napodobňuje fungovanie skutočného protokolu Double Ratchet a vytvorí sekvenciu kľúčov DH, potom prostredníctvom iteratívneho volania operácií `DH` a `KDF_RK` postupne odvodí nové reťazové kľúče a z nich odvodené kľúče správ.

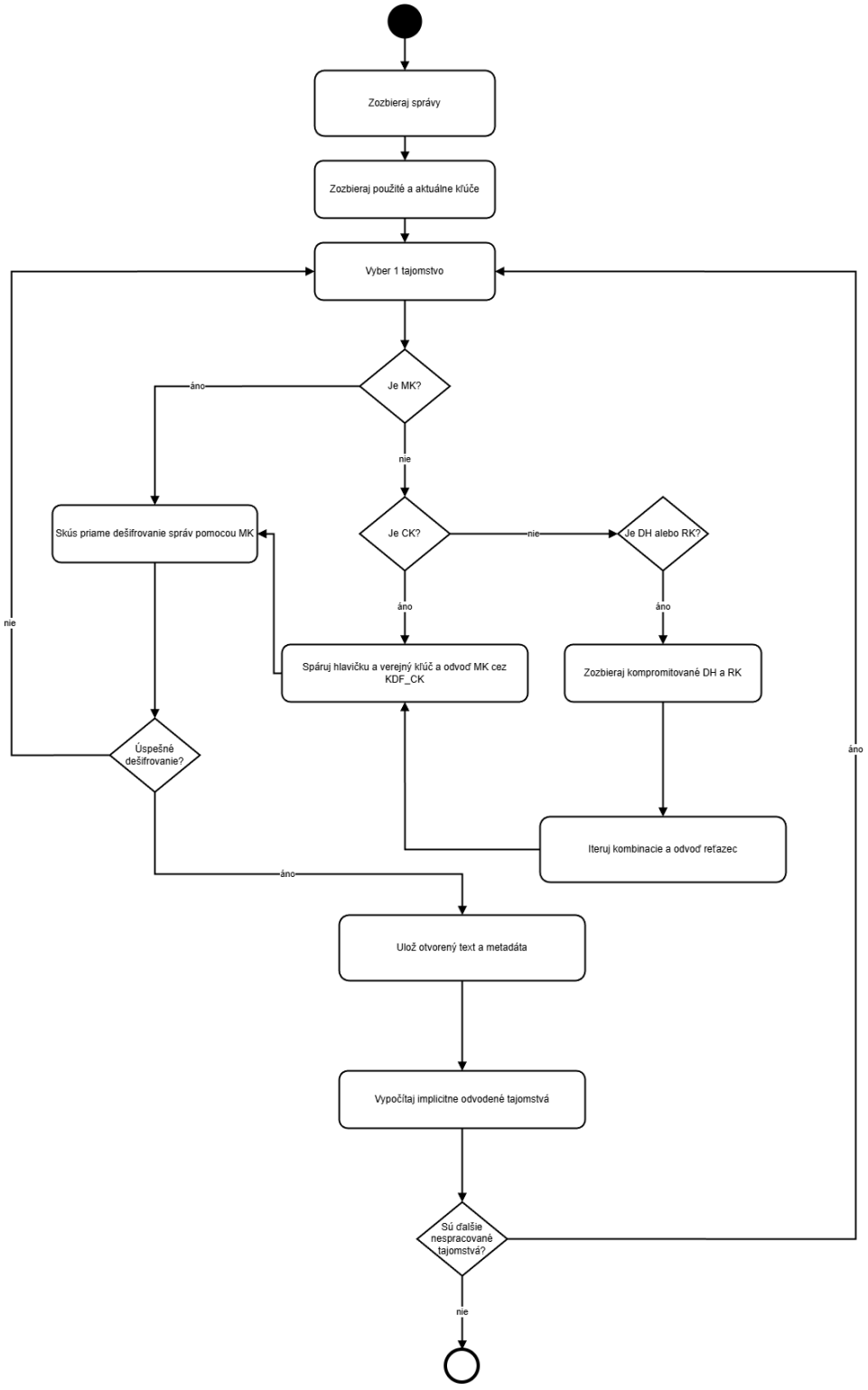
Systém dopĺňa každú úspešnú dešifrovaciu udalosť o údaje o použití (usage) a presnú trasu kompromitácie. Tieto údaje o sledovaní robia analýzu transparentnou. Okrem toho logika používateľského rozhrania určuje tranzitívne vplyvy, t. j. Odhľadu na množinu implicitne známych (implied) tajomstiev, ktoré logicky vyplývajú z kompromitovaných kľúčov. Prostredníctvom týchto implikácií rozhranie rozpoznáva odvodené reťazcové kľúče, identifikuje prvky MK dešifrované zo správ a nájde aj zrkadlové reťazcové kľúče komunikačného kanála v opačnom smere. Na grafickom rozhraní dostane používateľ vizuálnu spätnú väzbu o týchto súvislostiach: systém štruktúruje tajomstvá podľa typu a účastníka, pričom implicitne odvodené kľúče UI automaticky označuje zvýraznením v oranžovej (amber) farbe. Celkový priebeh tohto analytického procesu a postupnosť derivácie tajomstiev schematicky znázorňuje diagram aktivít na obrázku 14.

Predstavená simulácia útočníka koncepčne úzko zapadá do formálnych modelov hrozieb používaných pri analýze moderných protokolov bezpečného zasielania správ. Zatiaľ čo klasický model Dolev-Yao [38] sa zameriava predovšetkým na aktívneho sieťového útočníka (ktorý je schopný zachytiť, modifikovať alebo zadržať každú správu), logika Attacker Dashboard modeluje špecifickejšieho, takzvaného pasívneho útočníka, ktorý predpokladá kompromitáciu pamäte zariadení (state leakage).

Tento prístup v praxi vizualizuje formálne bezpečnostné hry (security games) definované pre protokol Signal Cohnom-Gordonom a spolupracovníkmi [39] a Alwenom a spolupracovníkmi [40]. V uvedených teoretických rámcoch je útočník (adversary) schopný zisťovať vnútorný stav účastníkov prostredníctvom takzvaných orákulumov. Vyvinutý modul prenáša tento abstraktný teoretický rámec do interaktívnej, praktickej simulácie. Vďaka tomu, že si používateľ môže voľne vybrať kompromitované tajné kľúče (MK, CK, RK a DH kľúče), systém explicitne ukazuje, do akej miery sú tieto hypotetické operácie schopné retrospektívne narušiť Forward Secrecy a po koľkých krokoch zabudovaný mechanizmus protokolu na samoopravu (Post-Compromise Security) opäť vylúči útočníka z komunikácie.

5.5 Modul PQXDH

Pri implementácii modulu PQXDH (*Post-Quantum Extended Triple Diffie-Hellman*) sme sa vo veľkej miere opierali o architektúru modulu X3DH, čím sme reflektovali logiku pôvodného protokolu, podrobne rozobranú v kapitole 1.3.2. Keďže PQXDH predstavuje variant protokolu X3DH rozšírený o postkvantové prvky, v aplikácii sme zachovali obdobnú modulárnu hierarchiu. Značná časť krokov a výpočtov je identická, pričom samotné rozšírenie spočíva v integrácii nových, voči kvantovým počítačom odolných mechanizmov zapuzdrenia kľúča (KEM).



Obr. 14: Diagram aktivít pre dešifrovací algoritmus z perspektívy útočníka

Na realizáciu postkvantových funkcií KEM sme využili knižnicu `pqcrypto`, konkrétne sme aplikovali algoritmus `ml_kem_1024`. Táto voľba zaručuje, že vzdelávací softvér plne reflektuje aktuálne trendy a prebiehajúce procesy štandardizácie v oblasti kryptografie.

5.6 Modul SPQR

Používateľské rozhranie (UI) modulu SPQR (*Sparse Post-Quantum Ratchet*) zámerne kopíruje dizajn modulu Double Ratchet, čo používateľovi uľahčuje identifikáciu parallel a rozdielov medzi oboma protokolmi. Pri implementácii sme vychádzali z referencií a ukážok kódu dostupných v oficiálnej dokumentácii [4, 19].

Pri realizácii mechanizmu SPQR sme modelovali protokol inkrementálneho KEM (*Incremental KEM*) opísaný v oficiálnej špecifikácii [19]. Vzhľadom na to, že v čase písania tejto práce (apríl 2026) nebola k dispozícii žiadna komplexná a stabilná programová knižnica pre protokol ML-KEM Braid, rozhodli sme sa modul implementovať prostredníctvom simulačnej vrstvy. V praxi to znamená, že sme v súbore `external.py` vytvorili pomocné funkcie, ktoré presne kopírujú rozhrania (vstupné parametre a návratové hodnoty) definované v pôvodnej špecifikácii. Táto architektúra zaručuje bezproblémovú rozšíriteľnosť do budúcnosti: akonáhle bude dostupná natívna knižnica implementujúca tento protokol, jej integrácia do aplikácie si vyžiada len minimálne úpravy.

Počas implementácie sme v oficiálnej špecifikácii [19] identifikovali určité nepresnosti týkajúce sa popisu procesu generovania kľúčov. Zatiaľ čo jedna časť dokumentácie definuje štruktúru `KeyGen(randomness)` → `(dk, ek_header, ek_vector)`, ukážka kódu pre stav *KeysUnsampled* a následná manipulácia s hlavičkou sa od tohto prístupu odlišujú:

```
# 1. Rozhranie podľa textovej definície:
# KeyGen(randomness) -> (dk, ek_header, ek_vector)

# 2. Ukážka kódu nachádzajúca sa v stave KeysUnsampled:
(dk, ek_seed, ek_vector) = KEM.KeyGen()
hek = SHA3-256(ek_seed || ek_vector)
header = ek_seed || hek

# 3. Dekompozícia hlavičky v inej časti dokumentácie:
ek_seed = header[:32]
```

Výpis kódu 1: Inkonzistencia medzi definíciou a ukážkami kódu v dokumentácii

Keďže špecifikácia na viacerých miestach referuje na `ek_seed` ako na prvých 32 bajtoch hlavičky (`header`), nami vyvinutá simulovaná funkcia `KeyGen()` priamo vracia trojicu `(dk, ek_header, ek_vector)`. Toto riešenie v sebe už integruje logiku zostavenia hlavičky, čím sa zachováva konzistencia s ostatnými časťami protokolu. Inicializácia tohto modulu (teda vytvorenie počiatočného stavu medzi komunikujúcimi stranami) prebieha prostredníctvom protokolu PQXDH, analogicky k tomu, ako modul Double Ratchet stavia na základoch X3DH.

S ohľadom na vzdelávací charakter aplikácie a zachovanie jej prehľadnosti nie je fragmentácia („chunking“) postkvantových správ obmedzená na pevnú veľkosť. V zdrojovom kóde je zadefinovaná modifikovateľná konštanta, ktorá určuje maximálny počet fragmentov, a na základe nej systém dynamicky rozdeľuje dáta. Týmto prístupom sa predchádza situácii, kedy by používateľ musel preklikať desiatky správ len na to, aby vygeneroval nové zdieľané tajomstvo a inicioval zmenu stavu. V kontexte edukačného softvéru mala logická sledovateľnosť procesov vyššiu prioritu ako prísna simulácia reálnych sieťových obmedzení a šírky pásma.

Na záver je v tomto module kľúčová podpora ukladania a načítavania stavu. Zatiaľ čo v jednoduchších moduloch možno väčšinu scenárov reprodukovať niekoľkými kliknutiami, komplexný stavový automat SPQR a mechanizmus fragmentácie vyžadujú na dosiahnutie zaujímavých kryptografických situácií značné množstvo interakcií. Implementovaná správa stavu tak používateľovi umožňuje okamžitú vizualizáciu a štúdium týchto pokročilých situácií bez nutnosti zdĺhavého manuálneho preklikávania.

5.7 Modul TripleRatchet

Modul Triple Ratchet predstavuje finálnu integračnú vrstvu aplikácie, v ktorej dochádza k hybridizácii klasických a postkvantových kryptografických mechanizmov. Ako bolo detailne popísané v teoretickej časti (kapitola 1.11), tento mechanizmus kombinuje vlastnosti algoritmu Double Ratchet s postkvantovým protokolom SPQR, čím vytvára robustné riešenie odolné voči hrozbám budúcich kvantových počítačov. Používateľské rozhranie tohto modulu zámerne zachováva kontinuitu s dizajnom modulu SPQR. Hlavný rozdiel spočíva v paneli účastníkov (Participant panel), kde sú okrem post-kvantových parametrov vizualizované aj aktuálne kľúče a stavy prislúchajúce klasickému Double Ratchet mechanizmu. Táto paralelná vizualizácia umožňuje používateľovi v reálnom čase sledovať, ako obe vrstvy protokolu koexistujú a nezávisle od seba aktualizujú svoje vnútorné stavy.

Pre počiatočné nadviazanie spojenia sme implementovali protokol PQXDH. Proces inicializácie v našej implementácii prebieha nasledovne:

1. Výsledkom protokolu PQXDH je zdieľané tajomstvo (*shared secret*), ktoré pomocou funkcie HKDF expandujeme na dvojnásobok jeho pôvodnej dĺžky.

2. Takto získaný binárny blok rozdelíme na dve rovnaké časti.

3. Prvú polovicu používame ako vstupný vzdialený kľúč (SK) pre inicializáciu modulu Double Ratchet.

4. Druhú polovicu používame na nastavenie počiatočného stavu modulu SPQR.

Samotná logika modulu Triple Ratchet funguje ako orchestrátor medzi oboma kryptografickými vrstvami. Pri odosielaní alebo prijímaní správy prebiehajú výpočty v oboch protokoloch paralelne a nezávisle na pozadí:

- **Klasická vrstva (DR):** Vygeneruje čiastočný kľúč ec_{mk} a príslušnú hlavičku správy.
- **Postkvantová vrstva (SPQR):** Vygeneruje post-kvantový čiastočný kľúč pq_{mk} a jemu prislúchajúcu hlavičku.

Kľúčovým krokom hybridizácie je finálna derivácia kľúča správy. Čiastkové kľúče ec_{mk} a pq_{mk} vstupujú do KDF funkcie, ktorej výstupom je výsledný symetrický kľúč mk. Tento kľúč sa následne použije na samotné zašifrovanie alebo dešifrovanie správy. Týmto spôsobom je bezpečnosť komunikácie zaručená dovtedy, kým je bezpečná aspoň jedna z použitých metód (klasická eliptická kryptografia alebo postkvantové mriežky).Finálna hlavička správy vzniká spojením (*concatenation*) hlavičiek oboch protokolov. V počiatočnej fáze komunikácie, kedy Alica ešte nemá potvrdenie o inicializácii na strane Boba, je táto štruktúra doplnená aj o hlavičku protokolu PQXDH, čím je zabezpečená správna synchronizácia všetkých kryptografických komponentov.

Vďaka modulárnej architektúre aplikácie bola implementácia modulu Triple Ratchet relatívne priamočiara. Modul využíva už existujúce abstrakcie a funkcie definované v moduloch Double Ratchet a SPQR. Triple Ratchet v praxi nevykonáva žiadnu komplexnú vlastnú výpočtovú logiku okrem spomínanej KDF kombinácie kľúčov a správy hlavičiek.

6 Testovanie a vyhodnotenie požiadaviek

6.1 Testovanie aplikácie

Testovanie hotovej aplikácie sme rozdelili do viacerých fáz s rôznou zložitosťou a zameraním. V každej fáze sme skúmali iný aspekt softvéru, čím sme zabezpečili spoľahlivú fungovanie systému.

Základnou podmienkou správneho fungovania aplikácie je, aby implementované a používané kryptografické algoritmy a funkcie fungovali bezchybne vo všetkých moduloch. Na zabezpečenie toho sme pre všetky podporné funkcie patriace k modulom vytvorili jednotkové testy (*unit test*), čím sme zaručili, že ich výstupy vždy spĺňajú očakávania. Vďaka architektúre MVC (*Model-View-Controller*) aplikácie je backendová logika a procesy jasne oddelené od používateľského rozhrania. Toto štrukturálne rozhodnutie výrazne uľahčilo izolované testovanie funkcií bez komplikácií. Na implementáciu a spúšťanie testov sme použili knižnicu `Pytest` [41], ktorá poskytla prehľadný rámec na definovanie testovacích prípadov, ich automatizované spúšťanie a umožnila odfiltrovanie prípadných logických chýb v ranom štádiu.

Jednou z hlavných úloh aplikácie je pomocou vizualizácie vysvetliť fungovanie a štruktúru protokolu Signal, ako aj samostatnú a komplexnú úlohu mechanizmov v ňom použitých. Keďže v predchádzajúcej fáze sme už pomocou jednotkových testov overili správnosť kryptografických funkcií (že pracujú so správnymi hodnotami a vracajú presné výsledky), hlavným cieľom v tejto fáze bola validácia vizualizácie. Pri tom sme ručne prešli všetky možné scenáre, ktoré vyžadujú špecifickú vizualizáciu, a potom sme skontrolovali, či rozhranie verne odzrkadľuje kroky protokolu.

Takýmto scenárom je napríklad situácia, keď prijímajúca strana identifikuje v hlavičke správy nový verejný Diffie-Hellman (DH) kľúč. Aplikácia musí túto situáciu zvládnuť a vizualizácia ju musí krok za krokom znázorniť: príjemca vytvorí nový prijímací reťazec (*Receiving Chain*) a príslušný kľúč správy, potom vygeneruje vlastný nový pár kľúčov DH. Pomocou toho inicializuje svoj vlastný nový odosielací reťazec (Sending Chain), ktorý neskôr použije na odvodenie kľúčov správ používaných na šifrovanie svojich vlastných správ. Tieto kroky nazývame aj „kroky DH ratchet“.

Toto je len jeden zo scenárov spomedzi mnohých prípadov skúmaných v module Double Ratchet. V každej situácii sme skontrolovali, či zobrazená vizualizácia úplne obsahuje kroky vykonané odosielateľom alebo príjemcom, a tiež sme vyhodnotili čitateľnosť a logickú sledovateľnosť zobrazenia. Ďalším, zložitejším prípadom je napríklad situácia, keď príjemca ešte nedostal všetky predchádzajúce správy, ktoré odosielateľ zašifroval pomocou predchádzajúceho reťazca, a novo prichádzajúca správa už patrí do

nového reťazca. V takomto prípade musí príjemca nielen vykonať kroky DH ratchetu, ale musí tiež spracovať kľúče patriace k chýbajúcim (*out-of-order*) správam. Vizualizácia musí tieto procesy znázorniť prehladne, aby bolo pre používateľa jasné, ako protokol rieši asynchrónnosť a mechanizmus na zvládanie sieťových oneskorení. Keďže poskytovanie údajov o funkciách na pozadí bolo preukázateľne presné, v tomto prípade sme skúmali výlučne existenciu a umiestnenie prvkov používateľského rozhrania a správnosť zobrazenia procesu.

Keďže hlavným zameraním práce je predstavenie mechanizmu Double Ratchet a tým aj fungovania protokolu Signal, tomuto modulu sme sa venovali najpodrobnejšie. Tu sme implementovali perspektívu útočníka a príslušné rozhranie (*Attacker Dashboard*). Hoci základné funkcie potvrdili jednotkové testy, logické vrstvy rozhrania, ktoré na seba nadväzujú, si vyžadovali ďalšie preskúmanie. Hoci sme počas vývoja neustále vykonávali manuálne kontroly, vzhľadom na zložitosť modulu jednoduché ručné testovanie nestačilo na to, aby sme mohli s istotou vyhlásiť, že všetky prvky perspektívy útočníka fungujú bezchybne. Z tohto dôvodu sme vytvorili samostatnú sadu testov (*test suite*), ktorá testovala výlučne tento modul. V tomto prostredí sme napríklad testovali, či aplikácia úspešne zhromažďuje všetky použité tajné kľúče aj po niekoľkých stovkách výmen správ, alebo či identifikácia a označenie implikovaných (odvoditeľných) tajných kľúčov prebieha správne. Napísali sme samostatné testovacie prípady na kontrolu správneho spracovania kompromitovaných kľúčov a súborov kľúčov.

Všetko sme ručne overili aj na menšom počte prípadov, ale pomocou automatizovaných testov sme mohli zaručiť stabilitu modulu oveľa rýchlejšie a s väčšou presnosťou.

V záverečnej fáze testovania sme skúmali, či hotový softvér bezchybne spĺňa požiadavky, ktoré sme formulovali v predchádzajúcich kapitolách.

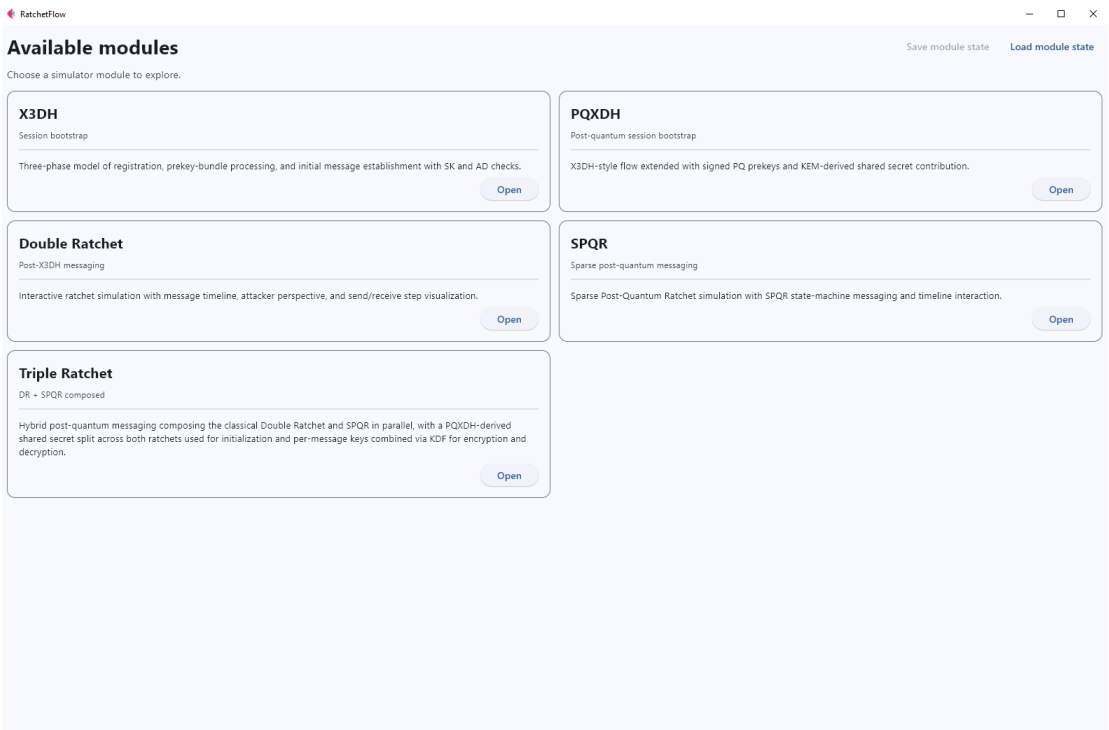
6.2 Požiadavky na aplikáciu

V tejto časti porovnáme výkon aplikácie s požiadavkami stanovenými v kapitole 3.

6.2.1 Funkčné požiadavky

- **Výber edukačného modulu:** Aplikácia ihneď po spustení ponúka možnosť výberu, ako je to znázornené na obrázku 15.
- **Správa stavu (Scenáre):** Systém podporuje ukladanie a načítanie aktuálneho stavu, čím zabezpečuje pokračovanie v práci.
- **Riadenie komunikácie:** Používateľ má plnú kontrolu nad odosielaním správ (obrázok 16). Je možné vybrať odosielateľa aj obsah správy.

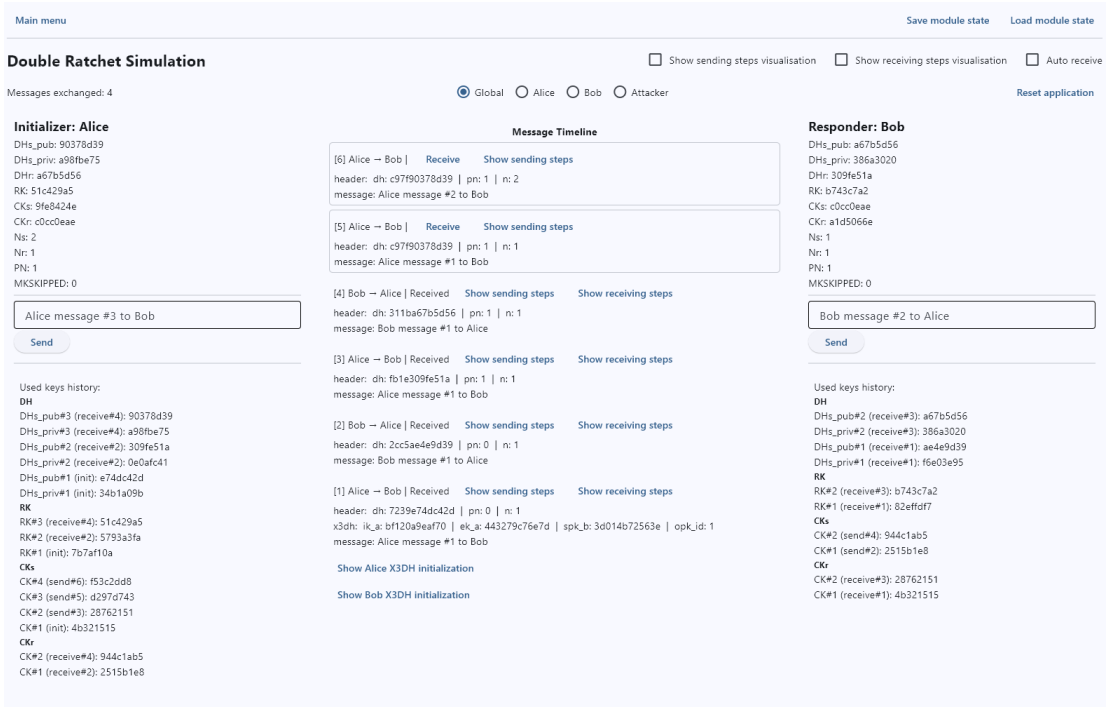
- **Zobrazenie stavu komunikácie:** Rozhranie dynamicky zobrazuje aktuálny vnútorný stav účastníkov a odoslané správy, čo ilustruje aj obrázok 16.
- **Detailná vizualizácia výpočtov:** Vizuálne rozloženie zložitých výpočtových krokov, ako je znázornené na obrázku 17, pomáha pri pochopení.
- **Správa histórie kľúčov:** Všetky použité kľúče a tajné kódy je možné vyhľadať v histórii pod panelmi účastníkov (obrázok 16).
- **Perspektívy používateľa (Úrovne viditeľnosti):**Softvér podporuje prepínanie medzi pohľadmi (Alice, Bob, Global). Špeciálny útočnícky pohľad dostupný v module Double Ratchet je znázornený na obrázku 18, ktorého fungovanie sme podrobne opísali v kapitole 5.4.1.



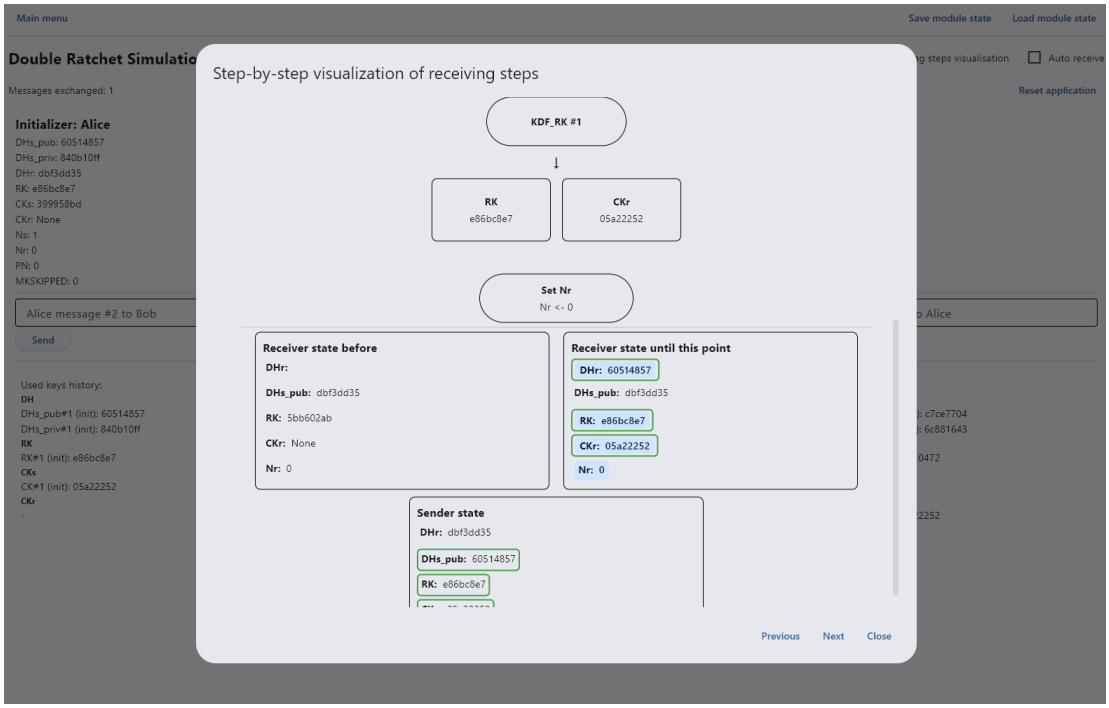
Obr. 15: Modulový výberový panel po spustení

6.2.2 Nefunkcionálne požiadavky

- **Kryptografická exaktnosť a presnosť:** Spomínané *unit testy* potvrdzujú kryptografickú presnosť.
- **Zameranie na používateľský zážitok (Usability) a ergonómiu:** Túto požiadavku spĺňa dizajnové rozhodnutie, podľa ktorého sa na rozhraní zobrazujú

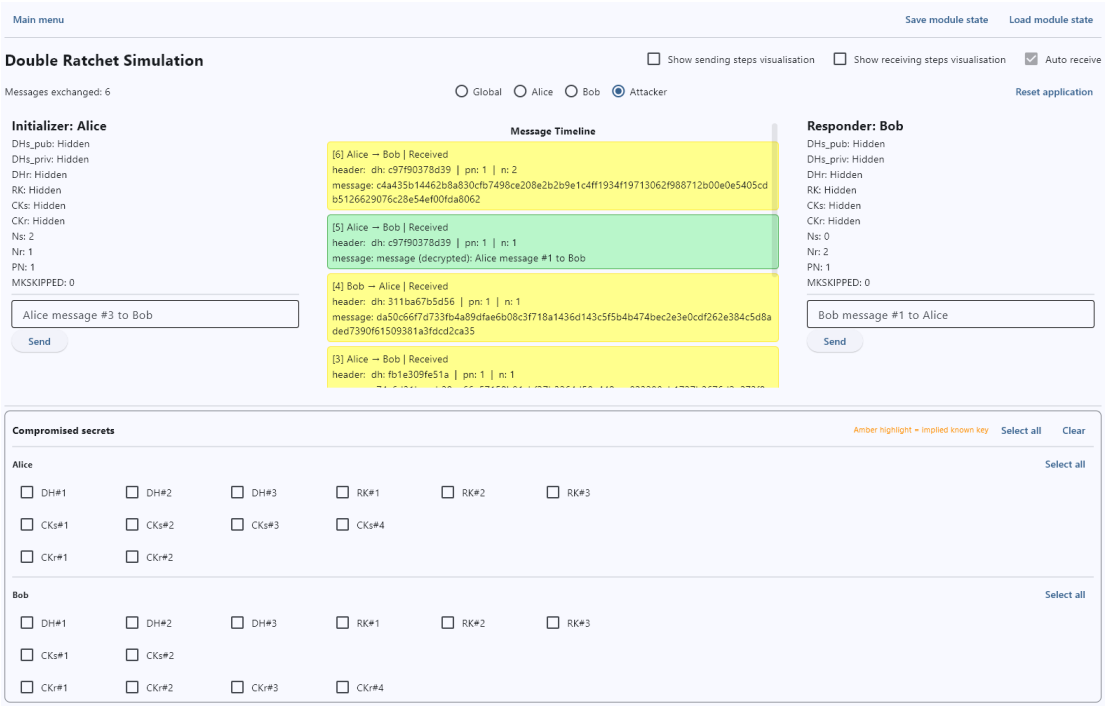


Obr. 16: Modul na odosielanie správ: odosielanie správ, história kľúčov a stav



Obr. 17: Vizualizácia krok za krokom

len posledné znaky dlhých hexadecimálnych kľúčov a tajných údajov. To výrazne zvyšuje prehľadnosť a uľahčuje vizuálnu identifikáciu údajov pre používateľa bez toho, aby sme obrazovku preťažili nadbytočnými informáciami. Toto isté cieľ slúži



Obr. 18: Útočníkova perspektíva a riadiaci panel

aj interaktívny systém označovania použitý v moduloch o dohodu kľúčov, ktorý vizuálnymi indikátormi vedie používateľa cez navzájom na seba nadväzujúce kroky protokolu, čím objasňuje kontext aktuálnej operácie.

- **Vizuálna spätná väzba:** Systém poskytuje okamžitú vizuálnu spätnú väzbu, napríklad farebným označením správ, ktoré môže útočník prelomiť/dešifrovať (18. obrázok).
- **Obmedzenie účelu aplikácie (Bezpečnostný štandard):** Z dôvodu vzdelávacích cieľov aplikácia zámerne ukladá také údaje (napr. súkromné kľúče v plaintextovom formáte), ktoré by v ostrých systémoch predstavovali bezpečnostné riziko.
- **Modulárna architektúra:** Model MVC a trojvrstvová abstrakcia zabezpečujú rozšírovateľnosť systému.
- **Platformová dostupnosť a architektúra:** Použitie jazyka Python a knižnice Flet zaručuje platformovo nezávislé spustenie a možnosť vytvárania samostatných binárnych súborov.

Záver

Hlavným cieľom našej diplomovej práce bolo vytvoriť edukačnú aplikáciu, ktorá uľahčí pochopenie fungovania protokolu Signal pre študentov a záujemcov s predchádzajúcimi znalosťami z kryptografie. V závere vývoja môžeme konštatovať, že sme tento cieľ úspešne splnili. Nami vyvinutý softvér sa neobmedzuje len na prezentáciu tradičného protokolu Signal (zahŕňajúceho mechanizmy X3DH a Double Ratchet), ale implementovali sme doň aj najmodernejšiu postkvantovú verziu, ktorá vychádza z protokolov PQXDH, SPQR a Triple Ratchet.

Za najkomplexnejšiu a z pedagogického hľadiska najnáročnejšiu časť klasického protokolu považujeme mechanizmus Double Ratchet. Zatiaľ čo protokoly na výmenu kľúčov nasledujú deterministickú a striktnú postupnosť krokov, asynchrónne fungovanie mechanizmu Double Ratchet (ktorého úlohou je zvládnuť reálne sieťové oneskorenia a prípadné straty paketov) je pre študentov ťažšie uchopiteľné. V našej aplikácii sme sa preto zamerali na detailnú vizualizáciu synchronizácie medzi stranami krok za krokom, ako aj na riešenie výpadkov správ a dodatočné spracovanie oneskorených paketov. Osobitný dôraz sme kládli na ďalší základný pilier protokolu, ktorým je PCS. Aby sme túto kľúčovú vlastnosť dokázali názorne a do hĺbky analyzovať, navrhli a integrovali sme do aplikácie špeciálne rozhranie pre simuláciu kompromitácie. Pomocou ktorého môžu používatelia interaktívne testovať, ako protokol obnovuje svoju bezpečnosť (tzv. samoopravný mechanizmus / self-healing) po prípadnom kompromitovaní kľúčov a aký to má vplyv na dôvernosť minulých i budúcich správ.

V rámci softvéru sme taktiež poskytli komplexný pohľad na vývoj mechanizmov dohody kľúčov. Vďaka nášmu modulárnemu dizajnu je možné jasne pozorovať prepojenie medzi protokolom X3DH, založeným na eliptických krivkách, a jeho verziou rozšírenou o postkvantový komponent, PQXDH. Pomocou aplikácie sme ilustrovali aj jednu z najväčších výziev postkvantových algoritmov, ktorou sú neúmerne veľké rozmery kľúčov.

Pri technologickej implementácii sme sa rozhodli pre využitie jazyka Python v kombinácii s knižnicou Flet, čo nám umožnilo dosiahnuť mimoriadne flexibilný výsledok. Náš softvér je možné spustiť nielen ako tradičnú lokálnu aplikáciu vyžadujúcu virtuálne prostredie, ale vďaka tejto voľbe sme ho sprístupnili aj ako online webovú aplikáciu a taktiež ako samostatný, platformovo nezávislý desktopový program, ktorý nevyžaduje inštaláciu žiadnych dodatočných programovacích knižníc.

Na záver môžeme zhrnúť, že výsledkom našej práce je komplexný a praktický nástroj, ktorý vypĺňa medzeru v oblasti výučby kryptografie. Študentom sme poskytli interaktívnu vzdelávaciu platformu a pedagógom nástroj na efektívnu demonštráciu. Vďaka integrovaným funkciám ukladania a načítania stavu si môžu vyučujúci vopred

pripraviť prednášky modelované s reálnymi kryptografickými hodnotami. Môžeme s hrdosťou vyhlásiť, že nami vyvinutá Edukačná aplikácia o Signal protokole bezvýhradne spĺňa stanovené ciele a úspešne vizualizuje zložité kryptografické procesy bežiace na pozadí moderných komunikačných systémov.

Literatúra

1. COHN-GORDON, K.; CREMERS, C.; DOWLING, B.; GARRATT, L.; STEBILA, D. *A Formal Security Analysis of the Signal Messaging Protocol* [online]. 2017. [cit. 2026-02-24]. Tech. spr. ACM SIGSAC Conference on Computer a Communications Security. Dostupné z: `https://doi.org/10.1145/3133956.3134063`.
2. RIVEST, R. L.; SHAMIR, A.; ADLEMAN, L. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*. 1978, **21**(2), 120–126.
3. KRAWCZYK, H.; ERONEN, P. *RFC 5869: HMAC-based Extract-and-Expand Key Derivation Function (HKDF)* [online]. 2010. [cit. 2026-02-24]. Tech. spr. IETF. Dostupné z: `https://www.rfc-editor.org/rfc/rfc5869`.
4. MARLINSPIKE, M.; PERRIN, T. *The Double Ratchet Algorithm* [online]. 2016-11. [cit. 2026-02-24]. Tech. spr. Signal Messenger. Dostupné z: `https://signal.org/docs/specifications/doubleratchet/`.
5. ALWEN, J.; CORETTI, S.; DODIS, Y. Post-Compromise Security. In: *Innovations in Theoretical Computer Science (ITCS)*. 2021.
6. DIFFIE, W.; HELLMAN, M. E. New Directions in Cryptography. *IEEE Transactions on Information Theory*. 1976, **22**(6), 644–654.
7. SHOUP, V. *A Computational Introduction to Number Theory and Algebra*. 2nd. Cambridge University Press, 2009.
8. KOBLITZ, N. Elliptic curve cryptosystems. *Mathematics of Computation*. 1987, **48**(177), 203–209.
9. MILLER, V. S. Use of Elliptic Curves in Cryptography. In: *Advances in Cryptology – CRYPTO '85*. 1986, s. 417–426.
10. KRAWCZYK, H.; ERONEN, P. *HMAC-based Extract-and-Expand Key Derivation Function (HKDF)* [RFC 5869]. RFC Editor, 2010. Request for Comments, č. 5869. Dostupné z DOI: `10.17487/RFC5869`.
11. BELLARE, M.; NAMPREMPRE, C. Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm. In: *ASIACRYPT 2000*. 2000, s. 531–545.
12. DWORKIN, M. *NIST SP 800-38D: Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC* [online]. 2007. [cit. 2026-02-24]. Tech. spr. National Institute of Standards a Technology. Dostupné z: `https://csrc.nist.gov/publications/detail/sp/800-38d/final`.

13. NIR, Y.; LANGLEY, A. *RFC 8439: ChaCha20 and Poly1305 for IETF Protocols* [online]. 2018. [cit. 2026-02-24]. Tech. spr. IETF. Dostupné z: https://www.rfc-editor.org/rfc/rfc8439.

14. CRAMER, R.; SHOUP, V. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*. 2003.

15. MARLINSPIKE, M.; PERRIN, T. *The X3DH Key Agreement Protocol* [online]. 2016-11. [cit. 2026-02-24]. Tech. spr. Signal Messenger. Dostupné z: https://signal.org/docs/specifications/x3dh/.

16. MESSENGER, S. *The PQXDH Key Agreement Protocol* [online]. 2023-09. [cit. 2026-02-24]. Tech. spr. Signal Messenger. Dostupné z: https://signal.org/docs/specifications/pqxdh/.

17. PERRIN, T. *The XEdDSA and VXEdDSA Signature Schemes* [online]. 2016-10. [cit. 2026-02-24]. Tech. spr. Signal Messenger. Dostupné z: https://signal.org/docs/specifications/xeddsa//.

18. MARLINSPIKE, M. *Safety number updates* [online]. Signal Messenger, 2016-11. [cit. 2026-02-21]. Dostupné z: https://signal.org/blog/safety-number-updates/.

19. SCHMIDT, R. *The ML-KEM Braid Protocol* [online]. 2025-09. [cit. 2026-02-24]. Tech. spr. Signal Messenger. Dostupné z: https://signal.org/docs/specifications/mlkembraid/.

20. MARLINSPIKE, M.; PERRIN, T. *The Double Ratchet Algorithm* [online]. 2016-11. [cit. 2026-02-24]. Tech. spr. Signal Messenger. Dostupné z: https://signal.org/docs/specifications/doubleratchet/#the-sparse-post-quantum-ratchet/.

21. POSITIVE-INTENTIONS.COM. *Signal Protocol in the Browser (demo)* [online]. [cit. 2026-02-24]. Dostupné z: https://signal.positive-intentions.com/.

22. XORON. *signal-protocol* [online]. [cit. 2026-03-31]. Dostupné z: https://github.com/positive-intentions/signal-protocol.

23. VASTRIK. *End-to-end Encryption* [online]. [cit. 2026-02-24]. Dostupné z: https://vas3k.com/blog/end_to_end_encryption/.

24. COMPUTERPHILE. *Computerphile* [online]. n.d. [cit. 2026-02-24]. Dostupné z: https://www.youtube.com/Computerphile. YouTube channel.

25. XARGS.ORG. *TLS 1.2* [online]. [cit. 2026-02-24]. Dostupné z: https://tls12.xargs.org/.

26. XARGS.ORG. *TLS 1.3* [online]. [cit. 2026-02-24]. Dostupné z: https://tls13.xargs.org/.

27. CHACON, S.; STRAUB, B. *Pro Git*. 2nd. Apress, 2014. Dostupné tiež z: https://git-scm.com/book/en/v2.

28. SWELLER, J. Cognitive load during problem solving: Effects on learning. *Cognitive science*. 1988.

29. MILLER, G. A. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological review*. 1956.

30. SRINATH, K. Python–The fastest growing programming language. *International Research Journal of Engineering and Technology*. 2017.

31. FANGOHR, H. A comparison of C, C++, and Python for use in teaching computational science. *International Conference on Computational Science*. 2004.

32. PRECHELT, L. An empirical comparison of C, C++, Java, Perl, Python, Rexx, and Tcl. *IEEE Computer*. 2000.

33. PYTHON CRYPTOGRAPHIC AUTHORITY (PYCA). *Cryptography Documentation*. 2026. Dostupné tiež z: https://cryptography.io/.

34. FLET FRAMEWORK. *Flet Documentation*. 2026. Dostupné tiež z: https://flet.dev/docs/.

35. SEBESTA, R. W. *Programming the World Wide Web*. 8th. Pearson, 2014.

36. SOMMERVILLE, I. *Software Engineering*. 10th. Boston, MA: Pearson, 2015.

37. HUNT, A.; THOMAS, D. *The Pragmatic Programmer: From Journeyman to Master*. Boston, MA: Addison-Wesley Professional, 1999.

38. DOLEV, D.; YAO, A. On the security of public key protocols. *IEEE Transactions on Information Theory*. 1983.

39. COHN-GORDON, K.; CREMERS, C.; DOWLING, B.; GARRATT, L.; STEBILA, D. *A Formal Security Analysis of the Signal Messaging Protocol* [Cryptology ePrint Archive, Paper 2016/1013]. 2016. Dostupné tiež z: https://eprint.iacr.org/2016/1013.

40. ALWEN, J.; CORETTI, S.; DODIS, Y. *The Double Ratchet: Security Notions, Proofs, and Modularization for the Signal Protocol* [Cryptology ePrint Archive, Paper 2018/1037]. 2018. Dostupné tiež z: https://eprint.iacr.org/2018/1037.

41. KREKEL, H.; TEAM, pytest-dev. *pytest: helps you write better programs* [online]. pytest-dev team, 2026. [cit. 2026-04-13]. Dostupné z: https://docs.pytest.org/en/stable//.

Použitie nástrojov umelej inteligencie

OpenAI (2026), ChatGPT (GPT-4/4o – free verzia), vybrané časti práce, pomoc pri formulácii kratších textových úsekov.

GitHub (2026), GitHub Copilot (využívajúci rôzne LLM modely – napr. OpenAI, Anthropic a ďalšie podľa dostupnosti), programová časť práce, asistencia pri písaní a úprave zdrojového kódu.

Google (2026), Gemini 3 Pro, hlavný text práce a grafické prílohy, podpora pri tvorbe a štylizácii textu a generovanie návrhu loga aplikácie.

Google (2026), Gemini 3 Thinking, 2.2, vyhľadávanie a analýza podobných aplikácií (rešerš z webových zdrojov).

DeepL SE (2026), DeepL Translator, viaceré časti práce, preklad textov medzi jazykmi do slovenčiny.

Dodatok A: Výstup z testovania pomocou unit testov

```
============================================================================= test
    session starts
    =============================================================================
platform win32 -- Python 3.12.7, pytest-9.0.3, pluggy-1.6.0 -- G:\My Drive\FEI STU
    \Ing studium\DP\app\.venv\Scripts\python.exe
cachedir: .pytest_cache
rootdir: G:\My Drive\FEI STU\Ing studium\DP\app
configfile: pyproject.toml
plugins: anyio-4.12.1
collected 55 items

tests/test_attacker_logic.py::test_collect_attacker_secret_options_one_message
    PASSED
tests/test_attacker_logic.py::
    test_attacker_analysis_decrypts_when_message_mk_compromised PASSED
tests/test_attacker_logic.py::test_limited_skipped_keys_capacity PASSED
tests/test_attacker_logic.py::test_decrypt_ignores_empty_mk_and_invalid_ck_entries
     PASSED
tests/test_attacker_logic.py::test_pending_message_with_invalid_header_ignored
    PASSED
tests/test_attacker_logic.py::test_ck_wrong_direction_does_not_decrypt PASSED
tests/test_attacker_logic.py::
    test_collect_attacker_secret_options_scales_with_large_sessions[100] PASSED
tests/test_attacker_logic.py::
    test_collect_attacker_secret_options_scales_with_large_sessions[1000] PASSED
tests/test_attacker_logic.py::
    test_collect_attacker_secret_options_scales_with_large_sessions[1200] PASSED
tests/test_attacker_logic.py::
    test_get_attacker_analysis_decrypts_with_mk_ck_and_rk_dh_compromises[mk-
    selection_filter0-expected_ids0-MK#2] PASSED
tests/test_attacker_logic.py::
    test_get_attacker_analysis_decrypts_with_mk_ck_and_rk_dh_compromises[cks-
    selection_filter1-expected_ids1-CKs#] PASSED
tests/test_attacker_logic.py::
    test_get_attacker_analysis_decrypts_with_mk_ck_and_rk_dh_compromises[rk_dh-
    selection_filter2-expected_ids2-iterative DH compromise] PASSED
tests/test_attacker_logic.py::
    test_build_attacker_dashboard_marks_implied_mirror_chain_key_and_message_key
    PASSED
tests/test_dr_logic.py::
    test_initialize_session_from_x3dh_sets_expected_initial_fields PASSED
```

```
tests/test_dr_logic.py::test_initialize_session_allows_end_to_end_send_receive
    PASSED
tests/test_dr_logic.py::test_in_order_messages_decrypt_successfully PASSED
tests/test_dr_logic.py::test_out_of_order_messages_use_and_consume_skipped_keys
    PASSED
tests/test_dr_logic.py::
    test_receive_trace_reports_double_ratchet_and_chain_key_inputs PASSED
tests/test_dr_logic.py::test_dh_ratchet_is_triggered_when_remote_dh_changes PASSED
tests/test_dr_logic.py::test_skip_message_keys_raises_when_skip_window_exceeded
    PASSED
tests/test_dr_logic.py::
    test_try_skipped_message_keys_returns_and_deletes_stored_key PASSED
tests/test_dr_logic.py::test_ratchet_encrypt_requires_initialized_dhs_keypair
    PASSED
tests/test_pqxdh_logic.py::test_new_state_bootstraps_bob_bundle_and_events PASSED
tests/test_pqxdh_logic.py::
    test_registration_upload_and_alice_opk_replenishment_flow PASSED
tests/test_pqxdh_logic.py::
    test_full_pqxdh_flow_with_pqopk_matches_shared_secret_and_ad PASSED
tests/test_pqxdh_logic.py::
    test_full_pqxdh_flow_without_pqopk_uses_last_resort_pqspk PASSED
tests/test_pqxdh_logic.py::test_pq_signature_mismatch_blocks_bundle_verification
    PASSED
tests/test_pqxdh_logic.py::test_requesting_bob_bundle_without_server_bundle_fails
    PASSED
tests/test_spqr_logic.py::
    test_initialize_session_from_pqxdh_initializes_alice_only PASSED
tests/test_spqr_logic.py::test_ratchet_init_sets_complementary_chain_directions
    PASSED
tests/test_spqr_logic.py::test_try_skipped_message_keys_returns_and_deletes_key
    PASSED
tests/test_spqr_logic.py::test_clear_old_epochs_removes_epoch_minus_two_data
    PASSED
tests/test_spqr_logic.py::
    test_clear_old_epochs_keeps_receive_chain_available_for_late_delivery PASSED
tests/test_spqr_logic.py::test_skip_message_keys_derives_and_stores_missing_keys
    PASSED
tests/test_spqr_logic.py::test_skip_message_keys_enforces_max_skip_window PASSED
tests/test_spqr_logic.py::test_scka_ratchet_encrypt_decrypt_roundtrip PASSED
tests/test_spqr_logic.py::test_scka_out_of_order_messages_use_skipped_keys PASSED
tests/test_spqr_logic.py::test_scka_ratchet_send_key_updates_epoch_on_output_key
    PASSED
tests/test_spqr_logic.py::
    test_scka_ratchet_send_key_rejects_unexpected_output_epoch PASSED
```

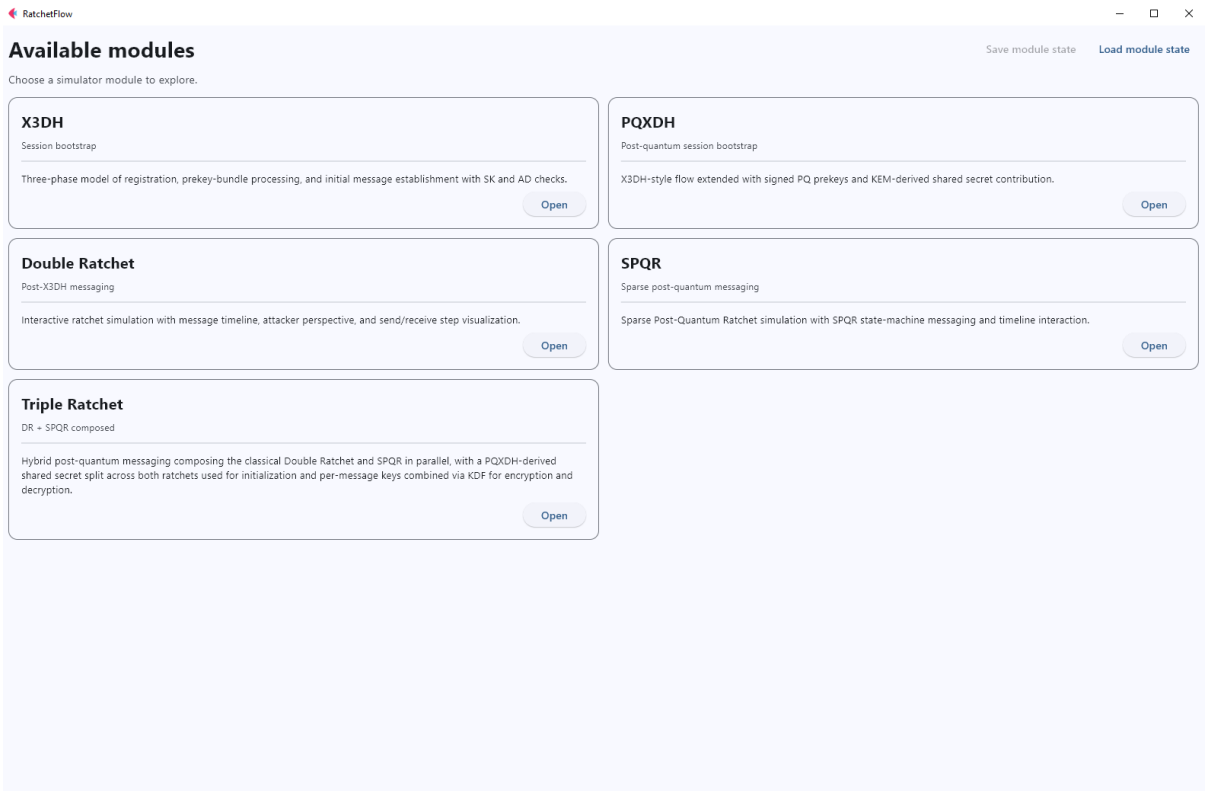
```
tests/test_spqr_logic.py::
    test_scka_ratchet_receive_key_uses_precomputed_skipped_key PASSED
tests/test_spqr_logic.py::
    test_scka_ratchet_receive_key_updates_epoch_when_output_key_present PASSED
tests/test_spqr_logic.py::test_authenticator_detects_tampered_header_mac PASSED
tests/test_spqr_logic.py::test_ratchet_send_and_receive_require_scka_state PASSED
tests/test_spqr_logic.py::
    test_late_message_after_epoch_transition_decrypts_from_history PASSED
tests/test_triple_ratchet_logic.py::test_kdf_tr_split_produces_distinct_seeds
    PASSED
tests/test_triple_ratchet_logic.py::
    test_initialize_session_from_pqxdh_sets_alice_only PASSED
tests/test_triple_ratchet_logic.py::
    test_ratchet_init_bob_triple_ratchet_sets_dh_and_root_key PASSED
tests/test_triple_ratchet_logic.py::test_triple_ratchet_encrypt_decrypt_roundtrip
    PASSED
tests/test_triple_ratchet_logic.py::test_triple_ratchet_decrypt_rejects_wrong_ad
    PASSED
tests/test_x3dh_logic.py::test_new_state_bootstraps_bob_bundle_and_events PASSED
tests/test_x3dh_logic.py::
    test_registration_upload_and_alice_opk_replenishment_flow PASSED
tests/test_x3dh_logic.py::
    test_full_x3dh_flow_with_opk_matches_shared_secret_and_ad PASSED
tests/test_x3dh_logic.py::test_full_x3dh_flow_without_opk_uses_three_dh_operations
     PASSED
tests/test_x3dh_logic.py::test_signature_mismatch_blocks_bundle_verification
    PASSED
tests/test_x3dh_logic.py::test_requesting_bob_bundle_without_server_bundle_fails
    PASSED

============================================================================== 55
    passed in 3.27s
    ==============================================================================
```

Výpis kódu 2: Výstup z testovania pomocou unit testov

Dodatok B: Používateľská príručka

Aplikácia je k dispozícii ako vopred zabalený spustiteľný súbor pre operačný systém Windows (`ratchetflow.exe`), pre Linux (`ratchetflow`) alebo online prostredníctvom webového prehliadača.[2] . Okrem toho je možné aplikáciu spustiť priamo pomocou zdrojových kódov, v tomto prípade je však potrebné nastaviť vhodné prostredie Python spolu s potrebnými závislosťami. Kompletný návod na inštaláciu je k dispozícii na GitHub-e, resp. v popise `README.md` nachádzajúcom sa v zdrojových kódoch.

Po spustení sa používateľovi zobrazí hlavná obrazovka, ktorá je znázornená na obr. 19. Na hlavnej obrazovke má používateľ možnosť vybrať si konkrétny modul a začať celú reláciu od začiatku, alebo stlačením tlačidla v pravom hornom rohu načítať už uložený stav. V druhom prípade aplikácia po načítaní automaticky otvorí daný modul v uloženom stave.

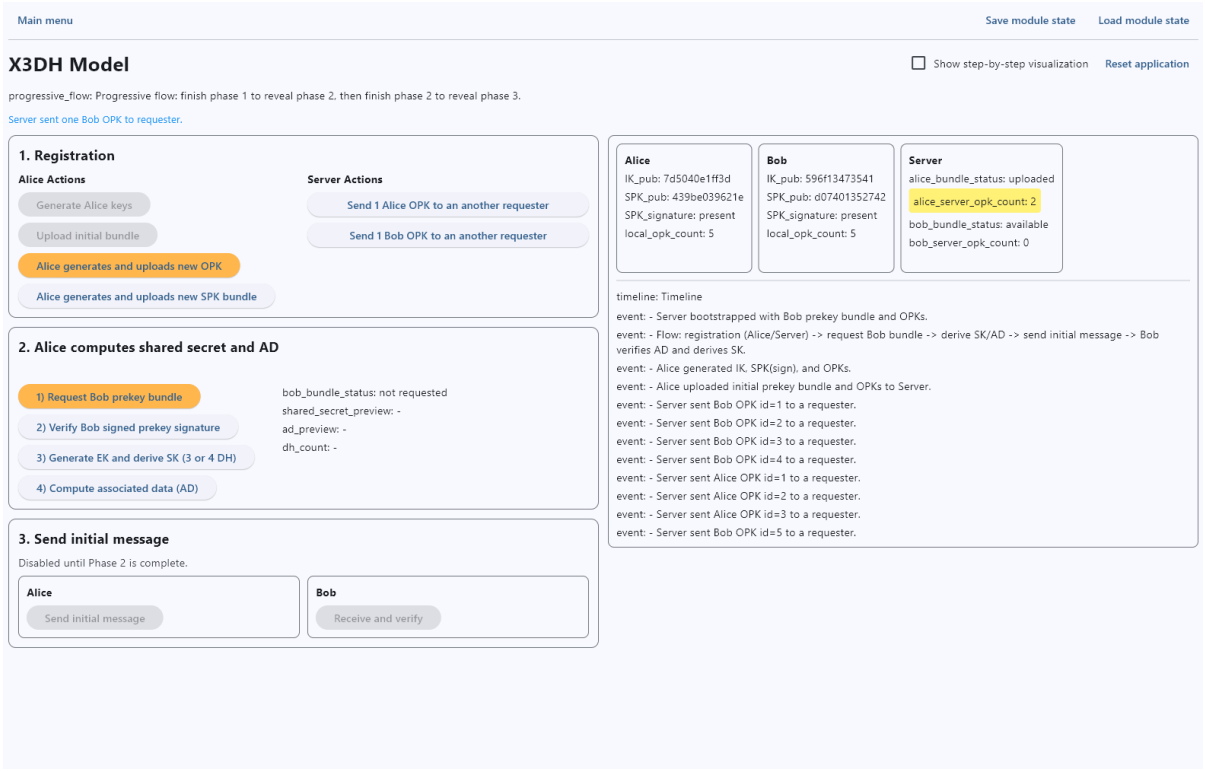


Obr. 19: Hlavná obrazovka aplikácie

[2]Dostupnosť online verzie: *https://ratchetflow.egytolnyolcig.uk*

B.1 Modul X3DH

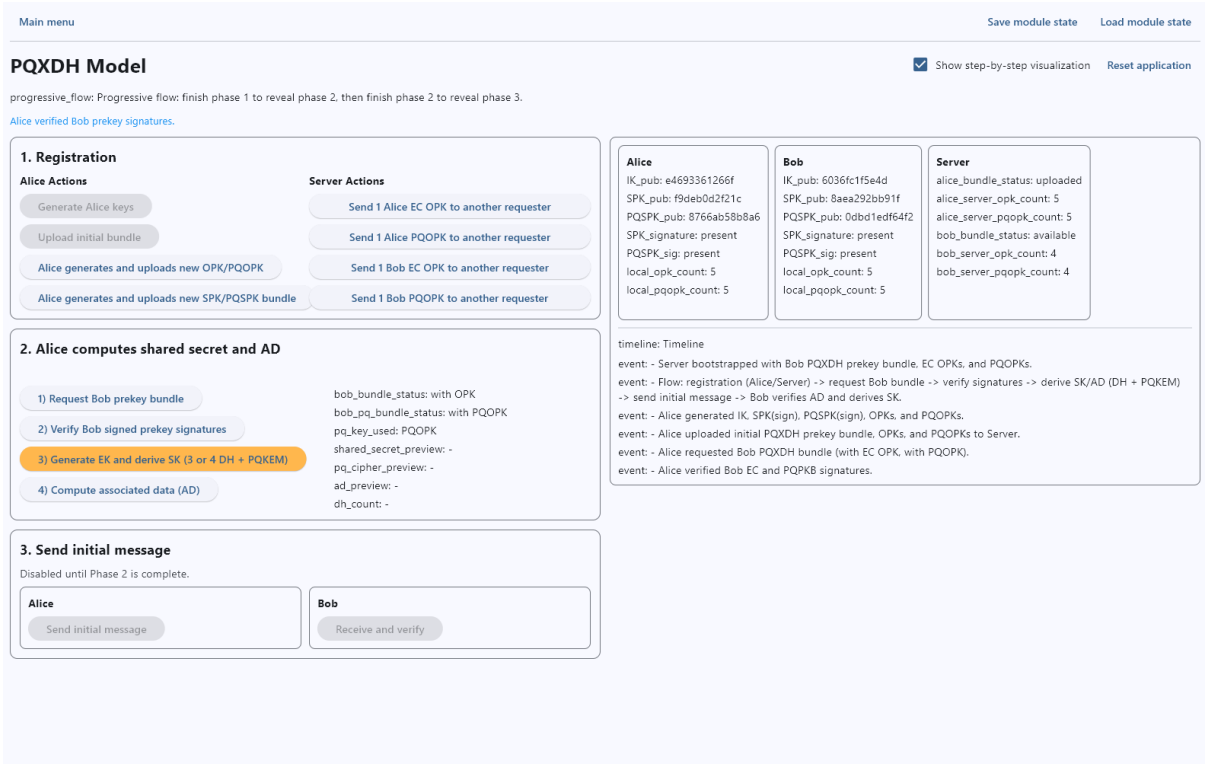
Používateľ môže sledovať priebeh protokolu z pohľadu Alice (iniciátora), s výnimkou posledného kroku, kde sú prezentované kroky Boba. Celá výmena kľúčov je rozdelená do troch fáz, pričom práve prebiehajúci krok je vizuálne zvýraznený, čím sa uľahčuje orientácia používateľa. V pravom hornom rohu sa nachádza zaškrtávacie políčko (checkbox), ktoré je predvolene zaškrtnuté, keď používateľ klikne na niektoré tlačidlo, aplikácia automaticky zobrazí vizualizáciu po jednotlivých krokoch, v ktorej je vysvetlené fungovanie daného kroku protokolu. Rozhranie modulu je zobrazené na obrázku 20.



Obr. 20: Modul X3DH

B.2 Modul PQXDH

Používateľské rozhranie modulu PQXDH je úplne zhodné s rozhraním modulu X3DH, avšak je rozšírené o postkvantový protokol ML-KEM a príslušné kľúče, ako aj o kroky KEM: enkapsulácia prebieha na strane Alice a dekapsulácia na strane Boba. Tento prvok sa takisto zahrnie do výpočtu konečného zdieľaného tajomstva. Rozhranie modulu je zobrazené na obrázku 21.



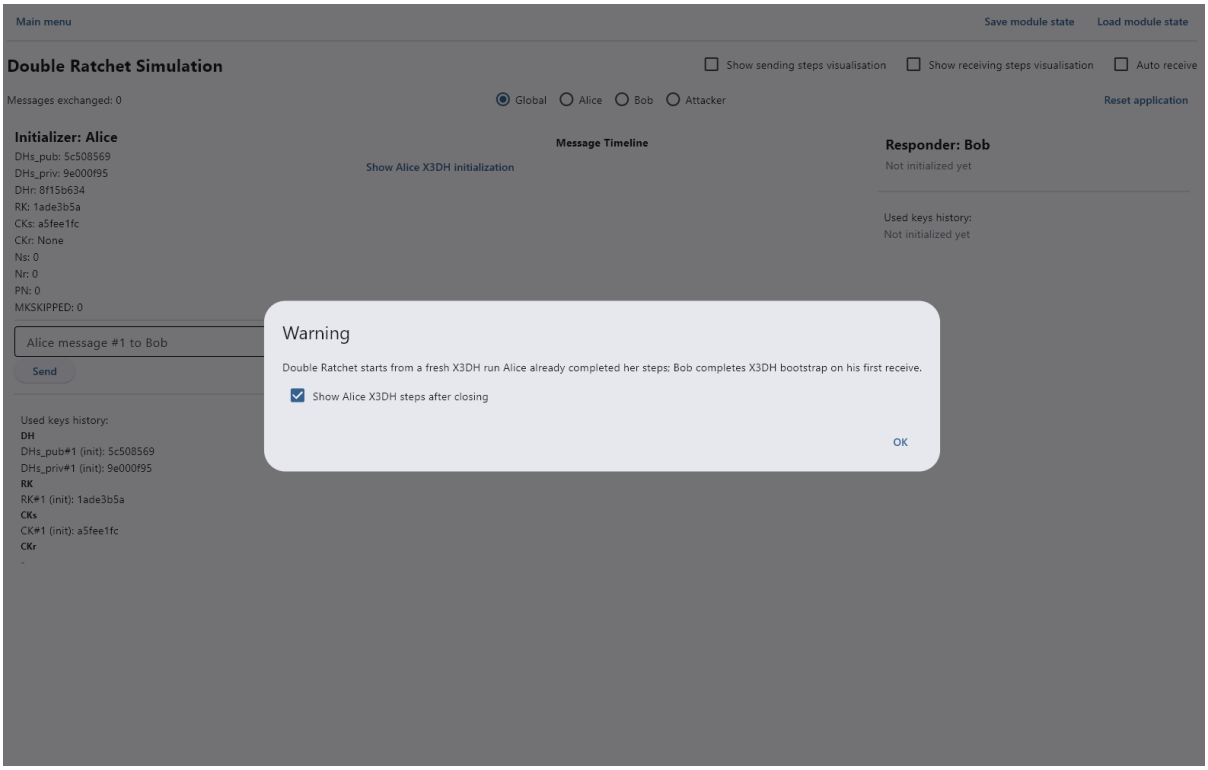
Obr. 21: Modul PQXDH

B.3 Modul Double Ratchet

o spustení modulu používateľa privíta upozornenie týkajúce sa krokov X3DH: informuje ho o tom, že účastník v úlohe iniciátora už vykonal kroky X3DH, zatiaľ čo odpovedajúca strana ich vykoná pri prijatí prvej správy. Toto upozornenie obsahuje aj zaškrtávacie políčko, ktorým používateľ určí, či si po zatvorení okna praje zobraziť vizualizáciu uvedených krokov. Upozornenie je zobrazené na obrázku 22.

V pravom hornom rohu sa nachádzajú tri zaškrtávacie políčka, pomocou ktorých si používateľ môže zvoliť, ktoré akcie a vizualizácie sa majú zobrazovať automaticky. Ak sú zaškrtnuté všetky tri, používateľ vidí kroky odosielania a následne (keďže je zapnuté aj automatické prijímanie a zobrazovanie krokov prijímania) sa ihneď zobrazia aj kroky prijímania. Ak už existujú správy čakajúce na prijatie a používateľ v tom momente zapne automatické prijímanie spolu so zobrazovaním krokov prijímania, aplikácia nezobrazí kroky prijímania pre predtým odoslané správy, ale iba pre nové. Tým sa predchádza situácii, keď by používateľ v dôsledku náhodného kliknutia musel zatvárať desiatky okien, aby mohol pokračovať v používaní aplikácie.

Pod zaškrtávacími políčkami sa v strede okna nachádzajú tlačidlá na výber perspektívy:



Obr. 22: Úvodné upozornenie modulu Double Ratchet

- **Global** – v tomto zobrazení sú viditeľné všetky údaje, všetky odoslané správy sa zobrazujú ako plaintext a používateľ môže interagovať s oboma stranami (môže odosielať a prijímať správy).
- **Alice / Bob** – v tomto zobrazení sú viditeľné len súkromné údaje daného účastníka a verejné údaje. Odosielanie je možné z oboch strán, prijímanie však len od tej strany, ktorej perspektíva je vybraná (výnimkou je zapnutý stav automatického prijímania).
- **Útočník** – túto perspektívu podrobne opisuje časť B.3.1.

Na oboch stranách okna sa nachádzajú vstupné pole a tlačidlo pod ním, prostredníctvom ktorých je možné odosielať správy. Ak používateľ nezadá vlastný text, systém automaticky odošle číslovanú správu. Ak je zapnutá vizualizácia krokov odosielania, po stlačení tlačidla *Send* sa automaticky zobrazia okná s jednotlivými krokmi. Tieto kroky je možné zobraziť aj spätne pomocou tlačidla pri príslušnej správe, toto tlačidlo je však viditeľné iba vtedy, ak danú správu prijímajúca strana už prijala.

Na prijatie správy existujú dve možnosti. Pri zapnutom automatickom prijímaní systém automaticky prijíma všetky správy v poradí ich odoslania. Druhou možnosťou je panel správ v strede okna, kde sa vedľa každej správy nachádza tlačidlo *Receive*,

pomocou ktorého môže používateľ manuálne zvoliť, ktorú správu a v akom poradí chce prijať.

Bob môže odoslať správu len vtedy, ak už prijal aspoň jednu správu od Alice.

B.3.1 Perspektíva útočníka

V tejto perspektíve sú viditeľné len verejné údaje a správy sa zobrazujú ako šifrovaný text. V spodnej časti okna sa nachádza tzv. *Attacker dashboard*, kde si používateľ môže vybrať, ktorého účastníka a ktorý kľúč sa podarilo kompromitovať, resp. ktorý kľúč správy sa podarilo prelomiť. Predvolene má každá správa žlté pozadie a je zobrazená vo forme šifrovaného textu, čo signalizuje, že danú správu nie je možné dešifrovať. Ak sa podarí kompromitovať príslušné kľúče a tým dešifrovať nejakú správu, pozadie správy sa zmení na zelené a stane sa viditeľným text správy v podobe plaintextu.

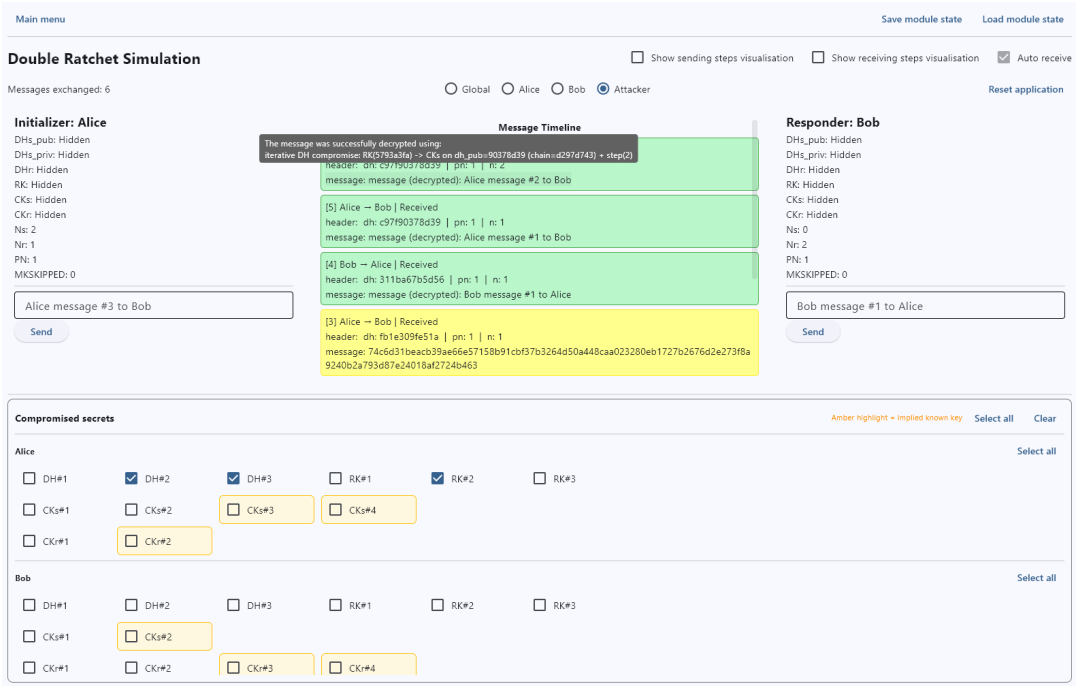
Z pohľadu útočníka je zapnuté automatické prijímanie, pretože tento pohľad slúži na ilustráciu vlastností protokolu FS a PCS , a nie na prezentáciu spracovania správ OOO. Ak útočník úspešne získa napríklad reťazový kľúč odosielateľa (CK_s), všetky správy odoslané týmto kľúčom možno dešifrovať, kľúče správ (MK) patriace k týmto správam budú tiež známe. Tieto implikované kľúče sú na paneli zvýraznené. Ak užívateľ prejde myšou nad úspešne dešifrovanú správu, zobrazí sa, akými krokmi útočník získal MK patriaci k danej správe. Na obrázku 23. je zobrazený prípad, keď útočník úspešne kompromitoval počiatočný koreňový kľúč (RK) a všetky súkromné kľúče DH, čím získal a odosielateľský, aj prijímateľský reťazec, posunutím týchto reťazcov o hodnotu n uvedenú v hlavičke správy je možné vypočítať príslušný MK.

B.4 Modul SPQR

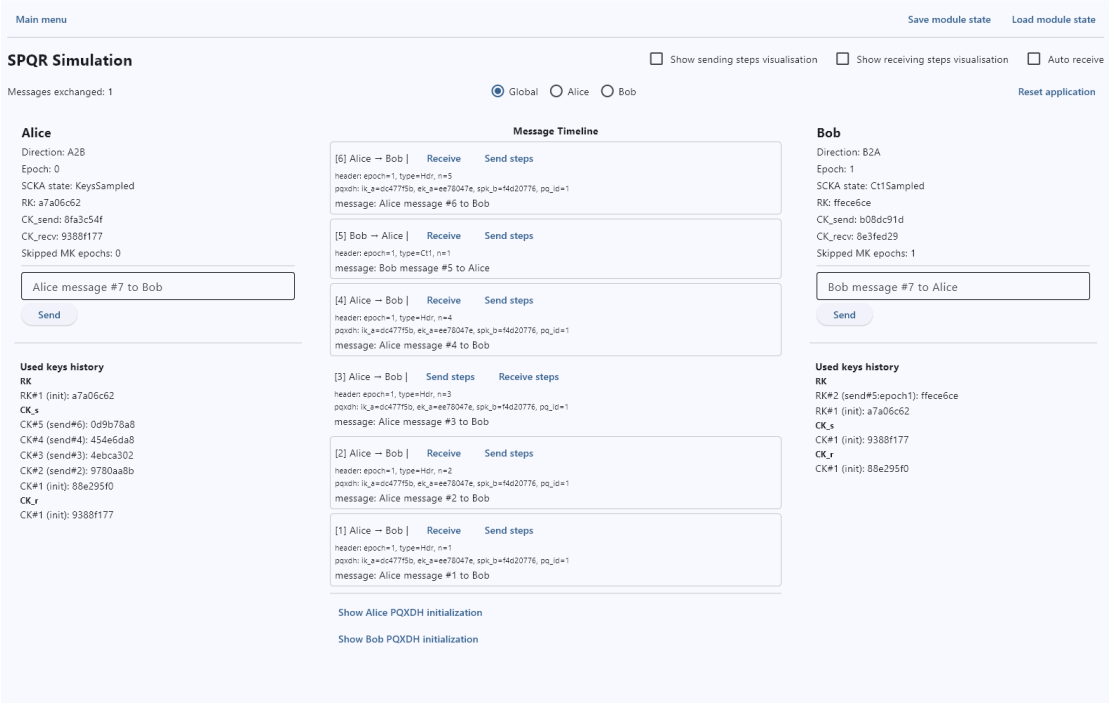
Používateľské rozhranie modulu sa vo veľkej miere podobá rozhraniu modulu Double Ratchet, s tým rozdielom, že keďže ide o postkvantový echanizmus, pri inicializácii protokolu sa používa protokol PQXDH a modul neobsahuje rozhranie útočníka. Z hľadiska a zobrazovanie krokov je používanie modulu úplne zhodné s modulom Double Ratchet. Vizualizácie jednotlivých krokov sa samozrejme líšia (keďže v pozadí prebiehajú iné procesy), ovládanie modulu je však identické. Rozhranie modulu je zobrazené na obrázku 24.

B.5 Modul TripleRatchet

Z hľadiska používateľského zážitku a samotného ovládania sa modul Triple Ratchet takmer úplne zhoduje s modulom SPQR, ako je viditeľné obrázku 25. Najvýraznejší rozdiel však spočíva v spôsobe vizualizácie jednotlivých kryptografických krokov. Keďže v

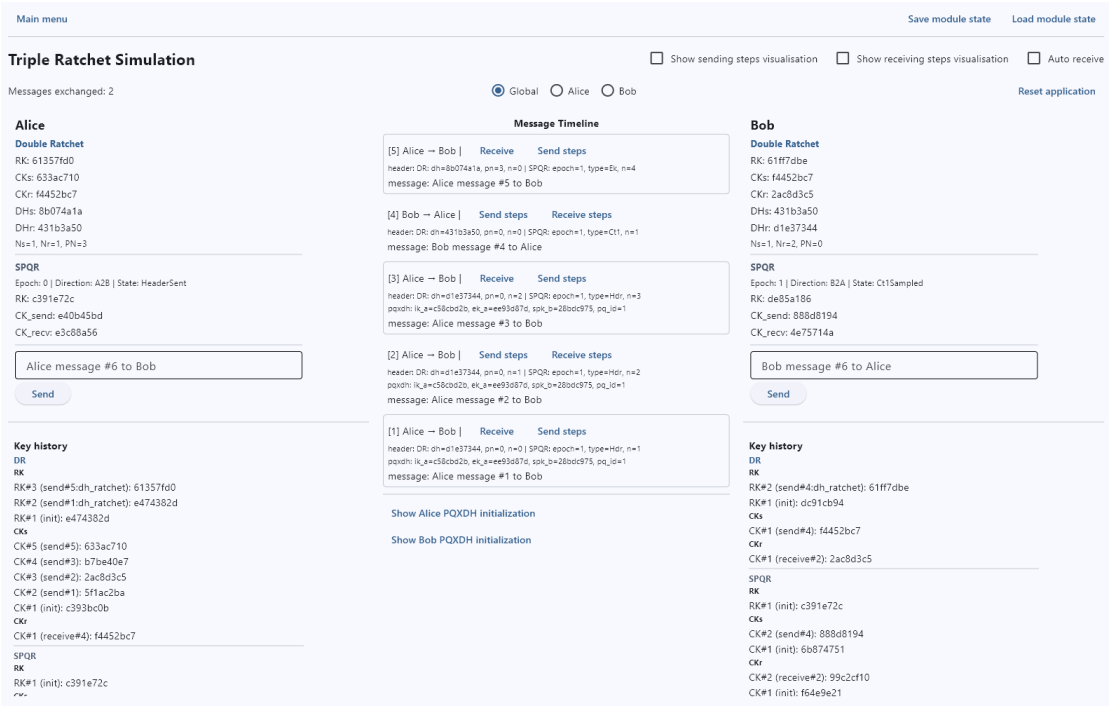


Obr. 23: Pohľad útočníka s krokmi kompromitácie kľúčov



Obr. 24: Modul SPQR

pozadí bežia paralelne dva zložité protokoly, štandardné zobrazenie všetkých detailov by bolo neprehľadné. Preto sa používateľovi pri interakcii primárne zobrazí len abstraktný prehľad (*high-level overview*) operácií zastrešujúcich vrstvy Double Ratchet aj SPQR.



Obr. 25: Modul Triple Ratchet

Ak má používateľ záujem o hlbšiu analýzu, má možnosť prostredníctvom tlačidla umiestneného priamo vo vizualizačnom okne (znázornené na obr. 26) rozbaliť detailný pohľad na procesy prebiehajúce v oboch protokoloch. Tieto detailné vizualizácie sú priamo importované z pôvodných samostatných modulov (Double Ratchet a SPQR), čo zaručuje ich úplnú terminologickú aj vizuálnu konzistenciu s predchádzajúcimi časťami aplikácie. V tomto zobrazení sú však odfiltrované redundantné informácie a vizualizácia sa zameriava výlučne na skutočné výpočtové kroky, posuny reťazcov a derivácie kľúčov v rámci oboch podkladových mechanizmov.

Obr. 26: Prehľad krokov v module Triple Ratchet s možnosťou zobrazenia detailov

Dodatok C: Zoznam elektronických príloh

C.1 Zdroje a elektronická príloha

- **Repozitár:** `https://github.com/BenceBoth115378/RatchetFlow` (zdrojové kódy)
- **Webová aplikácia:** `https://ratchetflow.egytolnyolcig.uk` (online demo)
- **Súbory v AIS:** `source.zip` (zdrojové kódy), `linux.zip` a `windows.zip` (distribúcie)

C.2 Štruktúra spustiteľných balíkov

Spustiteľné balíky pre Linux aj Windows zdieľajú analogickú štruktúru:

```
[linux/windows]/
|-- RatchetFlow (.exe)      # Hlavný spustiteľný súbor
|-- data/                   # Aplikačné dáta, assety a konfigurácia
|-- site-packages/          # Externé knižnice a závislosti Pythonu
|-- lib/ (alebo DLLs/Lib)   # Systémové knižnice a prostredie Python 3.12
\-- [flutter_windows.dll]   # Jadro enginu (len verzia pre Windows)
```