Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky

Ľahká kryptografia vo VPN sieťach

Diplomová práca

2023 **Bc. Marek Rohač**

Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky

Ľahká kryptografia vo VPN sieťach

Diplomová práca

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

Košice 2023 **Bc. Marek Rohač**

Abstrakt v SJ

slovak

Kľúčové slová v SJ

VPN

Abstrakt v AJ

english

Kľúčové slová v AJ

VPN

Bibliografická citácia

ROHAČ, Bc. Marek. *Ľahká kryptografia vo VPN sieťach*. Košice: Technická univerzita v Košiciach, Fakulta elektrotechniky a informatiky, 2023. 48s. Vedúci práce: prof. Ing. Miloš Drutarovský, CSc.

Tu vložte zadávací list pomocou príkazu
`\thesisspec{cesta/k/suboru/so/zadavacim.listom}`
v preambule dokumentu.
Kópiu zadávacieho listu skenujte čiernobielo (v odtieňoch sivej) na 200 až 300 DPI! Nezabudnite do jednej práce vložiť originál zadávacieho listu!

Čestné vyhlásenie

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

Košice, 28. 5. 2023 .....

Vlastnoručný podpis

Poďakovanie

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

Obsah

Zoznam obrázkov

Zoznam zdrojových kódov

Zoznam skratiek

AES Advanced Encryption Standard.

FSKD Full-State Keyed Duplex.

FTP File Transfer Protocol.

FW FireWall.

GW GateWay.

IP Internet Protocol.

LTS Long Term Support.

OS Operating System.

PDV Packet Delay Variation.

SSL Secure Socket Layers.

TCP Transmission Control Protocol.

TLS Transport Layer Security.

VB Virtual Box.

VM Virtual Machine.

VPN Virtual Private Network.

Úvod

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisissem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi necante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultriciesvel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero utmetus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit ametante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.VPN [**book**]

1 Virtual Private Network – VPN

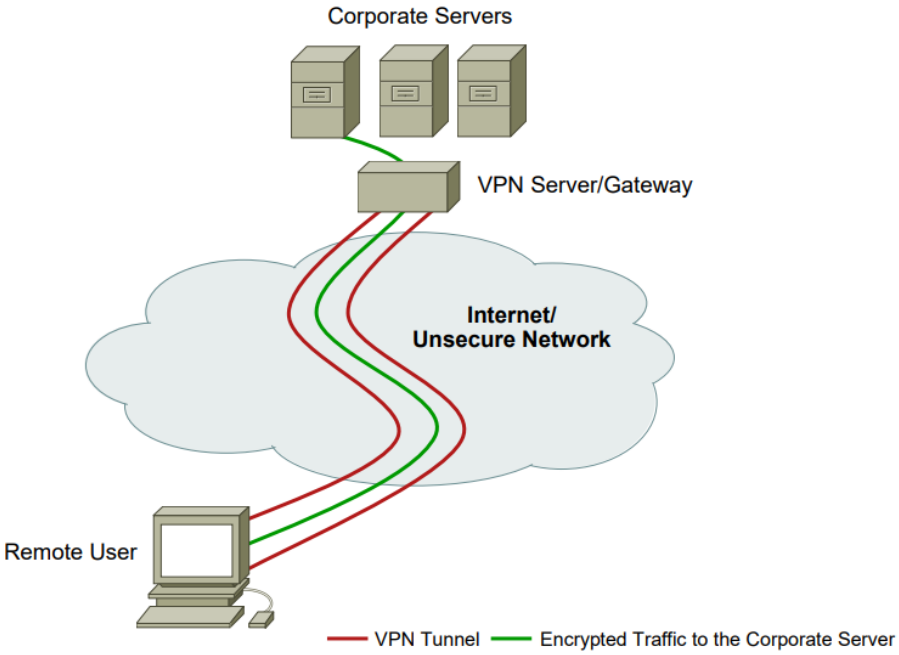
Virtuálna privátna sieť (ďalej **VPN**) je jeden zo spôsobov prepojenia zariadení, tak že internetová komunikácia medzi nimi je privátna, resp. zabezpečená aj v prípade používania nezabezpečenej, verejnej siete. Bezpečnosť spojenia je docielená pomocou kryptografických protokolov v tuneli, ktorý VPN vytvára. Pod pojmom tunel sa v skutočnosti myslí virtuálna zašifrovaná linka, ktorou je dátový paket prenášaný po sieti medzi koncovými zariadeniami. V skutočnosti tunel vzniká pomocou procesu zapuzdrenia dát. V závislosti od toho na akej úrovni OSI referenčného modelu sa pohybujeme. Táto technológia patrí aktuálne k najpoužívanejším spôsobom pripojenia sa medzi 2 rôznymi internetovými doménami. Najčastejší výskyt je možné sledovať v korporačnom prostredí, pričom cieľom je rozšírenie možností bezpečného pripojenia sa k firemnej sieti. Vzhľadom na firemné tajomstvá je nutné aby bolo takéto spojenie bezpečné a zamestnanci sa mohli pripojiť z rôznych miest. Vďaka uvedeným vlastnostiam je následne možná aj práca z domu (z ang. *Home office*), ktorá môže byť benefitom pre obe strany. Ukážka použitia VPN je znázornená pomocou 1.1.

1.1 Výhody a nevýhody VPN sieti

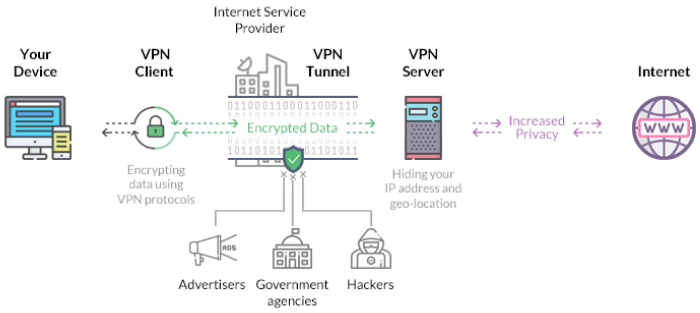
neviem ci to bude treba na samostatnu podkapitolu mozno len doplnit informacie do textu

1.2 Klasifikácia VPN sietí

V súčasnosti má čitateľ k dispozícii veľa rôznych internetových zdrojov o problematike VPN. Uvedené sú rôzne možnosti klasifikácie VPN siete. V rámci tejto práce klasifikujeme VPN siete podľa logickej topológie, použitých protokolov a vrstiev, na ktorých sú postupy aplikované. Obsahom tejto podkapitoly je rozdelenie a opis jednotlivých typov VPN sieti.



Obr. 1.1: Ukážka typického VPN pripojenia



Obr. 1.2: vpn fancy obrazok

1.2.1 Rozdelenie VPN sieti podľa logickej topológie

Podľa topológie, v ktorej VPN spojenie prebieha rozdeľujeme VPN do 3 kategórií:

- **VPN rovný s rovným** – *z ang. Peer to Peer VPN*,
- **VPN klient a server** – *z ang. Client to Server VPN*,
- **VPN sieť so sieťou** – *z ang. Site to Site VPN*.

VPN rovný s rovným

Uvedený spôsob vytvára zabezpečený tunel medzi dvoma rovnocennými uzlami, resp. zariadeniami[1], ktorý spoločne komunikujú cez verejnú sieť. Medzi zariadeniami je vytvorený tunel. Každý koniec má priradenú svoju IP adresu. Z uvedeného modelu vyplýva aj následná limitácia. VPN tunel vznikne iba medzi dvoma komunikujúcimi zariadeniami. Z toho dôvodu nie je toto použitie časté. Na obrázku 1.3 je znázornený uvedený typ VPN spojenia.

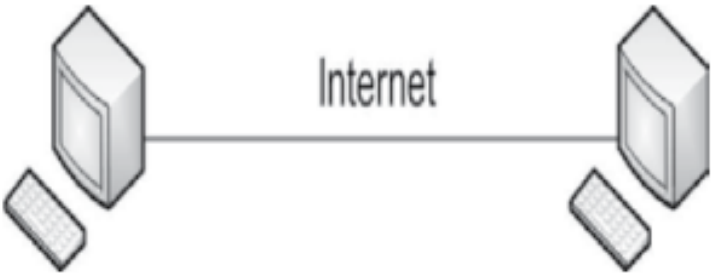


Figure 1: Peer to Peer VPN

Obr. 1.3

VPN Klient a Server

Tento typ spojenia pozostáva z pripojenia medzi nerovnocennými dvoma alebo viacerými zariadeniami. Najjednoduchší model musí pozostávať z jedného VPN servera a VPN klienta. Princíp spočíva vo vytvorení zabezpečeného tunela, ktorý je použitý na prenos dát medzi uvedenými zariadeniami. Zároveň je však možné vytvoriť $1 : N$ takýchto spojení. N reprezentuje počet VPN klientov, resp. prepojení, ktorý VPN Server dokáže nadviazať. Tento parameter je závislý najmä od hardvérových prostriedkov daného servera.

Úloha Klienta spočíva v presmerovaní všetkej svojej sieťovej komunikácie cez zabezpečený tunel, ktorý vznikol medzi ním a Serverom. Tento úkon je najčastejšie realizovaný presmerovaním trafiky cez sieťovú bránu[2] (ďalej GW). V danom OS, na ktorom VPN klient beží, je teda potrebné zmeniť IP adresu GW na adresu VPN servera. Vďaka tomu nastane presmerovania komunikácie. Tento úkon je väčšinou realizovaný programovo pomocou aplikácií. Typicky nastolia spojenie medzi Klientom a Serverom. Následne upravia sieťové nastavenia systému. Spomenuté úkony sú vysoko závislé od OS a daného programovacieho jazyka, pro-

[1] z ang. *peers*
[2] z ang. *GateWay*

stredníctvom ktorého sú úpravy realizované. Úloha Servera na druhej strane spočíva vo vytvorení možnosti pripojenia pre jedného alebo viacerých klientov. Následne Server zastupuje klientovu prítomnosť v danej sieti. Teda spracúva požiadavky Klienta a komunikuje s ostatnými zariadeniami. Komunikácia ďalej však už nie je zabezpečená pomocou šifrovania alebo tunelu. Tento fakt je znázornený na obrázku 1.4.

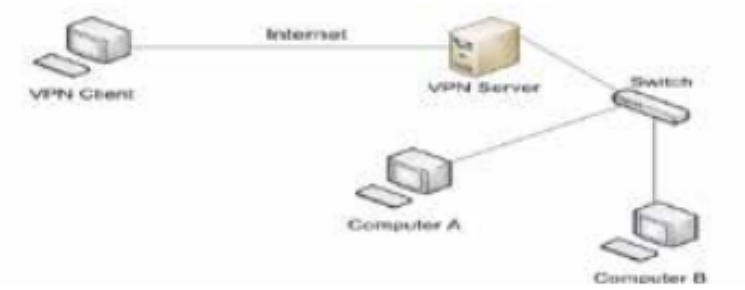


Figure 2: Client to server VPN

Obr. 1.4: potrebne prerobenie nech sa to nepodoba na 1.1 alebo vyhodit/upravit 1.1

V súčasnosti je tento spôsob považovaný za najviac používaný v oblasti korporátneho sveta. VPN server slúži ako vstupná brána do internej siete. Vďaka tomu je možné sprístupniť zdroje pre používateľov z rôznych oblastí sveta. Používateľ sa taktiež môže stretnúť s pomenovaním model **uzol k sieti**, resp. z ang *point-to-site*. Obidva pojmy sú správne a predstavujú rovnakú myšlienku zapojenia VPN.

Sieť so Sieťou VPN

VPN model Sieť so Sieťou vytvára zabezpečený tunel medzi dvoma rôznymi sieťami naprieč verejnou sieťou. Model pozostáva z 2 zariadení – VPN servera a VPN Koncentrátora[3].

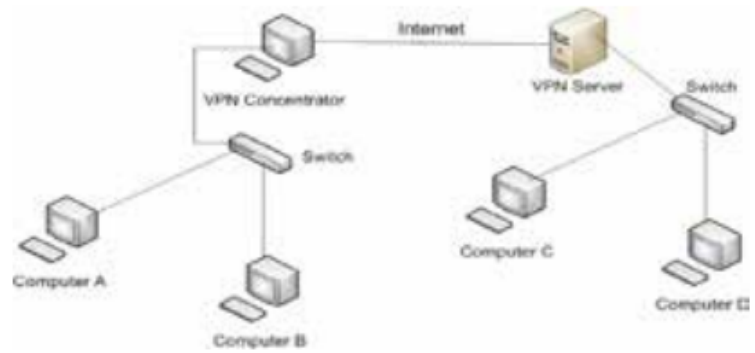
VPN Koncentrátor je typ sieťového zariadenia, ktoré poskytuje zabezpečené VPN spojenie a doručenie dát. Zvyčajne je to špecializovaný smerovač[4]. Dokáže vytvárať veľké množstvo VPN tunelov. Používa sa na nastolenie VPN modelu sieť so sieťou. Funkcionalita koncentrátora pozostáva z:

- nastolenie a konfiguráciu VPN tunela – z ang. *Establish and Configure tunnels*,
- autentizáciu používateľa– z ang. *Authenticate users*,

[3] z ang. *VPN Concentrator*
[4] z ang. *router*

- priradenie IP adries používateľov k tunelom – z ang. *Assign tunnel/IP addresses to users*,
- šifrovanie a dešifrovanie dát – z ang. *Encrypt and Decrypt data*,
- zabezpečiť integritu doručenia – z ang. *Ensure end-to-end delivery of data*.

Model Sieť so sieťou je používaný najmä pri spojení vedľajšej pobočky s hlavnou, ktoré sa nachádzajú na rozdielnych geografických lokalitách. Pomocou schémy 1.5 je znázronený tento model.



Obr. 1.5: rearanzovat

1.2.2 Rozdelenie VPN sieti podľa použitých protokolov

1.2.3 Rozdelenie VPN sieti podľa vrstiev TCP/IP referenčného modelu

Charakteristika referenčných modelov

V rámci podkapitoly 1.2.3 je potrebné pred samotnou klasifikáciou vysvetliť pojmy Referenčný model prepojenia otvorených systémov[5] (ďalej OSI) a Model opisujúci balíky internetových protokolov, známy ako, TCP/IP referenčný model.

Úlohou uvedených referenčných modelov je vizualizácia postupu spracovania dát od používateľa až k ich odoslaniu zo zariadenia[6]. Pojem spracovanie dát znamená opísanie toho ako dochádza v jednotlivých abstraktných vrstvách k pretransformovaniu používateľských dát na súbor jednotiek a núl, ktoré sú následne odoslané do iného zariadenia.

[5] z ang. *Open Systems Interconnection reference model*
[6] z ang. *end-to-end data communication*

OSI Model vznikol v skorších fázach evolúcie počítačových sietí. Vychádza skôr z teoretického než praktického prístupu. Pozostáva zo 7 abstraktných vrstiev:

- fyzická vrstva – vrstva 1 (ďalej L1),
- spojová vrstva – vrstva 2 (ďalej L2),
- sieťová vrstva – vrstva 3 (ďalej L3),
- transportná vrstva – vrstva 4 (ďalej L4),
- relačná vrstva – vrstva 5 (ďalej L5),
- prezenčná vrstva – vrstva 6 (ďalej L6),
- aplikačná vrstva – vrstva 7 (ďalej L7).

Čím je väčšie číslo vrstvy tým bližšie sa dáta nachádzajú pri používateľovi. Vďaka uvedeným vlastnostiam je tento model vhodnejší pri začiatku štúdia spracovania sieťových dát v počítači. Viac informácií o problematike nájde čitateľ v [**osi**].

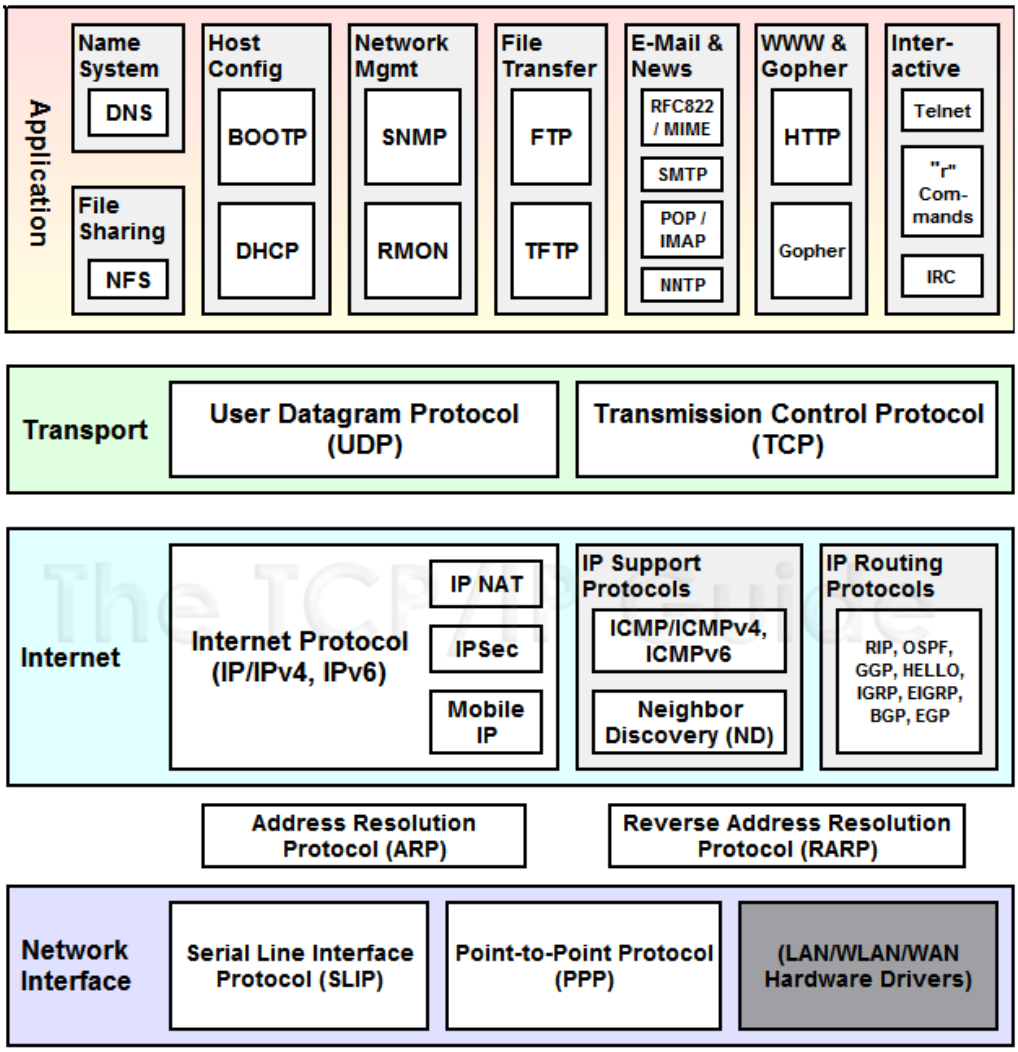
Na druhej strane TCP/IP model vznikol z praktického prístupu. Hlavný rozdiel je v počte abstraktných vrstiev, ktorý je v prípade TCP/IP zmenšený na 4 vrstvy [**tcpip**]. TCP/IP model je zobrazený pomocou schémy 1.6. V uvedenej schéme sú znázornené aj niektoré z protokolov, ktoré sa na jednotlivých vrstvách používajú.

Aplikačná vrstva (L5-7) zahŕňa protokoly používané väčšinou aplikácií na poskytovanie užívateľských služieb alebo výmenu aplikačných dát cez sieťové pripojenia, ktoré je vytvorené protokolmi na nižšej úrovni. Spája vrstvy L5 až L7 OSI modelu. Príklady známych protokolov aplikačnej vrstvy sú HyperText Transfer Protocol (HTTP), File Transfer Protocol (FTP), Simple Mail Transfer Protocol (SMTP) a iné. Údaje, resp. dáta sú pri spracovaní kódované podľa L4 protokolov. Sú zapuzdrené do protokolových jednotiek transportnej vrstvy, tzv. **segmentov**.

Transportná vrstva (L4) vytvára základné dátové kanály, ktoré aplikácie používajú na výmenu dát. Vrstva vytvára konektivitu medzi hostiteľmi, ktorá je nezávislá od siete, štruktúry užívateľských dát a smerovacích informácií. Konektivita na transportnej vrstve môže byť kategorizovaná ako orientovaná na spojenie, implementovaná v protokole TCP, alebo UDP bez orientácie na spojenie. Uvedené protokoly sú stručne charakterizované nižšie, v tejto podkapitole.

Protokoly v tejto vrstve zabezpečujú:

- riadenie chýb – z ang. *error control* [**ec**],



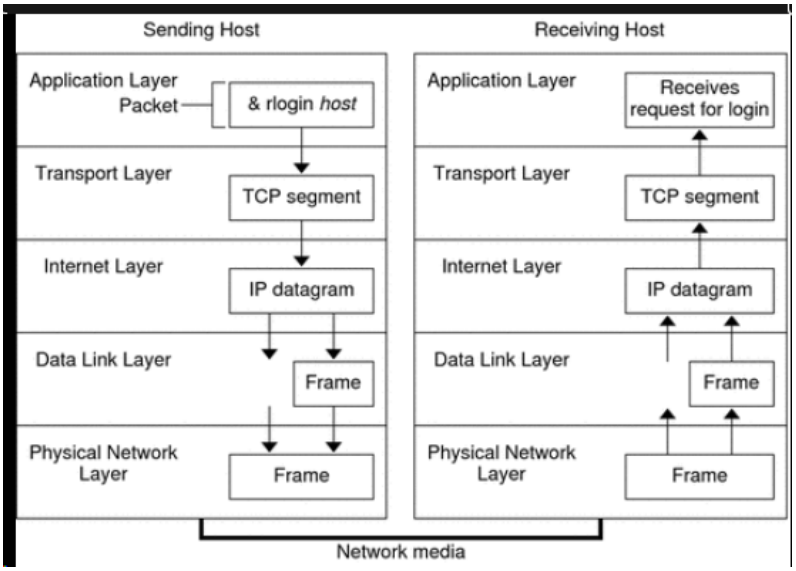
Obr. 1.6: Schéma TCP/IP modelu s niektorými protokolmi

- segmentáciu dát – z ang. *segmentation* [**sd**],
- riadenie toku dát – z ang. *flow control* [**fc**],
- riadenie preťaženia – z ang. *congestion control* [**cc**],
- adresovanie aplikácií – z ang. *application addressing* [**aa**].

Výstupom transportnej vrstvy sú segmenty, ktoré sú spracované v ďalšej vrstve referenčného modelu.

Internetová, resp. Sieťová vrstva (L3) je zodpovedná za odosielanie paketov cez jednu alebo viac sietí. S touto funkcionalitou internetová vrstva umožňuje sieťovanie, prepojenie rôznych IP sietí a v podstate vytvára internet. Z L4 segmentov tvorí pakety, tak že pridá informácie potrebné na ďalšie smerovanie.

Linková vrstva (L2) sa používa na presun paketov medzi rozhraniami internetovej vrstvy dvoch rôznych hostiteľov na rovnakom linku. Procesy vysielania a



Obr. 1.7: spoj tento obrazok s nasledujucim do jedneho

prijímania paketov na linke je možné konfigurovať. Zariadenia nazývané prepínače [7], vykonávajú rámcovania[8]. Pripravujú pakety z L3 vrstvy na prenos pridaním ďalších informácií. Týmto úkonom vzniknú rámce. Tie sa prenášajú do fyzickej vrstvy a cez prenosové médium až k hostiteľovi. Fyzická vrstva predstavuje prvok, cez ktoré sú dáta prenášane. Napríklad optický a ethernetový kábel.

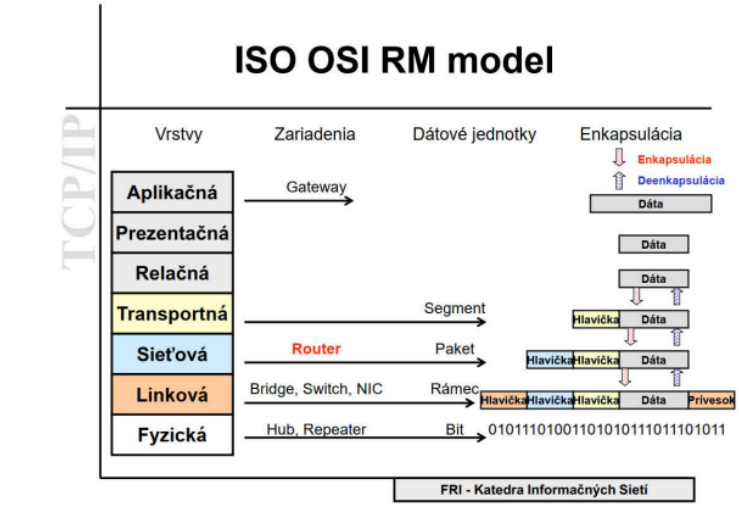
Vyššie uvedené procesy prípravy dát sú znázornené pomocou schémy 1.7 a 1.8.

Protokol riadenia prenosu (z ang. *Transmission Control Protocol*, ďalej TCP) je komunikačný protokol orientovaný na nadviazanie a udržanie sieťového spojenia medzi zariadeniami. Môže byť použitý pri úlohe príjímateľa aj odosielateľa (z ang. *full-duplex*). Úlohou je spoľahlivý prenos dát medzi komunikantmi. Odoslanie a príjem dát je v rovnakom poradí. Protokol zároveň obsahuje mechanizmy na kontrolu výskytu chýb. Svoj názov má podľa dvoch najdôležitejších protokolov:

- Protokol Riadenia Prenosu – z ang. *Transmission Control Protocol*,
- Internet Protokol – z ang. *Internet Protocol*.

Na začiatku 21. storočia je 95% paketov používaných na internete TCP. Bežné aplikácie používajúce TCP sú webové (HTTP/HTTPS protokoly), slúžiace na e-mailovú komunikáciu (SMTP/POP3/IMAP) a prenos súborov (z ang. *File Transfer Protocol – FTP*). Minimálna dĺžka hlavičky TCP je 20 bajtov a maximálna dĺžka 60 bajtov. Po pridaní údajov TCP hlavičky k prenášaným dátam, vzniká tzv. *segment*.

[7] z ang. *Switch*
[8] z ang. *frame*



Obr. 1.8: spoj tento obrazok s predchadzajucim do jedneho

V súčasnosti je možne TCP protokol implementovať softvérovo aj hardvérovo. Pri prvom z uvedených je problémom závislosť na OS a následne aj vysoká vyťaženosť procesora pri príprave a spracovaní dát. Pri hardvérovom riešení je výhodou optimalizácia a implementácia bez potreby dodatočnej úpravy OS. Hardvérové implementácie sa realizujú pomocou koprocesorov umiestnených vo vnútri procesora. Následkom toho môžeme dnes bežne pozorovať umiestnenie spomenutých zariadení na našich zariadeniach.

Podrobnejšie informácie o TCP protokole je možné nájsť na [3]. V uvedenej publikácií sa nachádza opis TCP hlavičky, metódy nadviazania a ukončenia spojenia. Obdobne je spomenuté ako dochádza k prenosu dát pomocou sekvenčných čísel. Ak má používateľ nejasnosti v fungovaní TCP protokolu, odporúča sa danú publikáciu prečítať.

Používateľský datagramový protokol (z ang. *User Datagram Protocol*, ďalej UDP) je jedným zo základných komunikačných IP protokolov. Používa sa na odosielanie správ iným hostiteľom v sieti. Správy sú prenášané ako datagramy v paketoch. UDP nevyžaduje predchádzajúcu komunikáciu na nastavenie komunikačných kanálov alebo dátových ciest. Používa jednoduchý komunikačný model bez spojenia s minimom protokolových mechanizmov. Poskytuje kontrolné súčty pre integritu údajov a čísla portov na adresovanie rôznych funkcií v zdroji a cieli datagramu.

Narozdiel od TCP) neposkytuje žiadnu záruka doručenia správy alebo duplicitnej ochrany. UDP) je vhodný na účely, kde kontrola a oprava chýb buď nie sú potrebné, alebo sa vykonávajú v aplikácii. Aplikácie citlivé na čas často používajú

UDP, pretože zahadzovanie paketov je vhodnejšie ako čakanie na pakety oneskorené v dôsledku opätovného prenosu. Príklad použitia môžu byť streamovacie služby. Podrobnejšie informácie o UDP) protokole nájde čitateľ v [**udp**].

1.2.4 Klasifikácie VPN sieti

[1], [2] https://www.vpnmentor.com/blog/different-types-of-vpns-and-when-to-use-them/ https://www.top10vpn.com/what-is-a-vpn/vpn-types/ https://www.auvik.com types/ https://www.geeksforgeeks.org/types-of-virtual-private-network-vpn-and-its-protocols/?ref=rp https://www.geeksforgeeks.org/difference-between-site-to-site-vpn-and-remote-access-vpn/?ref=rp

1.3 Kryptografické zabezpečenie protokolov

TCP protokol sam o sebe nezabezpečí dáta, ku ktorým sa pridáva TCP hlavička. Dôsledkom toho vznikli viacere protokoly slúžiace na autentizované šifrovanie dát. Najznámejší je protokol zabezpečenia prenosu – TLS a IPSec.

1.3.1 Transport Layer Security – TLS

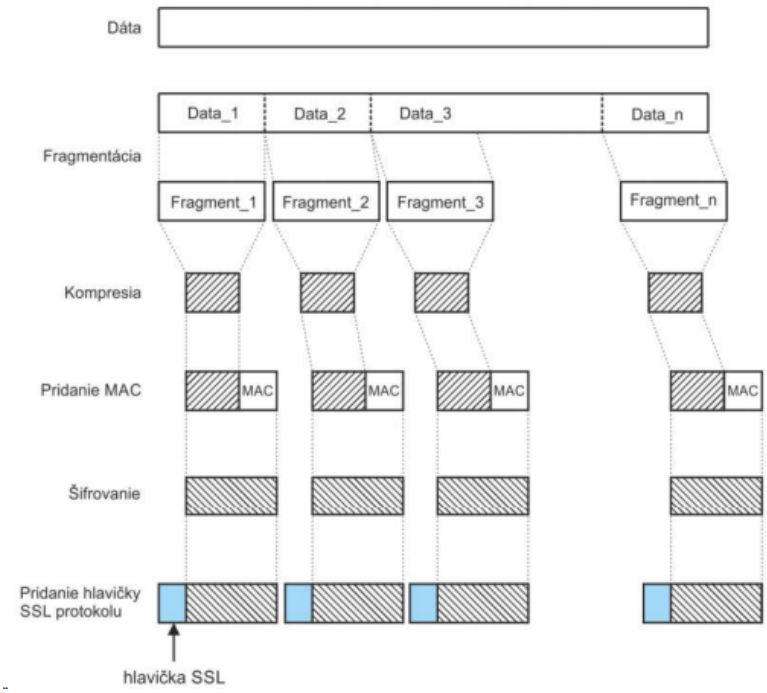
Zabezpečenie dát bolo prvotne vykonávané pomocou protokolu *Secure Sockets Layer* – SSL. Tento spôsob používa certifikáty na odšifrovanie dát. SSL malo od svojho vytvorenia dlhý vývoj, ktorý smeroval až k doteraz najpoužívanejšiemu TLS vo verzii 1.3. Inými slovami, TLS protokol je nástupca SSL pričom obsahuje rôzne úpravy a vylepšenia najmä z hľadiska rýchlosti. Zároveň sa v dnešnej dobe neodporúča používanie SSL protokolu. Dôsledkom optimalizácií je, že klienta komunikujúci so serverom cez HTTPS protokol s TLS 1.3 je rýchlejší ako v prípade použitia nešifrovaného HTTP variantu.

TLS pracuje niekde na pomedzi aplikačnej a transportnej vrstvy. Spôsob spracovania dát je zobrazený pomocou schémy 1.9 s SSL operáciami, prebratej z [4].

Viac informácií o TLS protokole, jednotlivých verziách a optimalizáciách je možné nájsť na [5].

1.3.2 Internet Protocol Security – IPSec

Ochrana medzi transportnou a sietovou vrstvou TCL/IP. Nabudúce. [4]



Obr. 1.9: Prehľad operácií v SSL protokole

2 Kryptografia vo VPN

Cieľom kryptografických blokov je utajiť správu pri jej ceste z bodu A do B. Teda od odosielateľa (tvorcu) správy, až k jej prijímateľovi. Dôsledkom tohto úkonu dochádza k zabezpečeniu 3 hlavných úloh kryptografie:

- **ochrana osobných údajov** (dôvernosť) – z ang. *Data Privacy*,
- **autenticita údajov** (prišla z miesta, kde sa uvádza) – z ang. *Data Authenticity*,
- **integrita údajov** (nebolia upravená počas prenosu) – z ang. *Data Integrity*.

Prvý z uvedených bodov je najčastejšie žiadaným a známym cieľom. Odosielateľ správy zašifruje jej obsah pomocou použitia niektorého z šifrovacích algoritmov, kryptografického kľúča a následne správu odošle. Na druhej strane, prijímateľ, musí použiť komplementárny dešifrovací algoritmus s prislúchajúcim kľúčom. Ktokoľvek kto sa dostane medzi týchto komunikantov, k takto preposielanej správe, z nej nedokáže obsahovo nič zistiť, v istom časovom období.

Dôsledkom týchto faktov je jasné, že komunikanti musia mať jasne definovaný použitý algoritmus. Zároveň je nutné aby došlo k výmene kryptografických kľúčov, ktoré sú pri šifrovaní a dešifrovaní aplikované. Tým sa zabezpečí aj druhá úloha – Autenticita dát, pretože potrebné informácie budú mať len komunikanti.

V kryptografických blokoch sú taktiež aplikované algoritmy na overenie integrity správ. Ich úlohou je potvrdiť, že do obsahu správy nebolo, počas jej transportu medzi komunikantmi, nič pridané, resp. ubrané.

V súčasnosti má používateľ možnosť vybrať si zo širokej ponuky kryptografických algoritmov. Medzi najznámejší patrí AES [6], ktorý je aktuálne používaný ako štandardný kryptografický algoritmus. Detailný opis jednotlivých blokov a postupov použitých v AES-e, je obsahom rôznych vydaní. Viac informácií o problematike nájde čitateľ v [7], ktorá obsahuje okrem opisu AES-u aj doležité informácie, týkajúce sa kryptografických základov.

V rámci tejto práce sme sa rozhodli opísať, v súčásnosti menej známe, algoritmy:

- XOODOO [8]
- Gimli [9]
- Simpira384 [10]

Uvedené permutácie sú použité ako kryptografické primitívum a pri následnej tvorbe ďalších kryptografickych blokov.

2.1 Kryptografický algoritmus XOODOO a variácie

XOODOO je sada 384-bitových kryptografických permutácií parametrizovaných počtom kôl. Funkcia kola/rundy[1] funguje na 12 slovách[2] po 32 bitoch. Vďaka tomu je efektívna aj na menej výkonných procesoroch nižšej triedy. Vytvoril ju tím Keccak, ktorý stojí za viacerými úspešnými kryptografickými algoritmami. Napríklad hashovacie funkcie z rodiny SHA-3 a iné – [11]. XOODOO algoritmus vznikol po vytvorení tzv. Kravatte autentizačno-šifrovacieho algoritmu [12], založené na Keccak-p permutácií [13]. Ten sa ukázal ako dostatočne rýchly na širokom spektre platforiem. Avšak nezapadá do kategórie tzv. ľahkej kryptografie [3].

Tím Keccak vypracoval nové riešenie. Ním bol port medzi ich prvotným Keccak-p dizajnom a Gimli-ho [14] permutačným algoritmom. Vo výsledku autori zlúčili lepšie realizované prvky z oboch algoritmov do jedného celku. Primárny problém samotnej Gimli permutácie bol v slabom prejave zmeny výstupu po malých zmenách vo vstupnej správe. Táto vlastnosť sa v kryptografii označuje pomocou anglického pojmu, tzv. *propagation properties*[4]. Novo-vzniknuté riešenie autori pomenovali XOODOO. Na základe rôznych variácií tohto kryptografického primitíva sa im následne podarilo vytvoriť sadu vysoko efektívnych kryptografických funkcií. Medzi sady, ktorých jadro tvorí XOODOO, patrí Xoodyak a Xoofff. Xoofff pozostáva zo zlúčenia Farfalle konštrukcie [15] so XOODOO permutáciou.

Xoodyak ma narozdiel od Xoofff duplexovú konštrukciu [16]. Vo výsledku máme ľahko prenosnú, všestrannú, kryptografickú knižnicu. Je vhodná do výkonovo obmedzených prostredí. Môže sa použiť pre väčšinu kryptografických funkcií, ktoré používajú symetrický kľúč. Napríklad hashovanie, šifrovanie, výpočet MAC alebo autentizované šifrovanie. O kvalite riešenia napovedá aj fakt, že

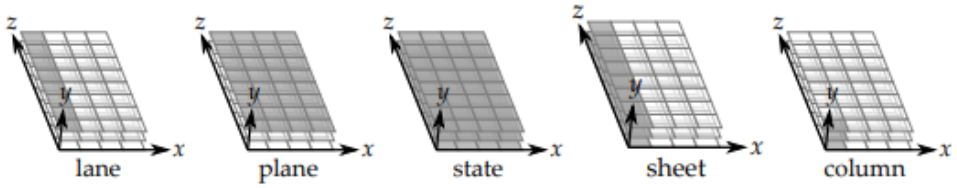
[1] z ang. *round*
[2] z ang. *words*
[3] z ang. *lightweight cryptography*
[4] Cieľom je aby aj zmena jedného bitu na vstupe, ovplyvnila čo najviac bitov vo výstupe – tzv. *Lavínový efekt*.

sada Xoodyak je jedným z 10 finalistov v oblasti ľahkej kryptografie NIST štandardizačného procesu. V rámci tejto kapitoly opíšeme kryptografické primitívum XOODOO a následne balík Xoodyak. Informácie o téme boli čerpané z týchto zdrojov: [8], [17], [18], [19], [20],[21].

2.1.1 XOODOO permutácia

XOODOO je rodina permutácií, ktorá je definovaná počtom rúnd. Má klasickú iteračnú štruktúru. Teda opakovateľne sa vola rundová funkcia s aktuálnym stavom. Pre pochopenie operácií je nutné pochopiť určité označenie použité v algoritme.

Stav – **state**, pozostáva z 3 rovnako veľkých horizontálnych rovín – **planes**. Každá z týchto rovín obsahuje štyri paralelne 32-bitové pruhy – **lanes**. Okrem tejto charakteristiky je možné opísať stav ako množinu 128 stĺpcov – **columns**, pričom jeden stĺpec obsahuje 3 bity v každej rovine. Stav je teda tvorený zo stĺpcov usporiadaných v poli o rozmere 4x32. Posledná položka na opis stavu sú tzv. listy – **sheets**. List sa skladá z 3 na sebe uložených pruhov. Uvedené pojmy sú znázornené pomocou schémy 2.1, ktorá bola prebraná z [18].



Obr. 2.1: Grafické znázornenie terminológie v XOODOO

Roviny majú index y. $y = 0$ zodpovedá spodnej rovine a vrchná rovina $y = 2$. Bit je označený s indexom z vrámci množiny pruhov. List ich označujeme pomocou indexu x. Takže pozícia pruhu v stave je definovaná pomocou dvoch súradníc (x, y). Konkrétny bit je možné v stave následne reprezentovať pomocou trojice súradníc (x, y, z). Pri učení stĺpca sú potrebné 2 súradnice (x, z). Pred spustením samotného algoritmu musí používateľ vykonať mapovanie 384-bitovej správy voči horizontálnym rovinám. Tento úkon sa realizuje pomocou vzorca 2.1.

$$i = z + 32(x + 4y) \tag{2.1}$$

Rundová funkcia pozostáva z 5 krokov:

1. miešanie vrstvy (z ang. *a mixing layer* θ),
2. posun rovín (z ang. *a plane shifting* ρ_{west}),

3. pridanie rundových konštánt (z ang. *the addition of round constants* ι),
4. nelineárna vrstva (z ang. *a non-linear layer* χ),
5. posun rovín (z ang. *an another plane shifting* ρ_{east}).

Opis jednotlivých krokov je znázornený pomocou schémy 2.2, ktorá je prebratá z [18]. Schéma opisuje jednotlivé kroky algoritmu vrátane rundových konštánt.

Table 1: Notational conventions

A_y	Plane y of state A
$A_y \lll (t, v)$	Cyclic shift of A_y moving bit in (x, z) to position $(x + t, z + v)$
$\overline{A_y}$	Bitwise complement of plane A_y
$A_y + A_{y'}$	Bitwise sum (XOR) of planes A_y and $A_{y'}$
$A_y \cdot A_{y'}$	Bitwise product (AND) of planes A_y and $A_{y'}$

Algorithm 1 Definition of Xoodoo$[n_r]$ with n_r the number of rounds

Parameters: Number of rounds n_r
for Round index i from $1 - n_r$ to 0 **do**
$A = R_i(A)$

Here R_i is specified by the following sequence of steps:

θ :
$P \leftarrow A_0 + A_1 + A_2$
$E \leftarrow P \lll (1, 5) + P \lll (1, 14)$
$A_y \leftarrow A_y + E$ for $y \in \{0, 1, 2\}$

ρ_{west} :
$A_1 \leftarrow A_1 \lll (1, 0)$
$A_2 \leftarrow A_2 \lll (0, 11)$

ι :
$A_0 \leftarrow A_0 + C_i$

χ :
$B_0 \leftarrow \overline{A_1} \cdot A_2$
$B_1 \leftarrow \overline{A_2} \cdot A_0$
$B_2 \leftarrow \overline{A_0} \cdot A_1$
$A_y \leftarrow A_y + B_y$ for $y \in \{0, 1, 2\}$

ρ_{east} :
$A_1 \leftarrow A_1 \lll (0, 1)$
$A_2 \leftarrow A_2 \lll (2, 8)$

Table 2: The round constants c_i with $-11 \leq i \leq 0$, in hexadecimal notation (the least significant bit is at $z = 0$).

i	c_i	i	c_i	i	c_i	i	c_i
−11	0x00000058	−8	0x000000D0	−5	0x00000060	−2	0x000000F0
−10	0x00000038	−7	0x00000120	−4	0x0000002C	−1	0x000001A0
−9	0x000003C0	−6	0x00000014	−3	0x00000380	0	0x00000012

Obr. 2.2

2.1.2 Xoodyak

Xoodyak možno považovať za všestranný kryptografický nástroj. Je vhodný pre väčšinu operácií využívajúcich symetrický kľúč. Napríklad generovanie pseudonáhodných bitov, autentizáciu, šifrovanie a iné. Tím Keccak použil pri návrhu duplexnú konštrukciu. Konkrétne variant s plným stavom a využitím kľúča. Tento dizajn označujeme ako Full-State Keyed Duplex (FSKD). Viac o tejto konštrukcii si čitateľ môže prečítať v [16]. Operačný režim, v ktorom Xoodyak pracuje sa nazýva Cyklista – z ang. *Cyclist*. Tento názov získal ako opozitum k pomenovaniu režimu Motorista, ktorý je možné nájsť v Keyak schéme [22]. Narozdiel od uvedeného balíka Keyak, nie je Xoodyak limitovaný len na autentizované šifrovanie. Je jednoduchší hlavne kvôli tomu že neobsahuje paralelné varianty.

Režim Cyklista

Režim Cyklista funguje na princípe kryptografických permutácií f, teda zmeny usporiadania bitov za pomoci tajného kľúča a matematických operácií. Parametrami sú veľkosti blokov R_{hash}, R_{kin}, R_{kout} a veľkosť račety, resp. západky[5] [23] $\ell_{ratchet}$. Uvedený pojem sa v kryptografii používa vo forme obrazného pomenovania. Cieľom je poukázať na jednoduchý pohyb vpred, ale s ťažkým, resp. zložitejším pohybom naspäť. Dôležité je, že uvedený scenár je vyvolaný zámerným dizajnom. Šírka permutácie b' je definovaná pomocou vzorca 2.2. Všetky uvedené parametre sú v bajtoch. Pre označenie prázdneho slova budeme používať ϵ.

$$max(R_{hash}, R_{kin}, R_{kout}) + 2 \leq b' \tag{2.2}$$

Cyklista operuje v dvoch režimoch – **hašovací a kľúčový**[6]. Inicializácia prebieha pomocou príkazu `CYCLIST(K,id,counter)`. Ak sa parameter K rovná prázdnemu slovu ϵ, tak potom nastane spustenie v hašovacom režime. Aktuálne nie je do implementácie zakomponovaná možnosť zmeny režimu po inicializácií. Vývojári však túto vlastnosť nevylúčili pre prípadné aktualizácie balíka.

Dostupné funkcie závisia od režimu, v ktorom sa Cyklista spúšťa. Medzi ne patria `ABSORB()` a `SQUEEZE()`. Možno ich volať v oboch režimoch, zatiaľ čo funkcie `ENCRYPT()`, `DECRYPT()`, `SQUEEZEKEY()` a `RATCHET()` sú dostupné len pre kľúčový režim. Účel každej funkcie je nasledujúci:

- `ABSORB(X)` absorbuje vstupný reťazec X,
- C ← `ENCRYPT(P)` zašifruje P do C a absorbuje P,

[5] z ang. *the ratchet size*
[6] z ang. *hash and keyed mode*.

- P ← `DECRYPT(C)` dešifruje C do P a absorbuje P,
- Y ← `SQUEEZE($\ell$)` vytvára ℓ-bajtový výstup, ktorý závisí od doteraz absorbovaných dát,
- Y ← `SQUEEZEKEY($\ell$)` funguje ako `SQUEEZE($\ell$)`, ale používa sa za účelom generovania odvodeného kľúča,
  ell nefunguje v listings pckg
- `RATCHET()` transformuje stav na nevratný tak, aby sa zabezpečila dopredná bezpečnosť[7] [24].

Stav bude závisieť od postupnosti volaní funkcií a od jeho vstupných reťazcov. Presnejšie povedané, zámerom je, že akýkoľvek výstup závisí od postupnosti všetkých vstupných reťazcov a volaní, tak že akékoľvek dva nasledujúce výstupné reťazce budú výstupom rôznych domén. Napríklad volanie `ABSORB(X)` znamená, že výstup bude závisieť od reťazca X. Na druhej strane `ABSORB()` vo funkcii `ENCRYPT(P)` vytvorí výstup závislý aj od P z funkcie šifrovania. Okrem uvedených závislostí ovplyvňujú výstup aj iné dizajnové riešenia. Príkladom je minimalizácia pamäťovej stopy. Vo výsledku teda výstup závisí od počtu predchádzajúcich volaní funkcie `SQUEEZE()` a predtým spracovaných textov pomocou funkcií `ENCRYPT()` a `DECRYPT()`. Viac informácií o režime je dostupných v kapitole 7.2, publikácie [18].

Definícia a bezpečnosť

Xoodyak je definovaný pomocou operatívneho režimu Cyklista nasledovne:

$$CYCLIST[f, R_{hash}, R_{kin}, R_{kout}, \ell_{ratchet}] \tag{2.3}$$

Kde jednotlivé parametre majú veľkosti:

1. f – permutácia XOODOO so šírkou 48 bajtov $(384$ bitov$)$,
2. R_{hash} – 16 bajtov,
3. R_{kin} – 44 bajtov,
4. R_{kout} – 24 bajtov,
5. $\ell_{ratchet}$ – 16 bajtov.

[7] z ang. *Forward secrecy*

Takto definované parametre algoritmu dokážu poskytnúť 128-bitovú bezpečnosť v oboch režimoch Cyklistu. Samozrejmosťou je, že v prípade kľúčového režimu, musí byť veľkosť kľúča rovná alebo väčšia ako 128 bitov. Viac informácií o kryptografickej bezpečnosti algoritmov je možné nájsť v [25].

Viac informácií o bezpečnosti Xoodyak-a je možné nájsť v [**7.3**, 18], odkiaľ boli informácie čerpané.

2.1.3 Možnosti použitia Xoodyak algoritmu

Obsahom tejto podkapitoly sú uvedené postupy ako a za akých okolností je daný balík možné použiť.

Použitie hašovacieho režimu

Xoodyak sa dá aplikovať ako hašovacia funkcia.

Použitie kľúčového režimu

2.2 Kryptografický algoritmus Gimli

2.3 Kryptografický algoritmus Simpira384

3 Prostredie virtuálnych strojov pomocou VirtualBox

Pri testovaní funkcionality VPN sme použili virtualizačný nástroj (ďalej **VM**) Virtual Box (ďalej **VB**), spoločnosti Oracle, vo verzii 6.1.30. VB je voľne dostupný. Inštalácia je jednoduchá a rýchla. Viac informácii o nástroji je možné dohľadať v [26].

Pre použitie je potrebné aby mal používateľ k dispozícií obraz operačného systému (ďalej OS). Tie nie je problém získať ani pre OS Windows a podobne, avšak pri našej práci sme zvolili využitie voľno dostupného OS – **Linux Ubuntu** vo verziách 20.04.3(LTS[1])– OSC a 21.10 – OSS. Pri opise práce použijeme označenia OSS pre VPN server a OSC pre klienta.

Pri jednoduchej inštalácií VM sme použili konfiguráciu s 2048 MB RAM a 2 jadrami. (Minimálna inštalácia). V prípade potreby dávame do popredia návod na prípravu Windows OS v VM – [27], avšak postup je triviálny. Po inštalácii sme OS aktualizovali pomocou príkazov:

```
sudo apt-get update
sudo apt-get upgrade
```

Následne sme doinštalovali potrebné súčasti k VM OS vo verzii ako je VB, teda 6.1.30. Dôvodom bolo zväčšenie rozlíšenia a využívanie možnosti zdieľaného priečinka s OS, na ktorom daný VB beží. Za účelom správneho fungovania priečinka bolo nutné v termináli použiť príkaz:

```
sudo usermod -aG vboxsf $(whoami)
```

a následne reštartovať OS.

Posledné úpravy prostredia sú spojené s jazykom C a balíčkom Make. Inštalácia je opäť jednoduchá. Použili sme príkazy:

```
sudo apt install gcc
sudo apt install make
```

[1] z ang. Long Term Support

Mode	VM→Host	VM←Host	VM1↔VM2	VM→Net/LAN	VM←Net/LAN
Host-only	+	+	+	–	–
Internal	–	–	+	–	–
Bridged	+	+	+	+	+
NAT	+	Port forward	–	+	Port forward
NATservice	+	Port forward	+	+	Port forward

Obr. 3.1: Konektivita jednotlivých sieťových adaptérov

Uvedené úpravy boli vykonané na oboch OS.

V prípade použitia OS Windows je postup inštalácie jazyka C a balička Make zložitejší. Používateľovi odporúčam použitie knižníc Winlibs, dostupne na webe. Po stiahnutí balíčkov musí používateľ importovať uvedený balíček, resp. cestu k nemu, do premenných prostredia OS Windows. Jeden zo spôsobov je uvedený aj na Winlibs stránke.

3.1 Zmena sieťových adaptérov

VB ponúka rôzne možnosti nastavenia sieťových adaptérov. Aktuálne su dostupné tieto:

- Not attached
- Network Address Translation (ďalej NAT)
- NAT Network
- Bridge adapter
- Internal
- Host-Only
- Generic driver
- Cloud-Based – experimentálne

Konektivita jednotlivých možností je znázornená pomocou obrázku 3.1. Viac informácií o jednotlivých režimoch je dostupných na [28].

Vzhľadom k našim potrebám, teda obojsmerná komunikácia medzi 2 VM, sú pre nás relevantné režimy bridge a NAT network. Ako si môžeme všimnúť, NAT vyžaduje dodatočnú konfiguráciu portov v prípadoch kedy chceme aby nastala komunikácia medzi dvoma VM. Z tohto dôvodu je pre čo najjednoduchší prístup

zvoliť práve režim bridge. Ten priradí VM vlastnú IP adresu, pomocou, ktorej stroj komunikuje.

Viac o jednotlivých režimov je taktiež možné nájsť v [29]. Autor sa venuje postupu konfigurácie jednotlivých režimov spoločne s ich opisom.

4 DSVPN – Dead Simple VPN

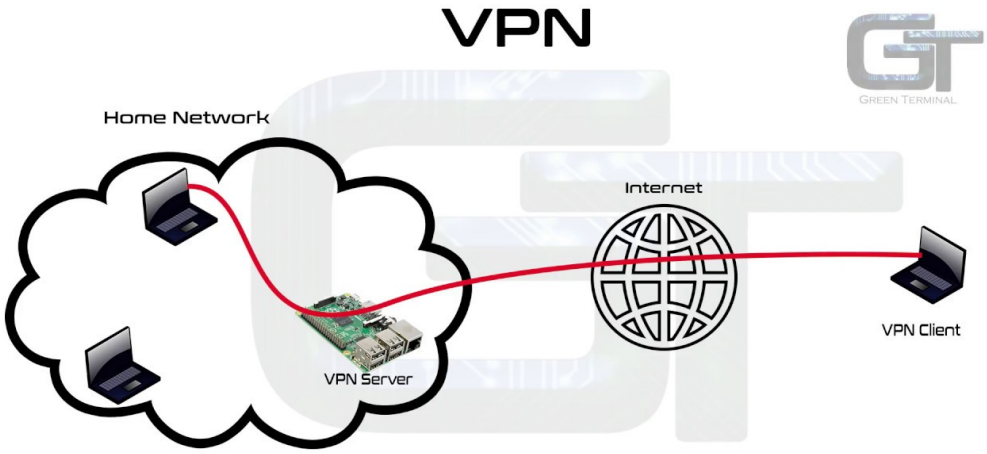
Dead Simple VPN je voľne dostupný[1] program, napísaný v jazyku C. Určený je pre operačný systém Linux. Autorom je Frank Denis. DSVPN rieši najbežnejší prípad použitia VPN, teda pripojenie klienta k VPN serveru cez nezabezpečenú sieť. Následne sa klient dostane na internet prostredníctvom servera. Uvedenú skutočnosť je možne vidieť na schéme 4.1.

DSVPN používa protokol riadenia prenosu – TCP[2] [30]. Medzi ďalšie pozitíva patrí:

- Používa iba modernú kryptografiu s formálne overenými implementáciami,
- Malá a konštantná pamäťová stopa. Nevykonáva žiadne dynamické alokovanie pamäte (z ang. *heap memory*),
- Malý (25 KB) a čitateľný kód. Žiadne vonkajšie závislosti (z ang. *Dependencies*),
- Funguje po preklade GCC prekladačom. Bez dlhej dokumentácia, žiaden konfiguračný súbor, dodatočná konfigurácia. DSVPN je spustiteľná jednoriadkovým príkazom na serveri, obdobne na klientovi. Bez potreby konfigurácie brány firewall a pravidiel smerovania,
- Funguje na Linuxe (kernel >= 3.17), macOS a OpenBSD, DragonFly BSD, FreeBSD a NetBSD v klientskych a point-to-point režimoch. Pridanie podpory pre iné operačné systémy je triviálne,
- Nedochádza k úniku IP medzi pripojeniami, ak sa sieť nezmení. Blokuje IPv6 na klientovi, aby sa zabránilo úniku IPv6 adries.

V uvedenej VPN autor zakomponoval aj možnosť pokročilejších nastavení. Celkový súhrn vstupných parametrov pri štarte programu je takýto:

[1] https://github.com/jedisct1/dsvpn
[2] z ang. *Transmission Control Protocol*



Obr. 4.1: Schéma jednoduchej VPN

```
./dsvpn   server
<key file>
<vpn server ip or name>|"auto"
<vpn server port>|"auto"
<tun interface>|"auto"
<local tunnel ip>|"auto"
<remote tunnel ip>"auto"
<external ip>|"auto"

./dsvpn   client
<key file>
<vpn server ip or name>
<vpn server port>|"auto"
<tun interface>|"auto"
<local tunnel ip>|"auto"
<remote tunnel ip>|"auto"
<gateway ip>|"auto"
```

Väčšina parametrov je však prednastavených na automatické hodnoty. Príkladom je port 443, vytvorenie rozhrania tun0, prevziatie externej IP adresy zo siete a ďalšie.

4.1 Kryptografia použitá v DSVPN

DSVPN používa v svojej implementácií malú sebestačnú kryptografickú knižnicu – *Charm*[3]. Jej autorom je tvorca DSVPN. Implementácia umožňuje autentizované šifrovanie (z ang.*authenticated encryption*) a hašovanie kľúčov (z ang. *keyed hashing*). Správnosť implementácie algoritmu v knižnici programátor overil pomocou nástroja **Cryptol**[4]. Uvedený nástroj slúži na zápis algoritmu do matematickej špecifikácií. Tým poskytne možnosť jednoduchšej a hlavne korektnej implementácie zvoleného kryptografického algoritmu. Zároveň je možné program využiť aj na verifikáciu vytvoreného riešenia. Obdobne sú v repozitári knižnice ponechané overovacie skripty pre jednoduché spustenie.

Kryptografický algoritmus použitý v DSVPN je Xoodoo permutácia v duplex móde, pričom môže byť jednoducho nahradená napríklad Gimli-im[5] [9] alebo Simpira384[6] [10], ktorá je založené na AES-e . Uvedené algoritmy sú predmetom opisu kapitoly 2. Pri zmene musí používateľ zasiahnuť do zdrojového kódu v súbore **charm.c**, ktorého obsahom sú kryptografické primitíva.

4.2 Experimentálne overenie VPN

DSVPN sme prakticky overili pomocou dvojice virtuálnych strojov OSS a OSC, ktorých opis je obsahom 3. Na zariadení OSS sme pomocou balička make a GCC prekladača vykonali inštaláciu DSVPN. Obdobný postup je aplikovaný aj vo VM OSC. Na OSS spúšťame VPN Server, ktorý nám poskytne IP adresu, prostredníctvom ktorej budeme komunikovať s vonkajším svetom. Na spustenie a vytvorenie spojenie vykonáme nasledujúce úkony:

1. Vygenerovanie zdieľaného kľúča:

   ```
   dd if=/dev/urandom of=vpn.key count=1 bs=32
   ```

   – zdieľaný kľúč, ktorý sme vygenerovali, sa nám uložil do súboru *vpn.key*. Ten je potrebné vložiť do priečinka s programom *dsvpn* v oboch zariadeniach – OSS aj OSC.

2. OSS zariadenie:

[3] https://github.com/jedisct1/charm
[4] https://cryptol.net/index.html
[5] https://github.com/jedisct1/gimli
[6] https://github.com/jedisct1/simpira384

```
sudo ./dsvpn server vpn.key auto
2340 auto 10.8.0.254 10.8.0.2
```

– tento príkaz zabezpečí spustenie VPN servera na prostredí OSS s IP adresou, priradenou k vytvoren=emu tunelovaciemu rozhraniu s menom *tun0*. Vytvorí sa pri spustení servera[7]. Príkazom ďalej definujeme portové číslo 2340, ktoré sa použije pri nastolení TCP spojenie medzi klientom a serverom. Poslednou konfiguráciou je priradenie IP adresy tunelov, ktoré bude využívať náš klient – 10.8.0.254 a server – 10.8.0.2. Používateľ má ešte možnosť nastaviť tzv. External IP. Tú by sme využili ak by sme spúšťali DSVPN na routri poskytovateľa internetu.

3. OSC zariadenie:

```
sudo ./dsvpn client vpn.key 192.168.88.62
2340 auto 10.8.0.2 10.8.0.254
```

– uvedený príkaz zabezpečí, že sa pripojíme na VPN Server, ktorý ma ip adresu *192.168.88.62* s portom 2340. Následne vzniká TCP spojenie. Dôležité je si všimnúť poradie adries tunelov. Je opačné ako v prípade servera.

4. V prípade úspešnej konektivity sa operácia podarila a pre okolitý svet sme viditelný pomocou IP adresy, ktorú sme zvolili.

Na overenie správnosti funkcionality nám postačí jednoduchý sieťový príkaz `traceroute`. Napríklad `traceroute google.sk`[8]. Prvá z uvedených adries je práve tá, ktorú dané zariadenie používa.

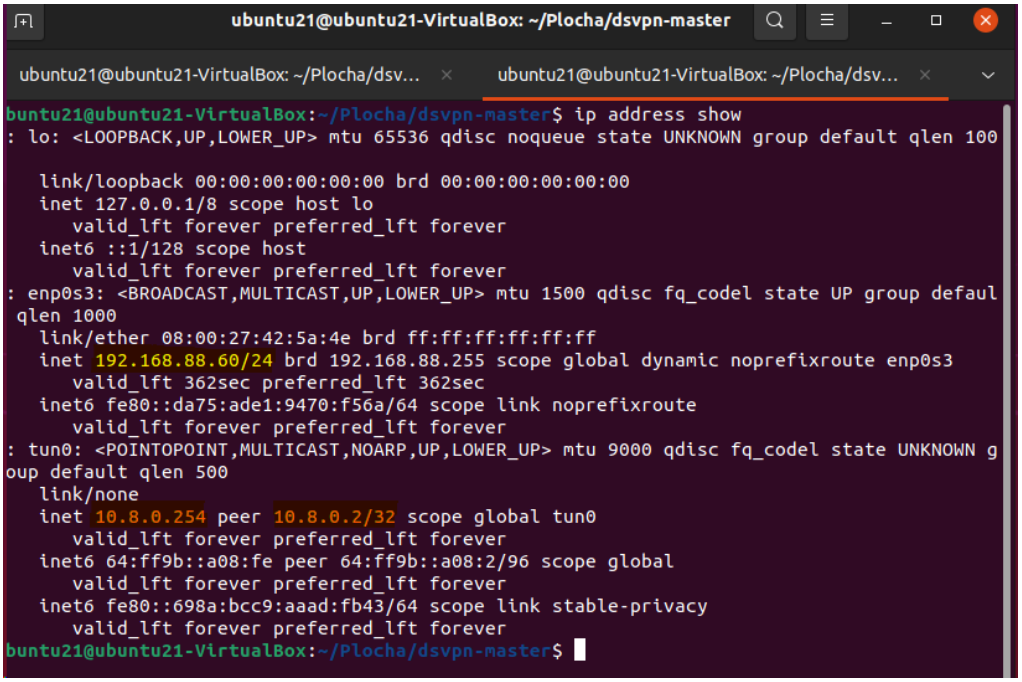
V našom prípade bolo nutné použiť lokálne adresy vzhľadom na to, že obe VM bežia na jednom hosťovskom počítači. Obidva zariadenia sú tým pádom pripojené k jednému internetovému poskytovateľovi, čo má za následok takmer rovnaké smerovanie k vzdialenej doméne.

Proces zistenia IP adresy VPN servera, po spustení, a overenie funkčnosti je následne znázornené pomocou 4.2,4.3, 4.4. V 4.2 sme žltou farbou znázornili IP adresu, na ktorej je VPN server dostupný. Oranžová farba znázorňuje IP adresy tunelu medzi serverom a klientom v tomto poradí. Následne v 4.3 môžeme vidieť to isté pre klienta.

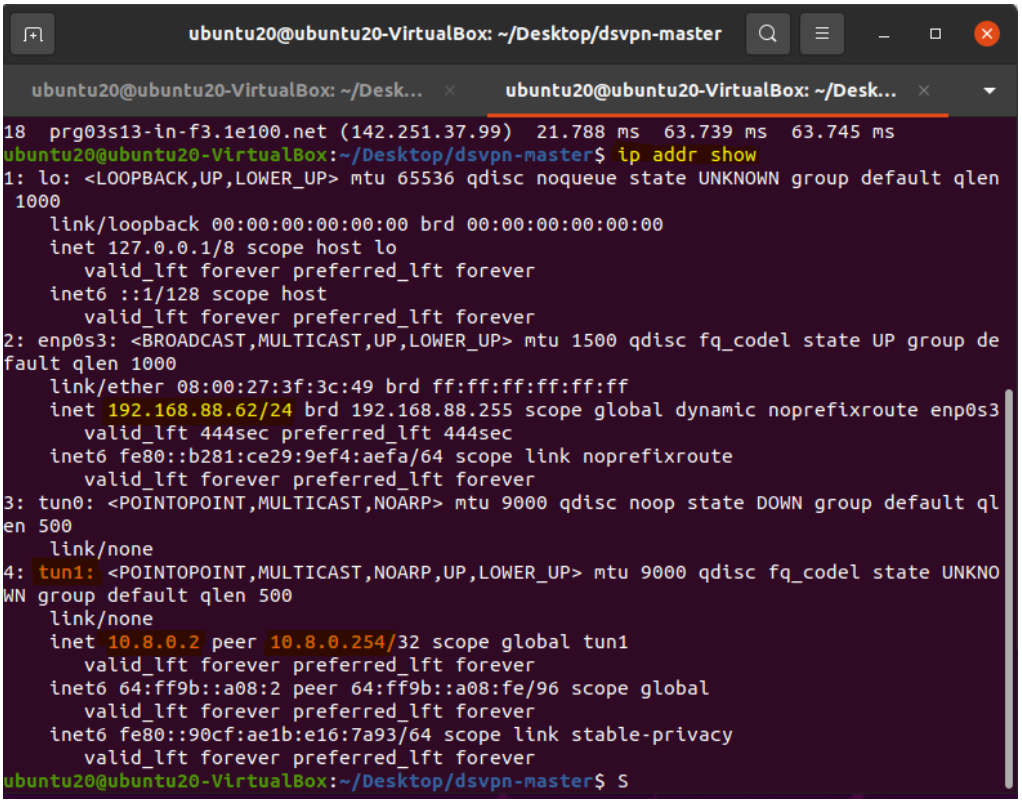
Nakoniec, v 4.4 môžeme vidieť ako klient pri internetovej komunikácií používa namiesto svojej vlastnej, adresu poskytnutú VPN Serverom na zariadení OSS – žltou zvýraznená IP. Červenou je zaškrnutá farba poskytovateľa internetu.

[7] Pomocou `ip address show tun0` zistíme IPv4 adresu VPN servera.

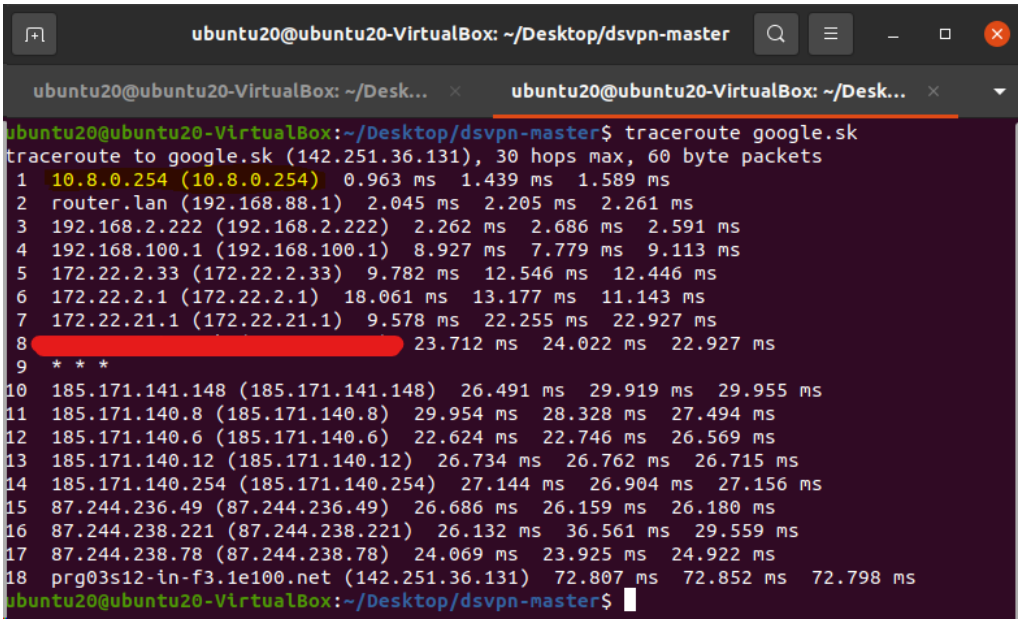
[8] vo Windows CMD prostredí: `tracert google.sk`



Obr. 4.2: Zistenie IP adries VPN Servera na VM OSS



Obr. 4.3: Zistenie IP adries VPN Klienta na VM OSC



Obr. 4.4: Overenie funkcionality DSVPN pomocou traceroute

4.3 Analýza zdrojového kódu DSVPN

Pri analýze sa zameriame výhradne na dôležité časti kódu DSVPN pre programovací jazyk C. Repozitár pozostáva z jedného make-file balíčka[31]. 3 hlavičkových (.h) a k ním korešpondujúcimi zdrojovými kódmi (.c), s pomenovaním:

1. **charm.h** – kryptografická knižnica so 6 funkciami,
2. **os.h** – funkcie čítania a zápisu paketov, vytvorenia, aplikácie a zrušenia tunelu,
3. **vpn.h** – deklarácia konštánt, endianity[32] a niektorých závislostí OS.

V spomenutých hlavičkových súboroch sa nachádzajú deklarácie funkcií, ktoré VPN používa. Definície sú obsahom .c súborov, ako je v jazyku C zaužívaným zvykom. Obsahom tejto podkapitoly je analýza týchto kódov.

4.3.1 Súbor charm.h a charm.c

Obsahom sú prevažne funkcie slúžiace pri behu kryptografického algoritmu XOODOO. Charm.h pozostáva z 6 funkcií. Ich implementácia nie je až tak rozsiahla. Zaberá celkovo 337 riadkov. Jednotlivú implementáciu každej funkcie postupne opíšeme.

Zdrojový kód 4.1: Obsah charm.h

```
1  void uc_state_init(uint32_t st[12], const unsigned char key[32],
2                     const unsigned char iv[16]);
3  void uc_encrypt(uint32_t st[12], unsigned char *msg,
4                  size_t msg_len, unsigned char tag[16]);
5  int uc_decrypt(uint32_t st[12], unsigned char *msg,
6                  size_t msg_len,
7                  const unsigned char *expected_tag,
8                  size_t expected_tag_len);
9  void uc_hash(uint32_t st[12], unsigned char h[32],
10              const unsigned char *msg, size_t len);
11 void uc_memzero(void *buf, size_t len);
12 void uc_randombytes_buf(void *buf, size_t len);
```

4.3.2 Súbor os.h a os.c

Obsahom su funkcie, ktorých úlohami sú čítanie alebo pridanie GW, vytvorenie a nastavenie tunelu v danom OS. Následne je aplikovaná úprava firewall pravidiel, tak aby všetka komunikácia bola presmerovaná na VPN server. Úprava firewall pravidiel je závislá od role, pod ktorou je VPN spustená. Teda či sa jedná o server alebo klienta. Celkovo je obsahom 12 funkcií, ktoré riešia uvedené úlohy. Detail je možné vidiet v 4.2.

Zdrojový kód 4.2: Obsah ZK os.h

```
ssize_t safe_read(const int fd, void *const buf_, size_t count,
                  const int timeout);
ssize_t safe_write(const int fd, const void *const buf_,
                   size_t count,
                   const int timeout);
ssize_t safe_read_partial(const int fd, void *const buf_,
                          const size_t max_count);
ssize_t safe_write_partial(const int fd, void *const buf_,
                           const size_t max_count);

typedef struct Cmds {
 const char *const *set;
 const char *const *unset;
} Cmds;

Cmds firewall_rules_cmds(int is_server);
int shell_cmd(const char *substs[][2], const char *args_str,
              int silent);
const char *get_default_gw_ip(void);
const char *get_default_ext_if_name(void);
int tcp_opts(int fd);
int tun_create(char if_name[IFNAMSIZ], const char *wanted_name);
int tun_set_mtu(const char *if_name, int mtu);
ssize_t tun_read(int fd, void *data, size_t size);
ssize_t tun_write(int fd, const void *data, size_t size);
```

4.3.3 Súbor vpn.h a vpn.c

Hlavičkový súbor obsahuje prevažne definovanie niektorých parametrov potrebných na správnu funkcionalitu VPN, spoločne s korekciou pre niektoré OS. Viď. 4.3. Ostatné parametre, ktoré bolí opísané pri spustení su definované práve v tomto súbore (MTU,Porty,IP atď.).

Zdrojový kód 4.3: Obsah ZK vpn.h

```
1  /*UNIX-like OS Dependent Libraries*/
2  #include <sys/ioctl.h>
3  #include <sys/socket.h>
4  #include <sys/types.h>
5  #include <sys/uio.h>
6  #include <sys/wait.h>
7  #include <net/if.h>
8  #include <netinet/in.h>
9  #include <netinet/tcp.h>
10 /*End UNIX-like OS Dependent Libraries*/
11 /*OS setup dependencies*/
12 #ifdef __linux__
13 #include <linux/if_tun.h>
14 #endif
15
16 #ifdef __APPLE__
17 #include <net/if_utun.h>
18 #include <sys/kern_control.h>
19 #include <sys/sys_domain.h>
20 #endif
21
22 #ifdef __NetBSD__
23 #define DEFAULT_MTU 1500
24 #else
25 #define DEFAULT_MTU 9000
26 #endif
27 /*End of OS setup dependencies*/
```

Main() – Beh programu

Pred samotným opisom by som rád upozornil na jeden fakt. Autor DSVPN používa vo veľkej miere zápis pomocou tzv. ternárnych operátorov. Viac informácií o tejto problematike je možné nájsť v [33].

Vpn.c je hlavným zdrojovým kódom DSVPN. V jeho vnútri nájdeme hlavnú, štartovaciu, funkciu main. Zároveň má prilinkované aj vyššie uvedené knižnice. Ako prvé dochádza k inicializácií premennej štruktúry `Context`4.4.

Zdrojový kód 4.4: Štruktúra Context

```
typedef struct Context_ {
  const char *  wanted_if_name;
  const char *  local_tun_ip;
  const char *  remote_tun_ip;
  const char *  local_tun_ip6;
  const char *  remote_tun_ip6;
  const char *  server_ip_or_name;
  const char *  server_port;
  const char *  ext_if_name;
  const char *  wanted_ext_gw_ip;
  char          client_ip[NI_MAXHOST];
  char          ext_gw_ip[64];
  char          server_ip[64];
  char          if_name[IFNAMSIZ];
  int           is_server;
  int           tun_fd;
  int           client_fd;
  int           listen_fd;
  int           congestion;
  int           firewall_rules_set;
  Buf           client_buf;
  struct pollfd fds[3];
  uint32_t      uc_kx_st[12];
  uint32_t      uc_st[2][12];
} Context;
```

Následne dochádza k načítaniu kľúča pomocou pomocnej funkcie `load_key_file()`4.5. Úlohou je prepočítanie zdieľaného kľúča, pričom sa používa funkcia z os.c – `safe_read()`. Realizuje sa znakové spočítanie, v ktorom je zakomponovaná funkcia `poll()`[34]. V prípade zhody veľkosti, funkcia vracia 0.

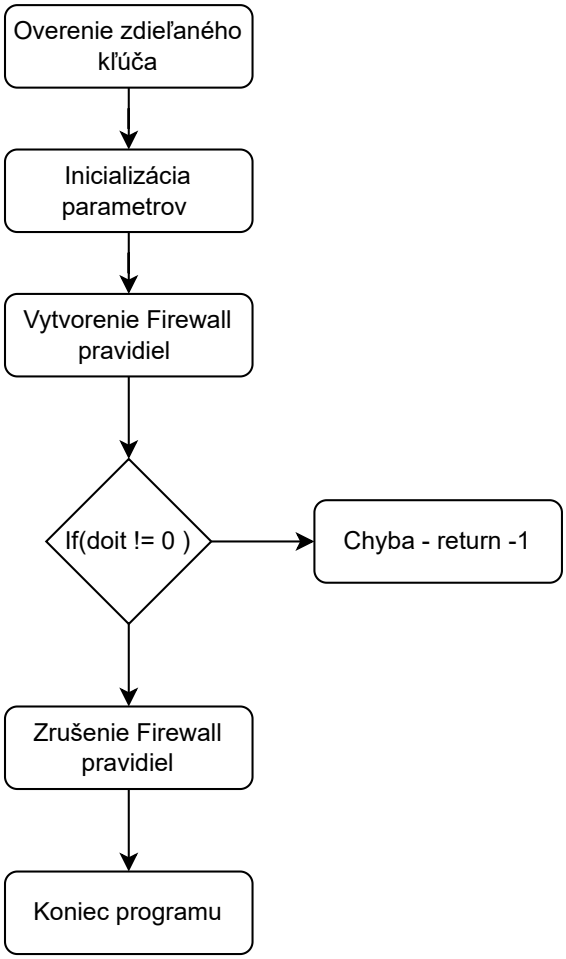
Zdrojový kód 4.5: Načítanie zdieľaného kľúča

```
1  static int load_key_file(Context *context, const char *file)
2  {
3   unsigned char key[32];
4   int           fd;
5
6   if ((fd = open(file, O_RDONLY)) == -1) {
7     return -1;
8   }
9   if (safe_read(fd, key, sizeof key, -1) != sizeof key) {
10    (void) close(fd);
11    return -1;
12  }
13  uc_state_init(context->uc_kx_st, key,
14                (const unsigned char *) "VPN Key Exchange");
15  uc_memzero(key, sizeof key);
16
17  return close(fd);
18 }
```

Po preverení kľúča dochádza k priradeniu parametrov do štruktúry `Context` na základe vstupu pri spustení DSVPN. V prípade že nedochádza k zmene GateWay (ďalej GW) IP adresy, tak sa priradí pôvodná. Tá sa získa pomocou `get_default_gw_ip()`, ktorá je deklarovaná v 4.2. Prostredníctvom shell príkazu `ip route...` a `read_from_shell_command()` funkcie, dochádza k extrakcii informácií priamo z príkazového riadka. Tento krok je teda závislý od OS, v ktorom používateľ pracuje. Nasleduje overenie návratových hodnôt z funkcií iba v prípade ak je DSVPN spustené ako Klient.

Program v prípade servera pokračuje s `get_default_ext_if_name()`. Obdobne ako v predchádzajúcom odstavci je realizácia funkcionality, vykonaná pomocou terminálu. Podstata spočíva v zistení mena rozhrania, ktoré bude presmerované na VPN server.

Po nastavení parametrov sa dostávame k vytvoreniu tunelovacieho rozhrania. V tomto kroku autor používa funckiu `tun_create`. Úloha je vysoko závislá od OS. Dôsledkom toho je možné vidieť vetvenie funkcionality vzhľadom k bežiacemu OS. Ako sa už spomínalo v úvode kapitoly. DSVPN poskytuje kompatibilitu pre 6 OS, medzi ktoré patrí Linux, FreeBSD, NetBSD, OpenBSD, MacOS, DragonFly. Následne na základe systému dochádza k vytváraniu tunelu s prednastaveným,



Obr. 4.5: Beh DSVPN

resp. zvoleným menom. Program ďalej nastaví hodnotu MTU.

Po príprave tunelu ešte repetitívne dochádza k overeniu. Tento krok vykonáva funkcia `resolve_ip`. Tá ma v sebe vnorené 2 funkcie, ktoré su menným ekvivalentom aj vo Windows knižniciach. Jedná sa o funkcie `getaddrinfo` a `getnameinfo`.

Posledným krokom súvisiacim s konfiguráciou prostredia je vytvorenie pravidla pre branu FireWall (ďalej FW). Tento úkon realizuje `firewall_rules()`. Tento proces je opäť systémovo závislý. Jeho realizácia je vykonaná pomocou globálne definovanej funkcie `Cmds firewall_rules_cmds()`. Tá obsahuje súbor prednastavených príkazov. Ich úlohou je presmerovanie celej premávky cez novovzniknutý tunel.

Posledným úkonom je samotný beh VPN. Ten spúšťa funkcia `doit()`. Doterajší beh je jednoducho znázornený pomocou flow diagramu 4.5.

Od tohto momentu sa presúvame k postupnému vnáraniu do procesu spojenia a spracovania dát. Vo vnútri `doit()` dochádza k vetveniu programu. V prípade, že je DSVPN spustená ako server spustí sa funkcia `tcp_listener()`. V

opačnom prípade `client_reconnect()`. Ulohami funkcií je nastolenie TCP spojenia medzi clientom a serverom. Listener vytvára socket pomocou systemovej funkcie `bind()`, ktorý následne čaká na klienta. Smerník na socket je uložený do štruktúry Context. V tejto vetve sa v Context ešte nastaví smerovanie na tento socket. `Client_reconnect()` podľa prednastaveného počtu pokusov o znovu nadviazanie spojenia sa pokúša pomocou `client_connect()` o spojenie. Pred týmto úkonom samozrejme dochádza k overeniu či už nedošlo k nadviazaniu spojenia a o to sa stará `client_disconnect()`.

`Client_connect()` slúži na pripojenie klienta k serveru. Vykonáva úpravu pravidiel v FW. Následne sa pokúša o nadviazania spojenia pomocou funkcie `tcp_client()`. Po tejto sérií úloh sa vraciame opäť do `doit()`. Tu je vo **`while`** cykle vykonávaná funkcia `event_loop()`, v ktorej dochádza k použitiu kryptografickej knižnice charm. Jej obsahom je inicializácia premenných. viď. 4.6. Schéma 4.6 znázorňuje opísané skutočnosti.

Zdrojový kód 4.6: Premenné funkcie event loop

```
1     struct pollfd *const fds = context->fds;
2   Buf                  tun_buf;
3   Buf *                client_buf = &context->client_buf;
4   ssize_t              len;
5   int                  found_fds;
6   int                  new_client_fd;
```

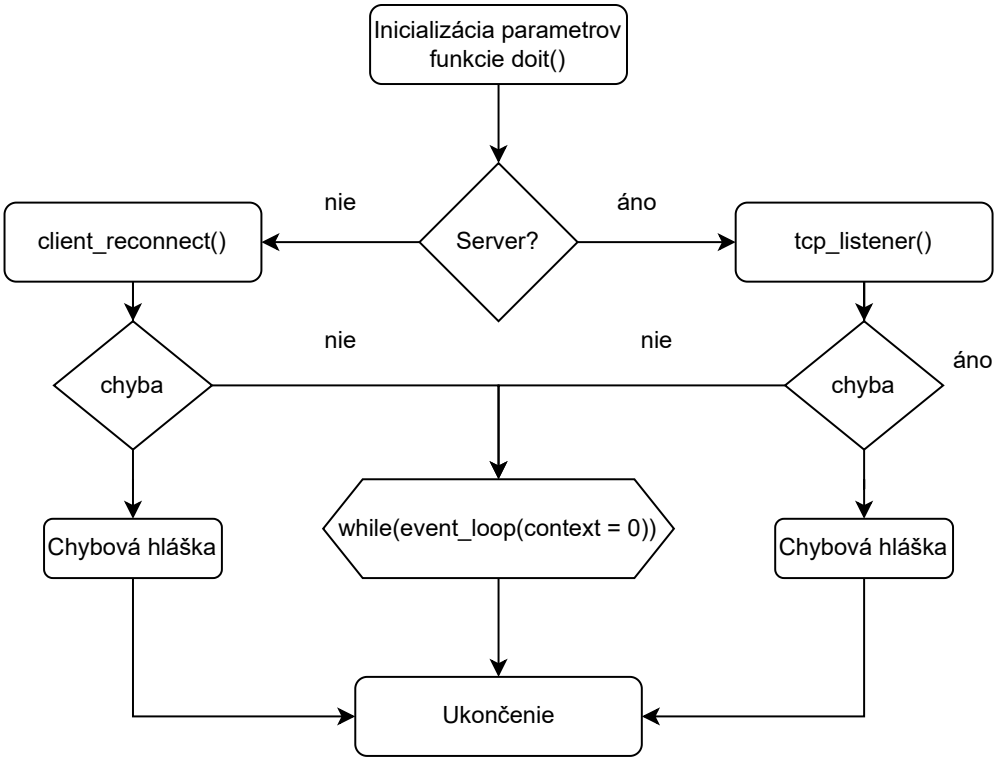
Funckia event_loop()

Event_loop je 111 riadkov dlhá funkcia. Jej obsah môžeme rozdeliť na overovací a výkonový. Úlohou niekoľkých if-ou vo funkcii je preverovanie signálov, spätných hodnôt a podobných premenných, ktoré by signalizovali chybu alebo používateľov záujem o ukončenie relácie programu.

Zaujímavé je taktiež predom definované makro `BUFFERBLOAT_CONTROL`. Jeho úlohou je zamedzenie problému zvaného **Bufferbloat**. V skratke, je to nechcený jav, ktorý je zapríčinený nadmerným ukladaním paketov do vyrovnávacej pamäte, tzv. zahltenie. To ma za následok vysokú latenciu a tzv. Packet Delay Variation – PDV (Jitter), v paketovo-orientovaných sieťach. Viac o tejto problematike je možne si prečítať na [35].

Na druhej strane výkonové funkcie, ako hovorí ich názov. niečo vykonávajú. Do tejto kategórie sme zaradili funkcie:

- `tcp_accept()` – slúži na nastolenie nového TCP spojenia s klientom,



Obr. 4.6: Funkcia Doit()

- `tun_read()` – volá `safe_read_partial` v prípade linuxového OS,
- `uc_encrypt()` – kryptografické šifrovanie správy,
- `safe_write_partial()` – používa štandardizovanú funkciu write vo while cykle, zapíše zašifrované dáta do buffera určeného pre odoslanie klientovi, vracia počet zapísaných dát,
- `safe_write()` – používa sa v prípade ak došlo k zahlteniu paketmi,
- `client_reconnect()` – slúži na obnovu spojenia v prípade chyby,
- `safe_read_partial()` – obdobne ako pri write, používa read funkciu,
- `uc_decrypt()` – kryptografické dešifrovanie správy,
- `tun_write()` – volá `safe_write` pri OS Linux.

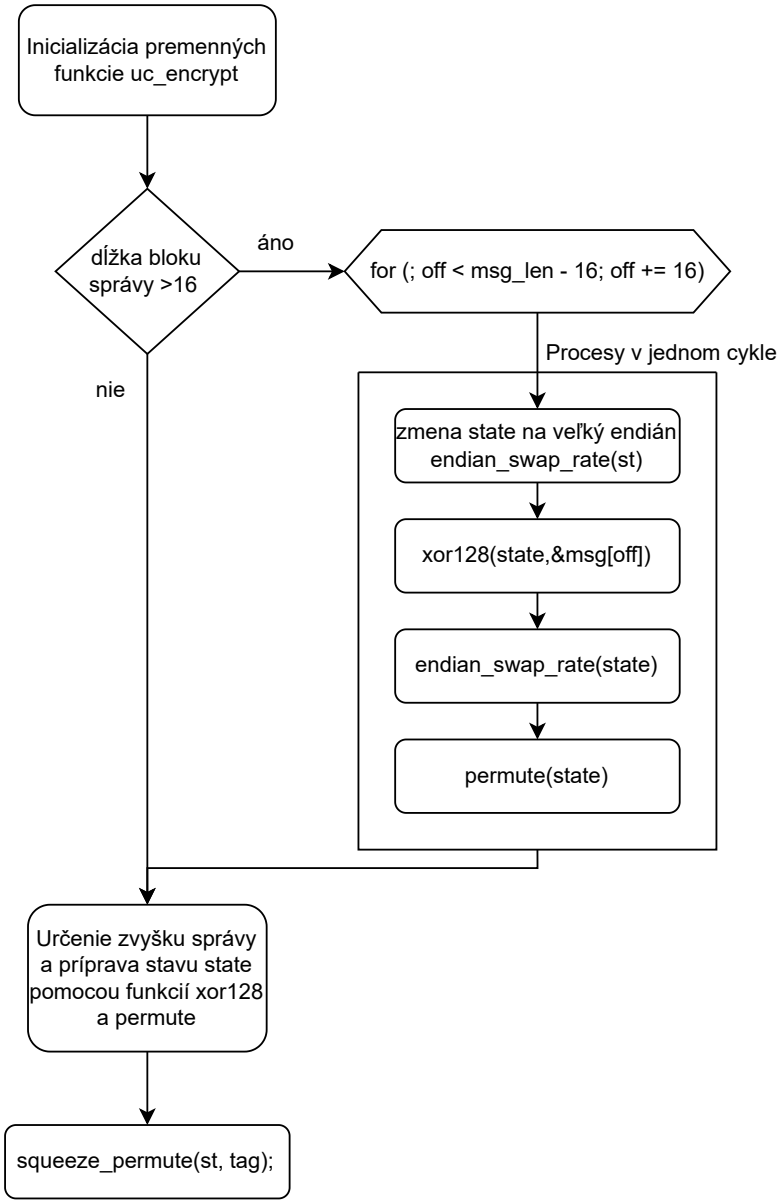
Metóda použitá pri zápise zašifrovaných dát vo funkcii `event_loop()` je znázornená v 4.7.

Zdrojový kód 4.7: Spôsob zápisu šifrovaných dát

```
1
2  writenb = safe_write_partial(context->client_fd, tun_buf.len,
3                               2U + TAG_LEN + len);
4  if (writenb < (ssize_t) 0) {// kontrola zahltenia -- bufferbloat
5   context->congestion = 1;
6   writenb             = (ssize_t) 0;
7  }
8 // ak je iny objem ako maximum
9  if (writenb != (ssize_t)(2U + TAG_LEN + len)) {
10  writenb = safe_write(context->client_fd, tun_buf.len + writenb,
11  2U + TAG_LEN + len - writenb, TIMEOUT);
12 }
```

Šifrovanie a dešifrovanie

Proces šifrovanie resp. dešifrovania správy nastáva na oboch stránách siete, teda pri klientovy aj serveri. 4.8 demonštruje šifrovanie implementované v funkcii `uc_encrypt()`. Tento proces sme sa pokúsili opísať v grafe 4.7.



Obr. 4.7: Funkcia uc_encrypt

Zdrojový kód 4.8: Šifrovanie správy

```
1
2  void uc_encrypt(uint32_t st[12], unsigned char *msg,
3                  size_t msg_len, unsigned char tag[16])
4  { //spracovanie po 16 znakov
5    unsigned char squeezed[16];
6    unsigned char padded[16 + 1];
7    size_t        off = 0;
8    size_t        leftover;
9
10   if (msg_len > 16) {
11     for (; off < msg_len - 16; off += 16) {
12       endian_swap_rate(st);
13       memcpy(squeezed, st, 16);
14       xor128(st, &msg[off]);
15       endian_swap_rate(st);
16       xor128(&msg[off], squeezed);
17       permute(st);
18     }
19   }
20   leftover = msg_len - off;
21   memset(padded, 0, 16);
22   mem_cpy(padded, &msg[off], leftover);
23   padded[leftover] = 0x80;
24   endian_swap_rate(st);
25   memcpy(squeezed, st, 16);
26   xor128(st, padded);
27   endian_swap_rate(st);
28   st[11] ^= (1UL << 24 | (uint32_t) leftover >> 4 << 25
29              | 1UL << 26);
30   xor128(padded, squeezed);
31   mem_cpy(&msg[off], padded, leftover);
32   permute(st);
33   squeeze_permute(st, tag);
34 }
```

Ako je viditeľne v kóde dochádza k častému použitiu 2 funkcií. Nimi sú `xor128()` a `permute()`. Jedná sa o pomerne dôležité bloky pre správne fungovanie šifrovacieho algoritmu. Obsah prvej z uvedených je preto znázornený v 4.9.

Zdrojový kód 4.9: Funkcia xor128

```
1  static inline void xor128(void *out, const void *in)
2  {
3    #ifdef __SSSE3__
4    _mm_storeu_si128((__m128i *) out,
5    _mm_xor_si128(_mm_loadu_si128((const __m128i *) out),
6    _mm_loadu_si128((const __m128i *) in)));
7    #else
8    unsigned char *      out_ = (unsigned char *) out;
9    const unsigned char *in_  = (const unsigned char *) in;
10   size_t               i;
11
12   for (i = 0; i < 16; i++) {  //xororvanie jednotlivych znakov
13     out_[i] ^= in_[i];    //v 16 bitovom bloku spravy
14   }
15   #endif
16 }
```

Na druhej strane `permute()` je pomerne rozsiahla funkcia pričom jej obsah sa rozprestiera na 100 riadkoch. Funkcionalita závisí od procesorových inštrukcií. Avšak nás bude zaujímať softvérová implementácia, ktorá je aj použitá pri behu. Dôvodom je, že naše zariadenie nemá k dispozícií uvedené procesorové inštrukcie. Blok, ktorý je použitý pri volaní sa nachádza v 4.10.

Zdrojový kód 4.10: Funkcia Permute + makrá

```
#define ROTR32(x, b) (uint32_t)(((x) >> (b)) | ((x) << (32 - (b))))
#define SWAP32(s, u, v)        \
do {                           \
  t      = (s)[u];             \
  (s)[u] = (s)[v], (s)[v] = t; \
} while (0)

static void permute(uint32_t st[12])
{
  uint32_t e[4], a, b, c, t, r, i;
  for (r = 0; r < XOODOO_ROUNDS; r++) {
    for (i = 0; i < 4; i++) {
      e[i] = ROTR32(st[i] ^ st[i + 4] ^ st[i + 8], 18);
      e[i] ^= ROTR32(e[i], 9);
    }
    for (i = 0; i < 12; i++) {
      st[i] ^= e[(i - 1) & 3];
    }
    SWAP32(st, 7, 4);
    SWAP32(st, 7, 5);
    SWAP32(st, 7, 6);
    st[0] ^= RK[r];
    for (i = 0; i < 4; i++) {
      a         = st[i];
      b         = st[i + 4];
      c         = ROTR32(st[i + 8], 21);
      st[i + 8] = ROTR32((b & ~a) ^ c, 24);
      st[i + 4] = ROTR32((a & ~c) ^ b, 31);
      st[i] ^= c & ~b;
    }
    SWAP32(st, 8, 10);
    SWAP32(st, 9, 11);
  }
}
```

5 Populárne VPN

V súčasnosti je trend, využívanie VPN na súkromné účely. Dôvodov môže byť viacero. Napríklad prístup k lokálne blokovaným doménam, anonymita v internetovom prostredí, zabezpečenie pripojenia a iné. V tomto prípade vstupujú do popredia tzv. VPN poskytovatelia (z ang. *VPN Providers*). Ktorý za poplatok poskytujú výhody pripojenia cez VPN k verejnej sieti.

opis fungovania, demonštrácia v prostredi VB.

5.1 OpenVPN

5.2 WireGuard

5.3 Porovnanie – mohlo by sa, ak sa stihne a bude vediet

6 Vyhodnotenie dosiahnutých výsledkov

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisissem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi necante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultriciesvel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero utmetus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit ametante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

7 Záver

Plán budúcej práce:

- Študium a opis kódu DSVPN
- Štúdium a opis funkcionality XOODOO,gimli atd.
- Implementacia inych kryptograf. blokov do DSVPN
- vytvorenie kompatibility DSVPN na OS Windows

Vo výsledku by sme mali mať k dispozícií funkcnu VPN, upravenú o kompatibilitu s OS windowse aj s moznostou vyberu kryp. algoritmov.

Literatúra

1. SRIDEVI, Sridevi; D H, Manjaiah. Technical Overview of Virtual Private Networks(VPNs). *International Journal of Scientific Research*. 2012, roč. 2, s. 93–96. Dostupné z DOI: 10.15373/22778179/JULY2013/32. [Online; Citované: 22.1.2022].
2. Virtual Private Networks Simplified. [B.r.]. Dostupné tiež z: https://www.cisco.com/c/dam/en_us/training-events/le21/le34/downloads/689/academy/2008/sessions/BRK-134T_VPNs_Simplified.pdf. [Online; Citované:26.1.2022].
3. COMPUTER SCIENCE UPJS, Institute of. 5. Prednáška - Transportná vrstva: Protokol TCP. [B.r.]. Dostupné tiež z: https://siete.ics.upjs.sk/prednaska-5/. [Online; Citované:6.2.2022].
4. DRUTAROVSKÝ, Miloš. Bezpečnosť v architektúre TCP/IP II (BIKS pr7). [B.r.]. [Online; Citované:6.2.2022].
5. Transport Layer Security. [B.r.]. Dostupné tiež z: https://en.wikipedia.org/wiki/Transport_Layer_Security#TLS_1.3. [Online; Citované: 21.5.2021].
6. Advanced Encryption Standard. [B.r.]. Dostupné tiež z: https://en.wikipedia.org/wiki/Advanced_Encryption_Standard. [Online; Citované: 8.5.2021].
7. LEVICKÝ, Dušan. *Kryptografia v informačnej bezpečnosti*. Elfa, 2005.
8. DAEMEN, Joan; HOFFERT, Seth; MELLA, Silvia; PEETERS, Michaël; VAN ASSCHE, Gilles; VAN KEER, Ronny. *Xoodyak*. Team Keccak, 2020. Dostupné tiež z: https://keccak.team/xoodyak.html. [Online; Citované:6.2.2022].
9. NIST LIGHTWEIGHT CRYPTOGRAPHY STANDARDIZATION PROCESS, Publication to. Gimli 2019-03-29. [B.r.]. Dostupné tiež z: https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/gimli-spec.pdf. [Online; Citované:6.2.2022].

10. SHAY GUERON, Nicky Mouha. Simpira v2: A Family of Efficient Permutations Using the AES Round Function. [B.r.]. Dostupné tiež z: `https://hal.inria.fr/hal-01403414/document`. [Online; Citované:6.2.2022].

11. DAEMEN, Joan; HOFFERT, Seth; MELLA, Silvia; PEETERS, Michaël; VAN ASSCHE, Gilles; VAN KEER, Ronny. *The Keccak-p Permutations*. Team Keccak, 2020. Dostupné tiež z: `https://keccak.team/specifications.html`. [Online; Citované:6.2.2022].

12. DAEMEN, Joan; HOFFERT, Seth; MELLA, Silvia; PEETERS, Michaël; VAN ASSCHE, Gilles; VAN KEER, Ronny. *Kravatte*. Team Keccak, 2020. Dostupné tiež z: `https://keccak.team/kravatte.html`. [Online; Citované:6.2.2022].

13. DAEMEN, Joan; HOFFERT, Seth; MELLA, Silvia; PEETERS, Michaël; VAN ASSCHE, Gilles; VAN KEER, Ronny. *The Keccak-p Permutations*. Team Keccak, 2020. Dostupné tiež z: `https://keccak.team/keccakp.html`. [Online; Citované:6.2.2022].

14. BERNSTEIN, Daniel J; KÖLBL, Stefan; LUCKS, Stefan; MASSOLINO, Pedro Maat Costa; MENDEL, Florian; NAWAZ, Kashif; SCHNEIDER, Tobias; SCHWABE, Peter; STANDAERT, François-Xavier; TODO, Yosuke et al. Gimli: a cross-platform permutation. In: *International Conference on Cryptographic Hardware and Embedded Systems*. Springer, 2017, s. 299–320.

15. DAEMEN, Joan; HOFFERT, Seth; MELLA, Silvia; PEETERS, Michaël; VAN ASSCHE, Gilles; VAN KEER, Ronny. *Farfalle: parallel permutation-based cryptography*. Team Keccak, 2020. Dostupné tiež z: `https://keccak.team/farfalle.html`. [Online; Citované:6.2.2022].

16. DAEMEN, Joan; HOFFERT, Seth; MELLA, Silvia; PEETERS, Michaël; VAN ASSCHE, Gilles; VAN KEER, Ronny. *The sponge and duplex constructions*. Team Keccak, 2020. Dostupné tiež z: `https://keccak.team/sponge_duplex.html`. [Online; Citované:6.2.2022].

17. JOAN DAEMEN Seth Hoffert, Gilles Van Assche; KEER, Ronny Van. The design of Xoodoo and Xoofff. 2018, roč. 2018. Dostupné z DOI: `10.13154/tosc.v2018.i4.1-38`. [Online; Citované:6.2.2022].

18. DAEMEN, Joan; HOFFERT, Seth; MELLA, Silvia; PEETERS, Michaël; VAN ASSCHE, Gilles; VAN KEER, Ronny. *Xoodoo cookbook*. Team Keccak, 2020. Dostupné tiež z: `https://eprint.iacr.org/2018/767.pdf`. [Online; Citované:6.2.2022].

19. DAEMEN, Joan; HOFFERT, Seth; PEETERS, Michaël; ASSCHE, Gilles Van; KEER, Ronny Van. *Xoodoo cookbook* (*2.revision*) [Cryptology ePrint Archive, Report 2018/767]. 2019. Dostupné tiež z: `https://eprint.iacr.org/2018/767.pdf`. [Online; Citované:6.2.2022].

20. DAEMEN, Joan; HOFFERT, Seth; MELLA, Silvia; PEETERS, Michaël; VAN ASSCHE, Gilles; VAN KEER, Ronny. *Xoodyak, an update*. Publication to NIST Lightweight Cryptography Standardization Process (round ..., 2020. Dostupné tiež z: `https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/status-update-sep2020/Xoodyak-update.pdf`. [Online; Citované:6.2.2022].

21. DAEMEN, Joan; HOFFERT, Seth; MELLA, Silvia; PEETERS, Michaël; VAN ASSCHE, Gilles; VAN KEER, Ronny. *Xoodyak*. Publication to NIST Lightweight Cryptography Standardization Process, 2020. Dostupné tiež z: `https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/Xoodyak-spec.pdf`. [Online; Citované:6.2.2022].

22. DAEMEN, Joan; HOFFERT, Seth; MELLA, Silvia; PEETERS, Michaël; VAN ASSCHE, Gilles; VAN KEER, Ronny. *The Keyak authenticated encryption scheme*. Team Keccak, 2020. Dostupné tiež z: `https://keccak.team/keyak.html`. [Online; Citované:6.2.2022].

23. What is a ratchet? [B.r.]. Dostupné tiež z: `https://crypto.stackexchange.com/questions/39762/what-is-a-ratchet`. [Online; Citované: 8.5.2021].

24. Forward secrecy. [B.r.]. Dostupné tiež z: `https://en.wikipedia.org/wiki/Forward_secrecy`. [Online; Citované: 8.5.2021].

25. Security level. [B.r.]. Dostupné tiež z: `https://en.wikipedia.org/wiki/Security_level`. [Online; Citované: 8.5.2021].

26. Oracle VM Virtual Box. [B.r.]. Dostupné tiež z: `https://en.wikipedia.org/wiki/VirtualBox`. [Online; Citované: 20.1.2022].

27. MIKETHETECH. Installing Windows 10 on Virtualbox 6.1.12 – FULL PROCESS, 2020 – video tutorial. 2020. Dostupné tiež z: `https://www.youtube.com/watch?v=gKQvaPejxpc&ab_channel=MikeTheTech`. [Online; Citované: 17.5.2021].

28. ORACLE. 6.2. Introduction to Networking Modes. [B.r.]. Dostupné tiež z: `https://www.virtualbox.org/manual/ch06.html`. [Online; Citované:26.1.2022].

29. BOSE, Michael. VirtualBox Network Settings: Complete Guide. 2019. Dostupné tiež z: `https://www.nakivo.com/blog/virtualbox-network-setting-guide/`. [Online; Citované:26.1.2022].

30. WIKIPEDIA. Transmission Control Protocol. [B.r.]. Dostupné tiež z: `https://en.wikipedia.org/wiki/Transmission_Control_Protocol`. [Online; Citované:26.1.2022].

31. ORG., GNU. GNU Make Manual. [B.r.]. Dostupné tiež z: `https://www.gnu.org/software/make/manual/html_node/index.html`. [Online; Citované:20.2.2022].

32. WIKIPEDIA. Endianita. [B.r.]. Dostupné tiež z: `https://sk.wikipedia.org/wiki/Endianita`. [Online; Citované:20.2.2022].

33. FREECODECAMP.ORG. Ternary Operator in C Explained. [B.r.]. Dostupné tiež z: `https://www.freecodecamp.org/news/c-ternary-operator/`. [Online; Citované:20.2.2022].

34. MAN7ORG. Poll. [B.r.]. Dostupné tiež z: `https://man7.org/linux/man-pages/man2/poll.2.html`. [Online; Citované:20.2.2022].

35. Bufferbloat. [B.r.]. Dostupné tiež z: `https://en.wikipedia.org/wiki/Bufferbloat`. [Online; Citované: 1.3.2021].

Zoznam príloh

Príloha A CD médium – viď. obsah CD média

A Obsah CD Média

Obsah tohto média je dostupný na gite:

- `https://github.com/mr171hg/DiplomaProject`

Pouzite obrazy, zdrojove kody...