FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

# ZADÁNÍ DIPLOMOVÉ PRÁCE

| | |
|---|---|
| **Název:** | Zjednodušení výběru šifrových sad v protokolech SSL/TLS |
| **Student:** | Bc. Otto Hollmann |
| **Vedoucí:** | Ing. Josef Kokeš |
| **Studijní program:** | Informatika |
| **Studijní obor:** | Počítačová bezpečnost |
| **Katedra:** | Katedra informační bezpečnosti |
| **Platnost zadání:** | Do konce letního semestru 2020/21 |

## Pokyny pro vypracování

1) Seznamte se s problematikou protokolů SSL/TLS a jejich implementacemi (SSL knihovnami).
2) Zaměřte se na problematiku použití kryptografie v těchto protokolech. Nastudujte její bezpečnostně-implementační aspekty, zejm. otázku výběru šifer pro jednotlivé verze protokolu TLS.
3) Na knihovně OpenSSL ukažte praktické dopady svých zjištění.
4) Navrhněte úpravy, které pozorované nedostatky řeší. Soustřeďte se při tom na minimalizaci zásahů do protokolů jako takových.
5) Implementujte navržené řešení do knihovny OpenSSL.
6) Diskutujte dosažené výsledky.

## Seznam odborné literatury

Dodá vedoucí práce.

prof. Ing. Róbert Lórencz, CSc.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 25. září 2019

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

# Zjednodušení výběru šifrových sad v protokolech SSL/TLS

***Bc. Otto Hollmann***

Katedra informační bezpečnosti
Vedoucí práce: Ing. Josef Kokeš

7. ledna 2020

# Poděkování

Děkuji svému vedoucímu práce Ing. Josefu Kokešovi za odborné vedení, cenné rady a připomínky při psaní této práce. Dále bych rád poděkoval své rodině za podporu během mého studia a psaní této práce.

# Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 7. ledna 2020 .....

České vysoké učení technické v Praze
Fakulta informačních technologií


### Odkaz na tuto práci

Hollmann, Otto. *Zjednodušení výběru šifrových sad v protokolech SSL/TLS.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

# Abstrakt

Tato práce se zabývá problematikou výběru šifrových sad v jednotlivých kryptografických knihovnách. Správný výběr šifer a jejich parametrů má zásadní vliv na úroveň zabezpečení šifrovaných dat a dobu potřebnou k prolomení použitých šifrovacích mechanismů. Navíc je potřeba zohlednit preference a možnosti obou komunikujících stran. Práce se zaměřuje zejména na knihovnu OpenSSL, kde se používají dvě odlišná rozhraní pro zadávání šifrových sad.

Výsledkem je návrh a následná implementace jednotného rozhraní pro tuto knihovnu, kdy je možné pomocí původního rozhraní *cipher_list* zadávat všechny šifrové sady. Dále byla přidána možnost pro omezení platnosti šifrového aliasu na vybrané verze protokolu a možnost posunutí šifrového aliasu nebo sady na začátek seznamu. Pro výběr šifrovéh sady v závislosti na preferenci klienta i serveru byla přidána do serverového seznamu podpora skupin o stejné preferenci a možnost zadávat příznaky preference, které jsou podobné volbě *-prioritize_chacha*. To vše při zachování zpětné kompatibility.

**Klíčová slova** výběr šifer, šifrový seznam, OpenSSL, TLS 1.3

# Abstract

The primary focus of this thesis is centered around the process of selecting suitable cipher suites from the available options present in cryptographic libraries. The choice of a correct cipher suite and its respectable parameters has a considerable impact on the overall security of a system and the time required to break into that system. The thesis is focused mainly on the OpenSSL library which has two different interfaces for cipher suites configuration.

The result of this thesis is a proposal and an implementation of a unified OpenSSL interface which allows all cipher suites to be enabled using the original *cipher_list* interface. Moreover, an option to limit validity of a cipher alias to selected versions of the protocol and to move a cipher alias or cipher suite to the beginning of the list was added. Furthermore, the server-side cipher string was extended to support equal-preference groups and to allow preference flags to be entered. The configuration of the preference flags is similar to the *-prioritize_chacha* option. This allows for the correct cipher suite to be selected based on the preference settings of both the server and client. All modifications were designed to be backward compatible.

**Keywords** cipher suite selection, cipherlist, OpenSSL, TLS 1.3

# Obsah

# Seznam obrázků

# Seznam tabulek

# Úvod

Počátky kryptografie sahají až do doby před naším letopočtem, kdy se používala jednoduchá substituční (Caesarova) šifra. Od té doby se mnohé změnilo, ale potřeba utajit obsah zpráv zůstává. S rostoucím výpočetním výkonem počítačů je potřeba neustále vylepšovat šifry, aby nebylo možné je snadno prolomit.

Při výběru šifer nemůžeme brát v potaz pouze jejich bezpečnost v aktuální době, ale musíme se řídit i tím, jak dlouho chceme dané informace zachovat utajené. Může se stát, že si útočník naši komunikaci odchytí a uloží. Tyto zprávy pak může prolomit několik let poté, až bude mít k dispozici dostatečný výpočetní výkon. Po šifrách tedy na jednu stranu požadujeme, aby nebylo možné je snadno prolomit, na stranu druhou zase chceme, aby šifrování i dešifrování bylo co nejrychlejší a výpočetně nenáročné. Tyto protichůdné požadavky vytváří potíže například u mobilních zařízení, která jsou limitovaná svým výpočetním výkonem a spotřebou. Servery zase potřebují být schopny odbavit velký počet klientů za co nejkratší čas. Každé zařízení tedy může preferovat jiné šifry a výběr bude záležet na nastavení obou komunikujících stran.

Z výše uvedených důvodů může mít jedna nebo druhá strana motivaci pro použití rychlejší šifry s kratším klíčem, zatímco druhá může preferovat pomalejší šifru používající delší klíč s očekáváním vyšší bezpečnosti. Přínosem optimální volby může být také úspora energie u mobilních zařízení a tedy jejich delší výdrž. Díky vhodné volbě lze také ušetřit finanční prostředky při nákupu nového serveru. Pokud budou upřednostněny šifry mající hardwarovou podporu na serveru, zvládne server s takovou konfigurací obsloužit více požadavků. Vylepšení specifikace šifer ocení zejména správci serverů, protože server podle vlastního seznamu šifrových sad a seznamu zaslaného klientem vybere výslednou šifrovou sadu, pomocí které bude šifrována komunikace. Klientské aplikace totiž často neposkytují žádné možnosti konfigurace.

Nastavení je důležité navrhnout tak, aby již ve výchozím stavu bylo bezpečné a snadno pochopitelné. Pokud bude nastavení příliš složité, správci serverů se v něm nemusí vyznat, což může vést až k nevědomému povolení slabých protokolů nebo šifer a tedy k prolomení celé komunikace.

Tato práce se zabývá analýzou zadávání šifer v SSL/TLS knihovnách, zejména v knihovně OpenSSL. Ukážeme, že hlavním nedostatkem této knihovny je používání dvou odlišných rozhraní pro zadávání šifrových sad, kdy každé z nich nastavuje šifrové sady jiné verze protokolu. Cílem práce je tedy návrh a následná implementace jednotného rozhraní pro zadávání šifrových sad v knihovně OpenSSL. Dále by toto řešení mělo umožnit větší variabilitu správcům serverů tak, aby při výběru šifer nezáleželo pouze na preferenci serveru nebo pouze na preferenci klienta, ale aby reflektovalo preference obou stran. Aby toto řešení mohlo být snadno použito v již existujících projektech, je důležité, aby nijak neovlivnilo stávající API/ABI rozhraní a bylo zpětně kompatibilní.

## Struktura práce

V kapitole 1 je čtenář seznámen se základními pojmy týkajících se kryptografie. Dále jsou popsány postupy při navazování spojení v jednotlivých verzích protokolů a rozdíly mezi nimi.

Kapitola 2 popisuje existující SSL/TLS knihovny a vysvětluje, které protokoly podporují a jak řeší problematiku zadávání šifer.

Kapitola 3 shrnuje problémy z předchozí kapitoly a navrhuje řešení těchto nedostatků.

Kapitola 4 na základě navrženého řešení popisuje jeho implementaci, implementační detaily použité v knihovně a související problémy.

Poslední kapitola 5 se zabývá analýzou implementovaných funkcionalit, jejich vlivem na rychlost a možnostmi integrace těchto změn do existujících projektů, které knihovnu OpenSSL využívají.

Kapitola 1

# Popis kryptografických protokolů

Tato kapitola seznamuje čtenáře se základními pojmy týkající se kryptografických knihoven a popisuje navázání spojení tak, jak je definováno v internetových standardech. Dále se věnuje rozdílům mezi jednotlivými verzemi TLS protokolů.

## 1.1 TLS protokol

Protokol Transport Layer Security (TLS), jak je definován v RFC 8446[1], 5705[2] a 6066[3], umožňuje klientským i serverovým aplikacím komunikovat prostřednictvím internetu a zajišťuje jim důvěrnost, integritu a autentizaci zpráv. Jeho aktuální verze nese označení TLS 1.3 a je definovaná v RFC 8446 [1].

Samotný protokol se skládá ze dvou vrstev. První vrstva nese označení *TLS record protocol* a pomocí dohodnutých parametrů z druhé vrstvy zabezpečuje komunikaci obou stran. Úkolem druhé vrstvy — *handshake protocol* — je ověřit obě strany a vyjednat parametry kryptografických funkcí dle preferencí a možností obou stran.

[1]RFC 8446: The Transport Layer Security (TLS) Protocol Version 1.3
[2]RFC 5705: Keying Material Exporters for Transport Layer Security (TLS)
[3]RFC 6066: Transport Layer Security (TLS) Extensions: Extension Definitions

### 1.1.1 Záznamový protokol TLS

Jedná se o nízkoúrovňový protokol označovaný jako *TLS record protocol*, který u zpráv určených k odeslání zajišťuje:

- rozložení do jednotlivých rámců (fragmentaci),
- (volitelně) kompresi rámců,
- (volitelně) autentizaci rámců pomocí MAC,
- (volitelně) šifrování rámců pomocí proudové nebo blokové šifry,
- posílání rámců.

Druhá strana přijme odeslané rámce a záznamový protokol zajistí inverzní proces k výše uvedené sekvenci kroků tak, aby z nich složil původní zprávu.

Tento protokol je využíván čtyřmi základními protokoly vyšších vrstev:

- **Handshake protocol** — Zodpovídá za navázání komunikace a dohodnutí šifrovacích parametrů. Více v následující sekci 1.2.
- **ChangeCipherSpec protocol** — Slouží k signalizaci změny šifrovacích parametrů. Obsahuje pouze jedinou zprávu, která je zašifrovaná a zkomprimovaná aktuálním nastavením. Zpráva je poslána jak klientem, tak serverem, a tím si obě strany vzájemně potvrdí, že další zprávy budou šifrovány nově dohodnutým způsobem. Tato zpráva je součástí navazování spojení a následuje po vzájemné dohodě na šifrovacích parametrech.
- **Alert protocol** — Zajišťuje přenos chybových zpráv. Jsou definované v RFC 5246 [2] a je jich celkem 25 druhů. Každá zpráva se skládá ze závažnosti chyby (varování nebo kritická chyba) a jejího popisu.
  - Kritické chyby (označené jako `fatal`) způsobí okamžité ukončení spojení, identifikátor relace musí být zneplatněn. To proto, aby relace již nemohla být obnovena.
  - Varování (označené jako `warning`) má pouze informativní charakter, po přijetí takové zprávy spojení může pokračovat. Záleží jen na příjemci, jak s tímto varováním naloží — jestli jej bude ignorovat nebo odpoví kritickou chybou, čímž ukončí spojení. Příkladem může být nedůvěryhodný nebo expirovaný certifikát.

  Samozřejmostí je šifrování a komprese i těchto zpráv.
- **Application protocol** — Přenáší zprávy, která obsahují samotná aplikační data. Zprávy jsou také šifrovány a komprimovány dříve dohodnutými šifrovacími parametry.

### 1.1.2 Autentizované šifrování

Autentizované šifrování je taková forma šifrování, která kromě důvěrnosti zajišťuje navíc integritu a pravost dat. Autentizované šifrování s asociovanými daty — z anglického výrazu *Authenticated Encryption with Associated Data*, zkráceně `AEAD` — přidává navíc schopnost ověřit integritu a pravost asociovaných dat, která ale nejsou zašifrovaná. Příkladem může být zpráva s hlavičkou, kde obě části budou ověřeny pomocí autentizačního kódu, ale pouze zpráva bude šifrovaná.

Mnoho kryptografických aplikací vyžaduje důvěrnost a autentizaci zpráv. Toho je obvykle docíleno šifrováním a následným ověřením pomocí MAC[4], kde každá operace používá jiný klíč. Používáním dvou operací se také zvyšuje pravděpodobnost špatného použití a tím může být ohrožena bezpečnost celé aplikace. Proto byly definovány algoritmy používající AEAD, které jsou implementovány buď jako samostatné algoritmy nebo jako operační módy blokových šifer.

Samotný algoritmus AEAD popisuje obrázek 1.1 a funguje tak, že na vstupu dostane:

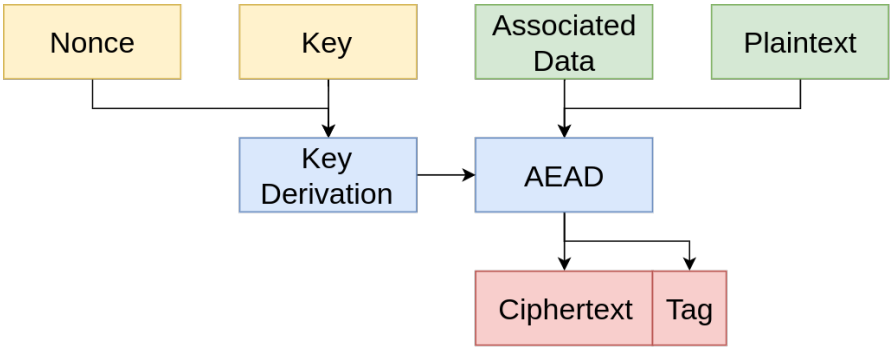
- asociovaná data, která mají být pouze ověřena, nikoliv zašifrována,
- otevřený text, který má být ověřen i zašifrován,
- hodnotu nonce — náhodné nebo pseudonáhodné číslo, které může být použito pouze jednou,
- klíč použitý pro šifrování.

Ze zadaného klíče a hodnoty nonce pomocí specializované funkce (například HKDF — podsekce 1.1.4) je odvozen klíč, který bude sloužit pro šifrování a ověření. Výstupem algoritmu pak je zašifrovaný text (ciphertext) a otisk (tag) pevné délky, který ověřuje jak asociovaná data, tak šifrový text.

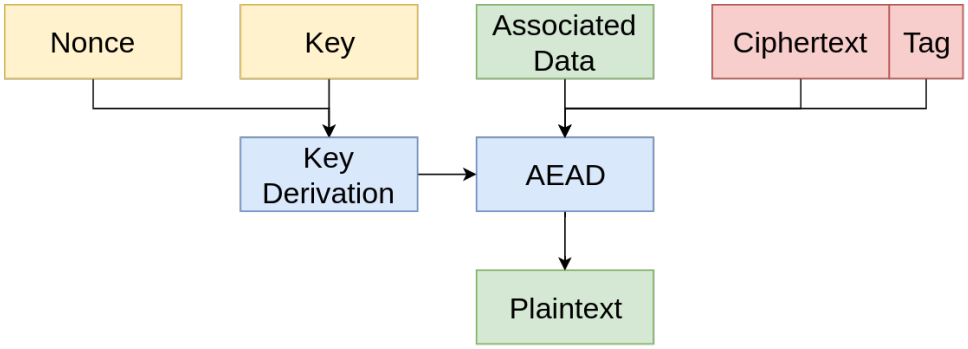
Obrázek 1.2 znázorňuje dešifrování dat a jejich ověření. Na vstupu algoritmu jsou: asociovaná data, šifrový text s ověřovacím otiskem, nonce a klíč. Z klíče a hodnoty nonce se opět odvodí klíče pro dešifrování a ověření. Pokud asociovaná data i šifrový text projdou ověřením, na výstupu algoritmu je původní otevřený text, jinak algoritmus nevrátí žádnou hodnotu.

Příkladem použití v protokolech TLS jsou šifry využívající proudovou šifru ChaCha. Dále se autentizované šifrování s asociovanými daty používá u blokových šifer AES a AIRA s operačním módem *Galois/Counter Mode* (GCM) nebo *Counter with CBC-MAC* (CCM).

[4]Message Authentication Code — Funkce zajišťující integritu a autentizaci zpráv.




Obrázek 1.1: Šifrování v AEAD režimech




Obrázek 1.2: Dešifrování v AEAD režimech

### 1.1.3 Autentizační kód zprávy

Při přenášení zprávy přes nespolehlivé médium představují autentizační kódy způsob, jak zkontrolovat integritu zpráv. Tento způsob ale nezaručuje ochranu před změnou zprávy a jejího autentizačního kódu útočníkem. To řeší až funkce `HMAC` — Keyed-Hashing for Message Authentication — která používá k samotnému výpočtu autentizačního kódu i tajný klíč. Podle [3] je funkce `HMAC` definovaná takto:

$$\mathrm{HMAC}_K(m) = \mathrm{H}\left((K \oplus \mathrm{opad}) || \mathrm{H}\left((K \oplus \mathrm{ipad}) || m\right)\right),$$

kde opad a ipad představují konstantní výplňové řetězce a H je iterativní rozptylovací funkce. Proměnná $K$ (klíč) a $m$ (zpráva), ke které se počítá ověřovací kód, $\oplus$ zastupuje operaci XOR a $||$ znamená zřetězení.

### 1.1.4 Funkce pro odvození klíče

Funkce pro odvození klíče založená na HMAC, známá pod zkratkou `HKDF` — z anglického výrazu *HMAC-based Key Derivation Function*, byla definována v [4] a je navržena jako základní komponenta pro kryptografické systémy, které potřebují z prvotního klíče odvodit jeden nebo více kryptograficky silných

klíčů. Funkce se skládá ze dvou modulů:

- první modul označovaný jako *extract* má za úkol koncentrovat entropii vstupního klíče do kratšího, ale kryptograficky silného pseudonáhodného klíče. Používá se zejména proto, že v mnoha aplikacích není vstupní klíč dostatečně rovnoměrně rozložený a útočník o něm může mít částečné znalosti. Naopak tam, kde je vstupní klíč dostatečně náhodný, lze tento krok vynechat.
- Druhý modul se jmenuje *expand* a jeho cílem je rozšířit pseudonáhodný klíč z předchozího kroku na požadovanou délku podle potřeb daného algoritmu, který tento klíč pak bude využívat.

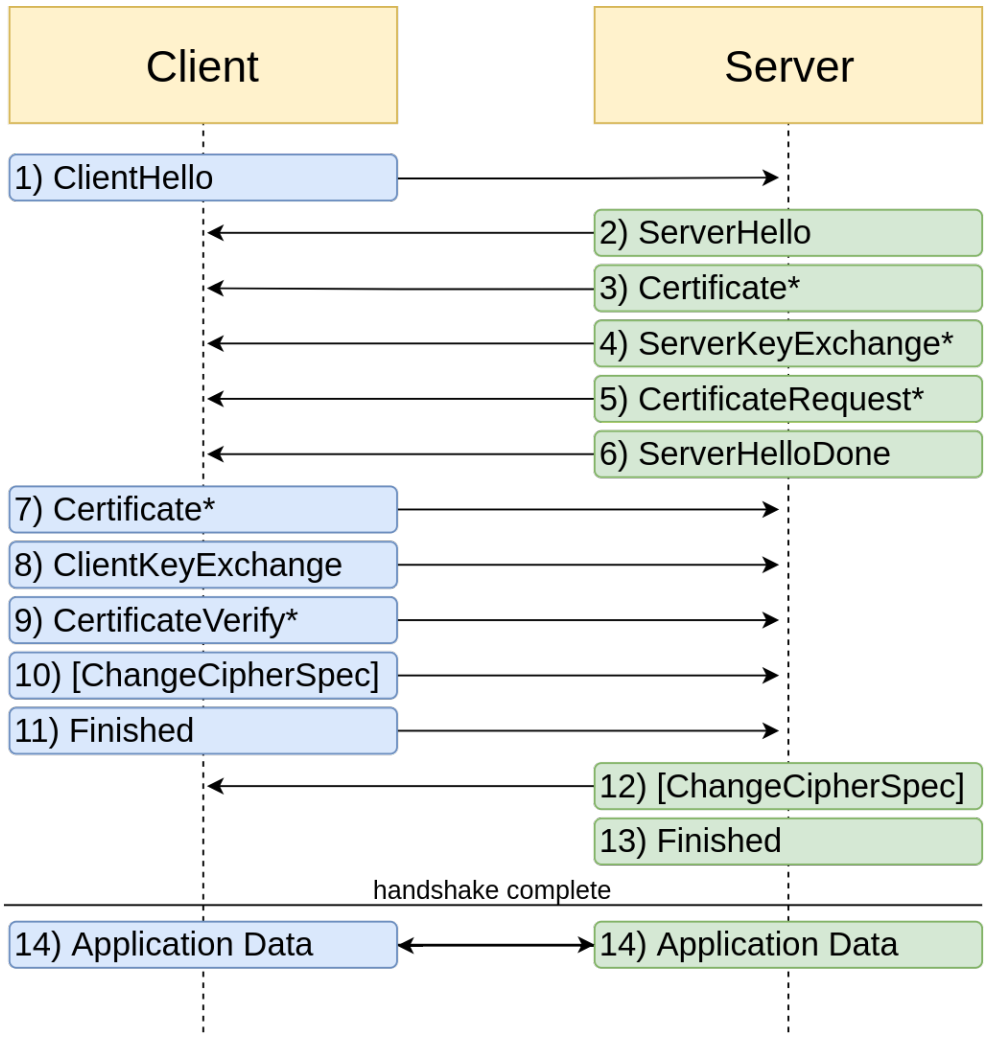
## 1.2 Navázání spojení

Za navázání spojení zodpovídá *handshake protocol*, který mezi klientem a serverem zajišťuje výměnu informací potřebných k navázání zabezpečeného spojení. S novou verzí TLS 1.3 přišlo i mnoho změn, proto je níže popsáno navázání spojení s dřívější verzí protokolu a následně s aktuální verzí.

### 1.2.1 Spojení v TLS 1.2

Postup navázání spojení ve verzi TLS 1.2 je definován v [2]. V případě, že klient se serverem ještě nekomunikoval, musí proběhnout úplné navázání spojení, tzv. *TLS full handshake*, který je znázorněn na obrázku 1.3 a skládá se z následujících kroků:

1. klient zahajuje komunikaci a pošle servu zprávu `Client Hello`, ve které bude:
    - verze protokolu, například 1.0, 1.1 nebo 1.2.
    - Náhodná data: 32 bajtů náhodných dat, která budou použita později. První 4 bajty představují aktuální čas ve standardním UNIX formátu.
    - Identifikátor relace: Volitelná položka, používá se pro obnovení relace. Pro navázání nového spojení není potřeba.
    - Seznam šifer: Seznam podporovaných šifrových sad definujících algoritmus pro výměnu klíče, šifrování pomocí tohoto klíče a algoritmus pro ověření zpráv. Seznam je seřazený podle preference tak, že šifra s nejvyšší preferencí je na prvním místě.
    - Seznam kompresních metod: Seznam podporovaných kompresních metod, které budou použity pro kompresi dat před šifrováním.




Obrázek 1.3: Úplné navázání spojení v TLS 1.2

`*` znázorňuje zprávy, které nemusí být vždy poslány, jsou volitelné nebo záleží na konkrétní situaci.

`[]` znázorňují zprávy protokolu `ChangeCipherSpec`.

- Seznam rozšíření: Seznam klientem podporovaných rozšíření, které může server použít. Mezi používaná rozšíření patří například *Server Name Indication*, díky kterému se server spravující více doménových jmen, dozví, kterou stránku chce klient navštívit a následně pošle odpovídající certifikát.

Tuto zprávu může klient poslat i v případě, že se pokouší o obnovu předchozího spojení nebo jako odpověď na žádost serveru `Hello Request`.

2. Server odpoví zprávou `Server Hello`, jejíž struktura je podobná jako `Client Hello`, pouze s tím rozdílem, že místo seznamů posílá již konkrétní verzi protokolu, šifrovou sadu a kompresní metodu. Volitelně, v závislosti na situaci, může server poslat i následující typy zpráv:

3. `Server Certificate`, která obsahuje certifikát (veřejný klíč serveru včetně mezilehlých klíčů certifikačních autorit).

4. `Server Key Exchange` — Tato zpráva se posílá pouze pro některé algoritmy výměny klíče a obsahuje veřejný klíč — například pro Diffie-Hellman výměnu klíče — pomocí kterého může klient dokončit výměnu sdíleného klíče, tzv. *premaster secret*.

5. `Certificate Request`, kterou si vyžádá zaslání certifikátu klienta.

6. Následně server pošle zprávu `Server Hello Done`, čímž signalizuje, že svoji část navázání spojení dokončil.

7. Pokud server žádal klienta o certifikát, klient mu na žádost odpoví zprávou `Client Certificate`.

8. Dále klient posílá zprávu `Client Key Exchange`, pomocí které se dokončí výměna sdíleného klíče *premaster secret*.

9. V případě, že v předchozích zprávách byl serverem poslán certifikát, klient ověří, že server má příslušný privátní klíč odesláním zprávy `Certificate Verify`.

10. Nyní klient pošle zprávu `Change Cipher Spec`, kterou signalizuje, že spočítal sdílený klíč, potvrdí použití dohodnutých způsobů a parametrů šifrování a všechny následující zprávy budou tímto způsobem šifrovány.

11. Pro ověření, že navázání spojení bylo úspěšné a nebylo nikým narušeno, klient spočítá hash ze všech přijatých zpráv. Následně je zašifruje svým klíčem a ve zprávě `Finished` je odešle serveru, který je zkontroluje.

12. Server také potvrdí používání dohodnutých způsobů šifrování zprávou `Change Cipher Spec`.

13. Poté pošle zprávou `Finished`, aby i klient mohl ověřit spojení.

14. Dále již následují zprávy typu `Application Data`, která obsahují samotná zašifrovaná data odesílaná klientem nebo serverem.

15. Po přenesení potřebných dat může klient nebo server ukončit spojení odesláním zprávy `Close Notify`. Druhá strana pak musí odpovědět stejnou zprávou. Jakákoliv data po těchto zprávách jsou ignorována.

Kromě tohoto úplného navázaní spojení je možná i jeho zkrácená verze. Podmínkou je, že klient se serverem již dříve komunikoval a identifikátor relace je ještě platný.

### 1.2.2 Spojení v TLS 1.3

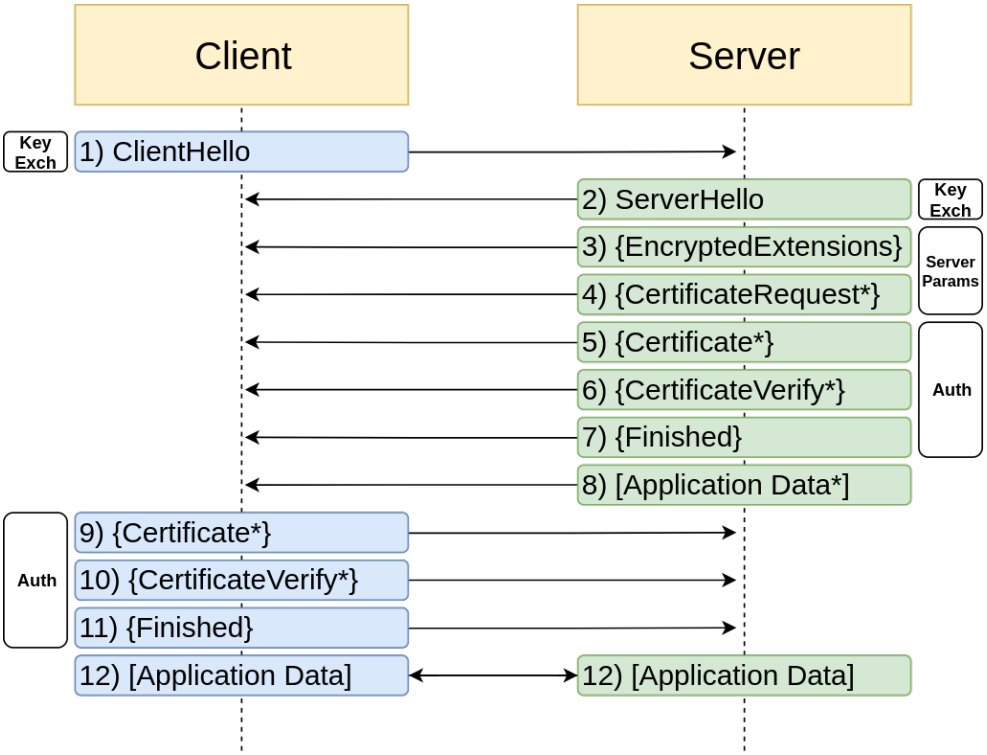
Verze TLS 1.3 přichází s šifrováním již během navazování spojení a celý proces zrychluje. Postup navázání spojení podle RFC [1] je znázorněn na obrázku 1.4 a jeho kroky jsou následující:

1. stejně jako v předchozí verzi protokolu i zde zahajuje klient komunikaci odesláním zprávy `Client Hello`, případně ji posílá znovu na žádost `HelloRetryRequest`.

   Obsah této zprávy je stejný jako v předchozí verzi, ale význam jednotlivých položek se liší:

   - verze protokolu — Položka, která se v dřívějších verzích používala k výběru verze. Některá zařízení, která jsou mezi komunikujícími stranami (například firewall nebo bezpečnostní systémy typu IDS, IPS) měla s novou hodnotou problém. Proto kvůli zpětné kompatibilitě je položka nastavena na hodnotu `0x0303`, která reprezentuje verzi TLS 1.2 a pro signalizaci verze protokolu je použito rozšíření `supported_versions`.
   - Náhodná data — 32 bajtů náhodných dat, jsou popsána podrobněji v následující zprávě.
   - Identifikátor relace — Tato položka již není potřeba, ale některá zařízení monitorující spojení mají s novou verzí protokolu problémy. Proto je tato položka vyplněna nenulovou hodnotou, aby spojení vypadalo jako v předchozích verzích.

     Klient si tedy identifikátor relace zvolí sám, pokud zahajuje nové spojení. Pokud jenom obnovuje spojení, měl by použít předchozí identifikátor. Hodnota nemusí být náhodná, ale měla by být nepredikovatelná.
   - Seznam šifer — Stejné jako v předchozích verzích.




Obrázek 1.4: Úplné navázání spojení v TLS 1.3

∗ znázorňuje zprávy, které nemusí být vždy poslány, jsou volitelné nebo záleží na konkrétní situaci.

`[]` znázorňují zprávy, které jsou chráněny klíčem odvozeným z `application_traffic_secret`.

`{}` znázorňují zprávy, které jsou chráněny klíčem odvozeným z `handshake_traffic_secret`.

- Seznam kompresních metod — Již nepoužívaná položka, musí být nastavena na nulu, což odpovídá kompresní metodě `NULL`. Pokud server obdrží jinou hodnotu, musí ukončit spojení chybovou zprávou `illegal_parameter`.
- Seznam rozšíření — Shodné s předchozí verzí, pouze s tím rozdílem, že některá rozšíření jsou povinná (například `supported_versions`), protože se část funkcionality přesunula právě do těchto rozšíření.

2. Server odpovídá pomocí `Server Hello`, pokud je schopný navázat spojení se zaslanými parametry.

   Zpráva dále obsahuje:

   - verzi protokolu — Nastaveno na hodnotu verze TLS 1.2.
   - Náhodná data — 32 bajtů pocházejících z kryptograficky bezpečného pseudonáhodného generátoru.

     Pokud se navazuje spojení s klientem, který podporuje TLS 1.2 nebo starší, musí server nastavit posledních 8 bajtů na konkrétní předdefinované hodnoty. Klienti, kteří podporují TLS 1.3 a obdrží od serveru v této zprávě `ServerHello` signalizující starší protokol, musí zkontrolovat posledních 8 bajtů těchto náhodných dat. Pokud se těchto posledních 8 bajtů bude shodovat s předdefinovanými hodnotami, znamená to, že server i klient podporují protokol verze TLS 1.3, ale z nějakého důvodu nejsou schopni navázat spojení v této verzi. Klient v takovém případě musí ukončit spojení zprávou `illegal_parameter`.
   - Identifikátor relace — Zkopírovaná hodnota z předchozí zprávy `Client Hello`. Tato hodnota je zkopírována i v případě, že odpovídá dřívější relaci, kterou se server rozhodl neobnovovat. Klient musí zkontrolovat, že tato hodnota odpovídá té, kterou on odeslal, jinak ukončuje spojení s chybou `illegal_parameter`.
   - Šifrová sada, kterou vybral server na základě svých povolených šifrových sad a těch zaslaných klientem.
   - Kompresní metoda — Nastaveno na hodnotu `NULL`.
   - Seznam rozšíření — pouze ta, která jsou potřeba pro výběr verze protokolu a navázání spojení. Povinným rozšířením, které zde musí být uvedeno, je `supported_versions`.

3. První šifrovanou zprávou poslanou serverem je `Encrypted Extensions`. Musí bezprostředně následovat za zprávou `ServerHello`. Je šifrovaná pomocí klíče odvozeného z `server_handshake_traffic_secret` a může obsahovat rozšíření, která nemusí být potřebná pro navázání spojení, ale

ani nejsou spojena s jednotlivými certifikáty. Klient pak musí zkontrolovat, že se zde nenachází žádné zakázané rozšíření a pokud ano, tak musí navazování spojení ukončit chybovou zprávou `illegal_parameter`.

4. Pokud se server autentizuje pomocí certifikátu, může po klientovi pomocí zprávy `Certificate Request` žádat také certifikát. Tato zpráva je nepovinná, pokud je ale poslána, musí následovat až za zprávou `Encrypted Extensions`.

5. Dále server může poslat zprávu `Certificate`, která je nepovinná. Jak název napovídá, v této zprávě server posílá svůj certifikát včetně jeho případných rozšíření. Zpráva se posílá pouze v případě, že se server ověřuje pomocí certifikátu.

6. Zpráva `Certificate Verify` poslaná serverem obsahuje podpis zpráv z procesu navazování spojení. Tento podpis je uskutečněn pomocí soukromého klíče, který odpovídá veřejnému klíči poslanému v předchozí zprávě. Tato zpráva je vynechána v případě, že se druhá strana neověřuje pomocí certifikátu.

7. Celý proces navázání spojení je ověřen zprávou `Finished`, která obsahuje otisk ze všech serverem přijatých zpráv. Oproti předchozí verzi, kde se používala pouze rozptylovací funkce, je zde použita funkce MAC, jejíž výstup závisí kromě samotných dat i na klíči. Kromě integrity tak zajišťuje i autentizaci zprávy.

8. Server teď může poslat zprávy typu `Application Data`. Zatím ale nemá jistotu, že je klient schopný komunikovat, a ani není ověřena klientova identita. Zprávy tohoto typu jsou šifrovány klíčem odvozeným z `application_traffic_secret`.

9. Pokud v předchozích zprávách server žádal klientův certifikát, nyní mu jej klient ve zprávě `Certificate` pošle. Jinak se tato zpráva neposílá.

10. Význam zprávy `Certificate Verify` poslané klientem je stejný, jako když ji posílá server. Je opět nepovinná a ověřuje všechny předchozí zprávy. Posílá se pouze v případě, že server žádal klientův certifikát pomocí zprávy `Certificate Request`.

11. Nyní stejným způsobem jako server i klient spočítá ověřovací zprávu `Finished`. Dokud tak klient neučiní, nesmí posílat aplikační data.

12. Od této chvíle je spojení navázáno, obě strany teď odvodí klíč pro záznamovou vrstvu a poté spolu můžou zabezpečeně komunikovat — posílat zprávy typ `Application Data`.

Klíče `handshake_traffic_secret_0` a `application_traffic_secret_0` jsou odvozeny ze zpráv poslaných během navazování spojení a samotných šifrových klíčů pomocí funkce HKDF popsané v podsekci 1.1.4; stejnou funkcí jsou z nich pak odvozeny samotné klíče pro šifrování zpráv. Pokud je potřeba změnit klíč pro šifrování zpráv, je z klíče `application_traffic_secret_N` odvozen klíč `application_traffic_secret_N+1` a z něj je následně odvozen klíč pro šifrování. Původní klíč s příponou `_N` by měl být smazán. Odvozování dalších klíčů pro `handshake_traffic_secret` funguje obdobně.

V předchozí verzích protokolu TLS byly při navazování spojení šifrovány jenom poslední zprávy typu `Finished`. Nyní jsou šifrované všechny zprávy již od `ServerHello`. Zrychlení navázání spojení pak spočívá ve snížení vlivu latence díky optimálnějšímu uspořádání zpráv. Je omezen počet stavů, kdy druhá strana čeká na přijetí zpráv od první strany, aby na základě těchto zpráv mohla první straně odpovídat. Samotná aplikační data je možné poslat již po zprávě typu `ServerHello` a není potřeba čekat, než ji klient přijme.

## 1.3 Šifrové sady

Pojmem *cipher suite* se v protokolu TLS označuje soubor algoritmů, které se použijí pro navázání zabezpečené komunikace. V případě jiných protokolů může mít tento pojem odlišný význam.

Pro verze TLS 1.2 a starší vypadá cipher suite například takto:

`TLS_DHE_RSA_WITH_3DES_CBC_SHA`.

Konkrétní konvence není určena, takže formát se může mírně lišit v každé knihovně, ale princip zůstává stejný. V knihovně OpenSSL a příslušných RFC, kde se dané šifry popisují, se název skládá z následujících částí:

- `TLS` — Název protokolu, pro který je cipher suite určena.
- `DHE_RSA` — Algoritmus použitý pro výměnu klíče a autentizaci klienta se serverem při navazování spojení. V tomto případě je použit `DHE` pro výměnu klíče a `RSA` pro autentizaci. Je-li uveden pouze jeden algoritmus, použije se pro obojí.
- `WITH_3DES_CBC` — Proudová nebo bloková šifra, která se použije pro samotné šifrování zpráv. Šifra je uvedena včetně svého operačního módu (v tomto případě `CBC`). Může-li používat různé délky klíčů, je uvedena i délka klíče — `AES_256_CBC`.
- `SHA` — Algoritmus pro ověření zpráv (MAC) — v tomto případě rozptylovací funkce SHA.

Pro TLS verze 1.3 stanovuje RFC 8446 [1] odlišný formát. Rozdíl spočívá v tom, že výměna klíče byla oddělena od samotného šifrování. K tomu se nepoužívají zastaralé šifry, ale pouze ty, které podporují ověřené šifrování s přidruženými daty (AEAD), které je podrobněji popsáno v podsekci 1.1.2. Samotné jméno šifrové sady (cipher suite) vypadá následovně:

`TLS_AES_128_GCM_SHA256`

- `TLS` — Název protokolu, pro který je cipher suite určena. Často ale knihovny zaměňují tuto část názvu za `TLS13`.
- `AES_128_GCM` — Uprostřed se nachází algoritmus pro ověřené šifrování s přidruženými daty (AEAD).
- `SHA256` — Název ukončuje funkce pro jednoduché odvození klíče (HKDF), popsaná v podsekci 1.1.4.

Toto RFC také pro verzi protokolu TLS 1.3 specifikuje nové šifry:

- `TLS_AES_128_GCM_SHA256`,
- `TLS_AES_256_GCM_SHA384`,
- `TLS_CHACHA20_POLY1305_SHA256`,
- `TLS_AES_128_CCM_SHA256`,
- `TLS_AES_128_CCM_8_SHA256`

a z důvodu kompatibility požaduje, aby alespoň první tři šifry byly implementovány.

### 1.3.1 Porovnání síly šifer

Budeme-li mít dvě stejné šifry, které se liší pouze délkou klíče, je snadné porovnat, která je bezpečnější. Jak ale porovnat mezi sebou různé šifry? Tím se zabývá publikace [5] od Národního institutu standardů a technologií (NIST).

Za předpokladu, že budou klíče šifer vygenerovány kryptograficky bezpečným generátorem, můžeme pak dva šifrovací algoritmy pro dané velikosti klíčů považovat za srovnatelné, pokud k jejich prolomení nebo získání soukromých klíčů bude potřeba podobné množství práce na stejném zařízení. Samotná bezpečnostní síla algoritmu je pak určena jako množství práce potřebné k vyzkoušení všech možných klíčů algoritmu o dané velikosti klíče, který nemá žádné zranitelnosti vedoucí ke zrychlení nebo zkrácení při zkoušení klíčů. Algoritmy, jejichž bezpečnostní síla je menší než 112 bitů, by se podle NIST neměly používat.

Kapitola 2

# Knihovny pro zabezpečení komunikace

V této kapitole jsou popsány jednotlivé kryptografické knihovny pro zabezpečení komunikace. Nejprve je každá knihovna stručně popsána a následně je detailně vysvětleno, jak řeší problematiku se specifikací šifer pro různé verze protokolů. Protože u některých knihoven není dokumentace dostatečně podrobná, byla v těchto případech s knihovnami navázána spojení s různým nastavením šifer pro otestování deklarovaného/odhadovaného chování. V tabulce 2.1 se pak nachází přehled knihoven, podporovaných verzí, jejich certifikací a licencí.

Tabulka 2.1: Srovnání jednotlivých SSL/TLS knihoven

| **Knihovna** | **TLS 1.3** | **Certifikace *** | **Licence** |
|---|---|---|---|
| OpenSSL | ✓ | FIPS 140-2 | Apache 2.0, dříve OpenSSL |
| wolfSSL | ✓ | FIPS 140-2 | GPLv2, komerční |
| BoringSSL | ✓ | FIPS 140-2 | OpenSSL, ISC |
| mbed TLS | ✗ | | Apache 2.0, GPLv2 |
| TLS Toolkit | ✓ | FIPS 140-2 | GPLv2, komerční |
| NSS | ✓ | FIPS 140-2 | MPL 2.0 |
| JSSE | ✗ | | GPLv2 |
| GnuTLS | ✓ | | LGPLv2.1 |
| Bouncy Castle | ✗ | FIPS 140-2 | MIT |
| Rustls | ✓ | | Apache 2.0, MIT, ISC |

* Certifikace se ve většině případů týkají pouze kryptografických modulů, nikoliv celých knihoven.

## 2.1 OpenSSL

Projekt OpenSSL je opensource kryptografická sada nástrojů, která je napsaná převážně v jazyku C a poskytuje knihovny `libssl` a `libcrypto`. První jmenovaná zodpovídá za SSL/TLS protokol a je závislá na `libcrypto`. Druhá knihovna poskytuje základní kryptografické rutiny a může být použita přímo prostřednictvím `libssl` nebo bez ní. Poslední verzi protokolu TLS podporuje od verze `1.1.1`, která vyšla 11. září 2018.

### 2.1.1 Použití knihovny

Knihovna nabízí pro aplikace třetích stran programové rozhraní API nebo nízkoúrovňové rozhraní ABI. Pro psaní skriptů, použití z příkazové řádky nebo prosté testování funkcionalit nabízí knihovna nástroje pro příkazovou řádku.

### 2.1.2 Zadávání šifrových sad

Ke specifikaci šifrových sad slouží dvě rozhraní:

- `cipher_list` — Používá se pro zadávání šifrových sad až do verze TLS 1.2. Poskytuje mnoho operátorů a funkcí pro zadávání, řazení a výběr šifrových sad. Kromě dvojtečkou odděleného seznamu šifrových sad je možné zadávat i šifrové aliasy, které zastupují několik šifer se stejným algoritmem, například pro výměnu klíče nebo samotné šifrování. Dále je možné používat operátory pro průnik aliasů, zástupné řetězce pro všechny nebo výchozí šifrové sady (`ALL`, `DEFAULT`) a jejich doplňky (`COMPLEMENTOFALL`, `COMPLEMENTOFDEFAULT`). Nechybí ani možnost v daném šifrovém seznamu trvale zakázat, odstranit nebo přesunout na konec (pomocí operátorů `!,-,+`) vybrané šifrové sady nebo aliasy a také je seřadit (například podle délky jejich klíče) pomocí speciálních příkazů `@STRENGTH` nebo `@SECLEVEL`.

  Příkladem je následující řetězec:

  ```
  ALL:!ADH:-SSLv3+DH:@STRENGTH:+CHACHA20,
  ```

  který nejprve povolí všechny šifrové sady a trvale zakáže šifrové sady využívající k výměně klíče algoritmus Diffie-Hellman bez autentizace (anonymní). Dále odebere šifrové sady, které používají Diffie-Hellman a zároveň jsou z verze SSLv3. Nakonec zbylé šifrové sady seřadí podle síly a ty, které používají algoritmus ChaCha20 posune na konec.

- `ciphersuites` — Slouží pro šifrové sady z verze TLS 1.3 a umožňuje zadávat pouze dvojtečkou oddělený seznam šifrových sad. Nepodporuje aliasy ani řazení.

  Příkladem šifrového seznamu, který povolí dvě šifrové sady, je následující řetězec:

  ```
  TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256
  ```

### 2.1.3 Číslování verzí

Kryptografické knihovny jsou součástí kritické infrastruktury a (nejenom na nich) spočívá bezpečnost celku. Proto je nutné, aby vývojáři, výrobci i uživatelé věděli, co mohou od které verze očekávat — jestli nová verze pouze opravuje objevené chyby nebo přináší zásadní změny, které mohou mít vliv na funkčnost. Proto byla stanovena pravidla pro označování verzí, dostupné z [6], která se neustále vylepšují tak, aby lépe splňovala výše uvedené požadavky.

Od OpenSSL verze 1.0.0 se pro označení verzí používají následující pravidla:

- písmeno na konci, například 1.1.0**j**, znamená opravení bezpečnostní zranitelnosti nebo chyby, ale nepřidává žádné nové vlastnosti.

- Změna posledního čísla, například z 1.1.**0** na 1.1.**1**, znamená, že do knihovny byly přidány nové funkce, ale její binární rozhraní zůstalo nezměněno. Pokud nějaká aplikace dynamicky linkuje OpenSSL knihovnu a dojde k takové aktualizaci, aplikaci není potřeba znovu kompilovat, bude fungovat stále dál.

- První dvě číslice v označení verze jsou určeny pro aktualizace, které změní API/ABI poskytované knihovnou.

S příchodem OpenSSL verze 3.0.0 dojde k vylepšení označování verzí na formát `MAJOR.MINOR.PATCH`. Význam jednotlivých částí je následující:

- verze, která změní `PATCH`, bude obsahovat pouze opravy chyb.

- Zvýšení `MINOR` verze znamená aktualizaci, která neporuší kompatibilitu API/ABI.

- Změna v `MAJOR` verzi je taková, která poruší kompatibilitu API/ABI. Tím, je zaručeno, že v rámci každé `MAJOR` verze zůstane API/ABI stejné.

Tyto změny ale nebudou mít vliv na politiku označování aktuálních verzí.

## 2.2 wolfSSL

Alternativou k OpenSSL je knihovna wolfSSL, využívající kryptografickou knihovnu wolfCrypt. Také je napsaná v jazyku C a díky své rychlosti, malé velikosti a podporovaným funkcím je vhodná pro vestavěná zařízení. Svoje uplatnění nachází ale i ve standardních operačních prostředích. Podle [7] podporuje progresivní šifry jako ChaCha20, Curve25519, NTRU, Blake2b a protokol TLS až do verze 1.3, stejně jako OpenSSL, ale ve srovnání s ní je až 20× menší. Stejný zdroj také uvádí, že knihovna wolfSSL dosahuje výrazně lepších výkonů. Aby se mohla tato knihovna prosadit i u již existujících projektů, které používaly OpenSSL, poskytuje *OpenSSL kompatibilní vrstvu*, díky které je možné přejít na wolfSSL bez nutnosti přepisovat celý kód.

Tato knihovna podporuje protokol TLS 1.3 již od května roku 2017, kdy se jednalo pouze o návrh. Nyní již podporuje schválenou verzi RFC a od března roku 2019 je její knihovna wolfCrypt ověřena standardem FIPS 140-2.

Pro povolení TLS 1.3 je potřeba před zkompilováním knihovny přidat konfiguračnímu skriptu volbu `--enable-tls13`. Ve výchozím nastavení se knihovna snaží vybrat nejbezpečnější/kryptograficky nejsilnější šifru, kterou podporují obě strany. Toto chování je ale možné změnit. Nejprve je nutné vytvořit kontext pomocí funkce:

```
WOLFSSL_CTX * wolfSSL_CTX_new ( WOLFSSL_METHOD *)
```

a následně je možné funkcí:

```
int wolfSSL_CTX_set_cipher_list ( WOLFSSL_CTX *, const char *)
```

specifikovat seznam šifer, které se mají použít. Prvním argumentem funkce je již dříve vytvořený kontext a jako druhý argument se zadá dvojtečkou oddělený seznam požadovaných šifer. Preference je dána jejich pořadím při zadávání.

V dokumentaci jsou šifry rozděleny do následujících 3 skupin podle jejich bezpečnosti. Bity bezpečnosti jsou určeny podle algoritmu doporučeného NIST uvedeném v podsekci 1.3.1:

- LOW — méně než 128 bitů,
- MEDIUM — 128 bitů,
- HIGH — více než 128 bitů.

Tyto skupiny ale není možné použít při specifikaci šifer, je nutné jednotlivé šifrové sady explicitně vyjmenovat.

Stejně jako při kompilování knihovny, je i při spuštění její ukázkové implementace serveru nebo klienta ve výchozím nastavení verze TLS 1.3 vypnuté. Pro povolení této verze je potřeba ji explicitně vybrat pomocí přepínače `-v 4`. Tím ale dojde k povolení pouze jedné verze, takže není možné používat zároveň obě verze.

## 2.3 BoringSSL

Knihovna BoringSSL vychází z OpenSSL, ale je navržena tak, aby splňovala potřeby vývojářů Google. S tím také souvisí to, že knihovna nezaručuje stálost svého API/ABI a není doporučeno na ni spoléhat v aplikacích třetích stran. Na tento fakt ale společnost Google důrazně upozorňuje na stránkách projektu [8]. Knihovna je napsaná v jazyku C++ (dříve v jazyku C) a její nynější využití spočívá zejména v implementaci SSL knihovny pro prohlížeč Google Chrome a operační systém Android. Knihovna sice není dobře zdokumentovaná, ale protokol TLS ve verzi 1.3 podporuje. Ve výchozím nastavení TLS 1.3 není povoleno a jako výchozí je nastavena verze 1.2.

Díky commitu[5] umožňuje knihovna vytvářet skupiny šifer, které budou mít stejnou preferenci. Pokud server bude podporovat dvě šifry se stejnou preferencí, umožní tím klientovi vybrat si, kterou šifru zvolit. To má smysl zejména v případech, kdy klient má hardwarovou podporu pro některou z šifer.

Tyto skupiny o stejné preferenci se zadávají do hranatých závorek a oddělují se pomocí znaku `|` (roura). Interně pak je příslušnost jednotlivých šifrových sad do skupin uložena v poli pomocí příznaků. Příznaky jsou nastaveny pro každou šifru kromě poslední ve skupině. Tím je jednoznačně identifikován začátek i konec skupiny a v případě, že skupiny nejsou použity, je pole prázdné.

Budeme-li uvažovat, že byl zadán následující řetězec, tak jeho pole příznaků bude vypadat takto:

```
[ A | B | C ] : D : [ E | F ]
  1   1   0     0     1   0
```

Pro možnost předávání tohoto pole příznaků, byl v souvislosti s equal-preference groups v šifrovém kontextu (`SSL_CTX`) nahrazen seznam šifer strukturou, která v sobě obsahuje tento seznam a k němu i požadované pole příznaků použité pouze pro skupiny stejné preference.

Všechny šifrové sady jsou definovány v jednom poli bez rozlišení verze, takže se zadávají pomocí jednoho rozhraní. Podpora šifrových aliasů vychází z OpenSSL, ale je jich zde méně, stejně jako šifrových sad. Dalším rozdílem oproti OpenSSL je, že u jednotlivých šifrových sad nejsou uvedeny verze protokolu, pro které platí.

## 2.4 mbed TLS

Mbed TLS je open source SSL knihovna, dříve známá pod názvem PolarSSL, která vznikla v roce 2008 jako pokračování projektu XySSL. Nyní je podporována společností ARM Holdings a jejím cílem je jednoduchost a funkčnost, jak

[5] Equal preference cipher groups. `https://boringssl.googlesource.com/boringssl/+/858a88daf27975f67d9f63e18f95645be2886bfb`

uvádí na svých stránkách [9]. Je napsaná v jazyku C a podobně jako wolfSSL (sekce 2.2) je díky svým minimální hardwarovým požadavkům vhodná pro vestavěná zařízení. I přes svoji jednoduchost podporuje různé blokové a proudové šifry, certifikáty a mnoho dalších. V aktuální verzi 2.16.1 ale protokol TLS podporuje pouze do verze 1.2. Verze TLS 1.3 není aktuálně podporovaná a stránkách knihovny o plánované podpoře není žádná zmínka.

## 2.5 TLS Toolkit

Další knihovnou pro zabezpečení komunikace je TLS Toolkit, dříve známá jako MatrixSSL, od Inside Secure. Je opět napsaná v jazyku C a cílí na zařízení s omezenými zdroji. Na svých stránkách [10] autoři projektu uvádí, že velikost knihovny je pouhých 66 kB a v případě manuální optimalizace pro dané zařízení se může velikost ještě zmenšit. Podobně jako u wolfSSL i zde najdeme OpenSSL kompatibilní vrstvu, která umožní přechod z jedné knihovny na druhou. Čím tato knihovna ale vyniká, jsou zejména rychlé reakce na nové specifikace, což potvrzuje i fakt, že podpora TLS verze 1.3 byla přidána několik dní poté, co byla schválena finální specifikace této verze.

Verze TLS 1.3 je sice ve výchozím stavu povolená, ale ukázková implementace serveru má nastavenou jako výchozí verzi TLS 1.2. Pro navázání spojení na TLS 1.3 je nutné tuto verzi při spuštění serveru povolit. Je možné povolit více verzí najednou a samotné povolení lze provést také více způsoby — například pomocí přepínače `-W`, kterým lze nastavit i prioritu verzí. Tato volba bohužel nefunguje vždy podle očekávání. Pokud by uživatel chtěl například upřednostnit verzi TLS 1.2 před 1.3 a druhá strana bude podporovat obě verze, spojení se naváže na TLS 1.3. Jak již bylo zmíněno v podsekci 1.2.2, tak pokud obě strany podporují TLS 1.3, z bezpečnostních důvodů nemohou přejít na nižší verzi protokolu ani v případě, že nenajdou žádnou společnou šifrovou sadu.

V ukázkové implementaci serveru jsou povoleny všechny šifrové sady které knihovna podporuje, a je možné jednotlivé z nich zakázat zadáním jejich číselného kódu. Měnit jejich pořadí lze pouze omezeně, pomocí priority verzí.

Práce s šiframi přímo v programu je možná pomocí následující funkce:

```
int32_t matrixSslSetCipherSuiteEnabledStatus(
          ssl_t *ssl,
          psCipher16_t cipherId,
          uint32_t flags
         )
```

Této funkci je potřeba předat šifrový kontext, číslo šifrové sady se kterou chceme pracovat a příznak. Ten může být buď `PS_TRUE`, čímž dojde k opětovnému povolení dané šifrové sady a nebo `PS_FALSE`, kterým se šifrová sada zakáže. Ve výchozím nastavení jsou všechny sady povoleny.

## 2.6 Network Security Services

Alternativní možností pro implementaci bezpečné komunikace je soubor knihoven s názvem *Network Security Services*, zkráceně NSS, dostupný z [11]. Jedná se o řešení podporující více platforem, které je napsané převážně v jazyku C a zbývající část je psaná přímo v nízkoúrovňovém jazyku symbolických adres pro dosažení větší efektivity pro konkrétní platformy. Pod podobným názvem — Network Security Services for Java (JSS) — existuje ještě verze knihovny napsaná v jazyku Java. Vývoj obou zastřešuje organizace Mozilla Foundation, která poskytuje repozitáře a potřebnou infrastrukturu. Dále se na vývoji se podílí AOL, Red Hat, Google a další. Samotné knihovny z NSS nachází uplatnění například v programech Firefox, Thunderbird, Evolution nebo Pidgin. Protokol TLS 1.3 podporuje od 26. března 2019.

Podporované šifrové sady knihovna ukládá ve statickém poli, bez ohledu na verzi. Z pěti šifrových sad protokolu TLS 1.3 jsou zde implementovány pouze první tři, které požaduje standard. Jednotlivé šifrové sady je možné povolit pomocí funkce `SSL_CipherPrefGet` pro jeden konkrétní síťový socket nebo pomocí funkce `SSL_CipherPrefSetDefault` změnit výchozí nastavení. Takto je možné povolit nebo zakázat jednotlivé šifrové sady, ale není možné změnit jejich pořadí nebo použít aliasy pro povolení nebo zakázání více šifer najednou.

## 2.7 Java Secure Socket Extension

Jak již název napovídá, tak tato šifrovací knihovna *Java Secure Socket Extension* je napsaná v Javě a za jejím vývojem stojí společnost Oracle. Podle stránek dokumentace [12] je knihovna založena na stejných principech jako zbytek frameworku Java Cryptography Architecture. O podpoře protokolu TLS 1.3 není na stránkách žádná zmínka a knihovna jej podporuje jen do verze 1.2.

## 2.8 GnuTLS

Projekt GnuTLS je knihovna implementující SSL, TLS a s nimi související protokoly. Na svých stránkách [13] autoři projektu uvádí, že je napsaná v jazyku C a díky jednoduchému API je možné jí snadno začlenit do dalších linuxových knihoven. Kromě stažení ze stránek projektu je knihovna také dostupná ve formě balíčků pro všechny běžně používané linuxové distribuce. Dále knihovna nabízí rozhraní, rozšíření nebo wrappery, díky kterým ji lze použít v jazycích C++, Python, PHP nebo Guile. Nejnovější verzi protokolu TLS podporuje od verze GnuTLS 3.6.4, která byla vydána 24. září 2018.

Mezi podporované šifrové sady z verze TLS 1.3 patří všech pět definovaných v příslušném standardu. Pro práci se šiframi se zde používá jednotné

rozhraní pro všechny verze protokolu — prioritní řetězce (*Priority Strings*). Pomocí nich lze povolit šifrové sady seřazené podle různých způsobů. Řetězcem `PERFORMANCE` dojde k povolení sad používajících 128 bitové šifry a jsou seřazeny podle rychlosti. Dalším příkladem prioritního řetězce je `SECURE256`, který povolí pouze šifry používající 256 bitový klíč. Kromě těchto prioritních řetězců je možné použít i klíčová slova pro algoritmy a tím dodatečně povolit nebo zakázat vybrané šifrové sady. Tato klíčová slova jsou podobná šifrovým aliasům z OpenSSL a zastupují například algoritmy šifrování, výměny klíče, ověřovacích kódů nebo verzí protokolu. Oddělují se pomocí dvojtečky a lze je použít až za prioritním řetězcem. K rozlišení, zda se mají šifry přidat nebo odebrat, slouží následující tři znaky: `! - +` uvedené před klíčovým slovem, kde první dva slouží k odebrání a poslední k přidání.

Pokud bude zadán například následující šifrový řetězec: `SECURE256:-SHA256:-AES-256-GCM:+AES-128-CCM` dojde k povolení šifer používajících 256 bitový klíč, odebrání těch, které používají k šifrování AES-256-GCM a nakonec přidání AES-128-CCM.

## 2.9 Bouncy Castle

Za vývojem knihovny Bouncy Castle stojí australská nezisková společnost *the Legion of the Bouncy Castle Inc.*, která se stará o její správu a aktualizace. Pod stejným jménem se nachází dvě knihovny — jedna napsaná v jazyku Java a druhá v jazyku C#. I přesto že vývoj knihovny neustále pokračuje, na stránkách [14] autoři uvádí podporu protokolu TLS pouze ve verzi 1.1.

## 2.10 Rustls

Rustls, vyslovováno jako „rustles“, je moderní knihovna napsaná v jazyku Rust, za jejímž vývojem nestojí žádná organizace, ale jednotlivec — Joseph Birr-Pixton. Podle [15] je aktivně vyvíjena a protokol TLS 1.3 podporovala už v době, kdy existovala pouze neschválená pracovní verze. Od verze 0.14.0 vydané 30. září 2018 ale podporuje konečnou verzi tohoto protokolu odpovídající příslušnému RFC. Naopak protokoly, které jsou zastaralé, špatně navržené nebo nejsou bezpečné (př. TLS 1.1), v této knihovně podporované nejsou.

Před použitím TLS je potřeba vytvořit instanci objektu pro ukládání konfigurace serveru (v případě klienta je funkce podobná):

```
let mut config = rustls::ServerConfig::new(
    client_cert_verifier: Arc<dyn ClientCertVerifier>);
```

Tato funkce povolí šifry odpovídající výchozímu nastavení (tj. ty, které jsou uložené v proměnné `rustls::ALL_CIPHERSUITES`). Jejich priorita je určena pořadím, v jakém jsou v této proměnné uvedeny. Tento seznam šifer je ale

možné přepsat, stačí proměnné `ciphersuites` objektu `ServerConfig` přiřadit vlastní seznam v podobě vektoru textových řetězců:

```
config.ciphersuites = my_ciphersuites;
```

Šifry je nutné uvádět postupně, kromě `ALL_CIPHERSUITES` knihovna nepodporuje žádné seznamy šifer. Seznam všech aktuálně podporovaných šifrových sad je na stránkách[6] dokumentace pro jazyk Rust. Jména jednotlivých šifrových sad vychází ze standardů RFC a pro TLS 1.3 začínají předponou `TLS13_`. Z této nejnovější verze protokolu knihovna podporuje všech pět šifrových sad, které jsou definovány v [1].

[6] `https://docs.rs/rustls/0.16.0/rustls/enum.CipherSuite.html`

Kapitola 3

# Návrh

Tato kapitola shrnuje zjištěné nedostatky a navrhuje nové funkcionality, které tyto nedostatky řeší.

Jednotlivé knihovny řeší problematiku specifikace šifrových sad různě. Z výše uvedených OpenSSL nabízí řešení, které poskytuje v porovnání s ostatními největší flexibilitu při zadávání, a to také souvisí s důvody, proč bylo nutné oddělit rozhraní pro nejnovější verzi protokolu.

Jedním z hlavních důvodů pro rozdělení rozhraní pro zadávání šifer byla zpětná kompatibilita. Šifry z verze TLS 1.2 nelze použít ve verzi TLS 1.3 a opačně. Logickým důsledkem tohoto je fakt, že pokud není povolena žádná šifra z verze TLS 1.3, nelze v této verzi navázat spojení.

Pokud tedy někdo používal konfiguraci `ECDHE:!COMPLEMENTOFDEFAULT`, povolil tím všechny šifry používající výměnu klíče Diffie-Hellman nad eliptickými křivkami a pak všechny ostatní šifry, které nejsou ve skupině `DEFAULT` zakázal. Ve verzi TLS 1.2 je toto v pořádku, ale u verze TLS 1.3 to dvě šifry zakáže a tři výchozí nepovolí. V této nejnovější verzi již není součástí šifrové sady konkrétní algoritmus výměny klíče a tedy aliasy pro tyto algoritmy neobsahují žádné šifry z této verze. Naopak alias `COMPLEMENTOFDEFAULT` funguje jako doplněk všech verzí a snadno zakáže všechny ostatní šifrové sady.

V případě, že není povolena žádná šifra z protokolu 1.3 nebo 1.2, OpenSSL vrátí chybu a ukončí se. Je potřeba buď zvolit šifry tak, aby v obou protokolech byla alespoň jedna šifra, a nebo protokol, který nechceme používat, zakázat.

Tímto tedy může u některých uživatelů dojít k tomu, že po aktualizaci na novou verzi knihovny nebudou schopni bez změny konfigurace (a seznámení se s touto problematikou) navázat spojení. Proto bylo rozhraní pro zadávání rozděleno. Vzhledem k tomu, že šifrových sad v TLS 1.3 je zatím pouze pět, nebylo potřeba implementovat komplexní rozhraní, ale postačil pouhý seznam šifrových sad oddělených dvojtečkou.

Dalším potenciálním nedostatkem jsou možnosti výběru šifrových sad. V některých případech je vhodné brát v úvahu i preference klienta a ne pouze serveru. Jedním z případů je i šifra `ChaCha20-Poly1305` použitá v šifrové sadě

`TLS_CHACHA20_POLY1305_SHA256`. Některá mobilní zařízení mohou mít hardwarovou podporu pro tuto šifru a na takových zařízeních je tedy rychlejší a má i menší vliv na spotřebu oproti ostatním šifrám. Naopak servery a počítače můžou mít hardwarovou podporu nebo dedikované instrukce procesoru pro šifry typu `AES`, takže pro ně bude vhodnější použití těchto šifer.

Aktuálně existují tři způsoby jak, vybrat šifrovou sadu v závislosti na preferencích:

- preference šifer podle klienta. Tato možnost je výchozí a funguje tak, že se postupně prochází seznam šifer od klienta a zjišťuje se, zda ji nemá server také povolenou. Pokud ano, vybere se. Tento způsob zaručí, že z průniku dvou šifrových množin je vybraná šifra pro klienta nejlepší možná.

- Druhou možností je výběr podle preference serveru. Pro tuto možnost je nutné použít volbu `-serverpref` a funguje opačně než možnost první. Prochází se seznam serveru a v tomto pořadí se hledají šifry v seznamu klienta.

- Poslední možností je variace předchozí. Bude-li použita kromě volby `-serverpref` i volba `-prioritize_chacha`, dojde před vybráním konkrétní šifry k jejich přeuspořádání. Díky tomu se šifrové sady používající k šifrování algoritmus ChaCha20 dostanou na začátek serverového seznamu (pokud jsou povoleny) a výběr pak pokračuje jako v předchozí možnosti.

Mobilní zařízení musí brát ohled na omezené výpočetní prostředky a spotřebu energie, servery zase na velké množství klientů, kteří se připojují, takže nelze jednoznačně říct, která z výše uvedených možností je nejvýhodnější pro obě strany.

Na základě výše uvedených nedostatků se v této práci pokusím o implementaci následujících funkcí:

## 3.1 Operátor pro výběr verze

Operátor `|` (roura) bude možné použít k omezení platnosti zadaného šifrového aliasu. Dále jej bude možné použít za každým prvkem seznamu šifer (oddělených dvojtečkou). V případě, že bude použit opakovaně, k povolení více verzí jednoho šifrového aliasu, bude se chovat jako množinová operace sjednocení.

Aby byla zachována konzistence s již existujícím kódem, bude se verze porovnávat s minimální verzí u jednotlivých šifrových sad.

Pro zjednodušení povolení všech verzí bude existovat i alias `ALL` obsahující všechny verze.

Příkladem použití je řetězec:

`AES128|SSLv3|TLSv1.2:AES256|ALL`,

který bude rozdělen podle dvojtečky na dvě části:

- `AES128|SSLv3|TLSv1.2` — První část povolí šifrové sady z verzí SSLv3 a TLS 1.2, které používají šifrou AES se 128-bitovým klíčem.
- `AES256|ALL` — Druhá část povolí šifrové sady používající AES s klíčem délky 256 bitů.

## 3.2 Výchozí verze

V případě, že u aliasu nebude omezena verze výše uvedeným operátorem, použije se výchozí. Tuto výchozí verzi, respektive seznam výchozích verzí, bude možné zadávat pomocí nového parametru `version_mask`. Nebude-li tento parametr zadán, nastaví se na hodnoty všech verzí až do `TLS 1.2`, čímž bude zajištěna zpětná kompatibilita s aktuálním formátem seznamu šifer — `cipher_list`. Formát této výchozí verze bude stejný jako používá operátor pro výběr verze, tedy seznam verzí, které jsou odděleny znakem `|` (roura).

U aliasů, které reprezentují samotné verze, a u explicitně zadaných šifrových sad by nedávalo smysl brát v úvahu ani aktuální ani výchozí verzi. Proto v těchto případech bude zadaná verze ignorována. Tím, že se pro specifikaci verze používá znak `|` (roura) a pro oddělení šifrových sad nebo aliasů dvojtečka, je možné vždy jednoznačně určit, zda zadaný řetězec představuje šifrový alias (`AES:TLSv1.3`) nebo specifikaci verze (`AES|TLSv1.3`).

Podle výše uvedených pravidel při výchozí masce nastavené na hodnotu `SSLv3|TLSv1|TLSv1.2` a zadání následujícího řetězce:

`TLSv1.3:AES128|ALL:ECDH+AES|TLSv1.2|TLSv1:DH`,

dojde k následujícím krokům. Tyto kroky odpovídají jednotlivým částem, které jsou oddělené dvojtečkami:

- `TLSv1.3` — Nejprve se přidají šifrové sady z verze TLS 1.3, jedná se o šifrový alias pro konkrétní verzi, takže se parametr pro výchozí verzi neaplikuje.
- `AES128|ALL` — Dále se přidají šifrové sady používající šifru AES s délkou klíče 128 bitů. Protože se za tímto aliasem nachází operátor pro výběr verze a řetězec `ALL`, použijí se příslušné šifrové sady ze všech verzí.
- `ECDH+AES|TLSv1.2|TLSv1` — Následně se povolí sady, které používají pro výměnu klíče algoritmus Diffie-Hellman nad eliptickými křivkami, šifru AES k šifrování a jsou buď z verze TLS 1.2, nebo TLS 1.0.

- `DH` — Poslední položka šifrového seznamu přidá šifrové sady používající k výměně klíče algoritmus Diffie-Hellman. Protože zde není specifikovaná verze, vyberou se sady, jejichž verze je uvedena v parametru `version_mask`, tedy všechny verze kromě TLS 1.3.

## 3.3 Sjednocení rozhraní

V aktuální verzi knihovny OpenSSL existují dvě rozhraní, *ciphersuites* pro zadávání šifrových sad verze TLS 1.3 a *cipher_list* pro TLS 1.2 a starší.

Aby byla zachována zpětná kompatibilita, zůstanou obě rozhraní, pouze bude umožněno zadávat šifrové sady z verze TLS 1.3 do rozhraní *cipher_list* a v případě, že do něj bude šifrová sada zadána, bude druhé rozhraní *ciphersuites* ignorováno.

Výhodou možnosti konfigurace z jednoho rozhraní bude i zjednodušení pro aplikace třetích stran, které využívají tuto knihovnu. Nebudou muset implementovat další rozhraní pro nastavování šifrových sad z verze TLS 1.3. Nové rozhraní sice přináší i nový parametr (výchozí maska verze), ale její nastavení lze obejít tím, že bude maska vedena za každým šifrovým aliasem. Tento problém se týká i webového serveru Nginx, který nyní umožňuje nastavit pouze šifrové sady do verze TLS 1.2. Nový protokol je možné buď povolit a tím povolit i všechny výchozí šifrové sady a nebo jej zakázat.

Budeme-li chtít vybrat všechny šifrové sady, které používají šifru `AES` s délkou klíče 128 bitů, tak v aktuální verzi je potřeba vyjmenovat v rozhraní `ciphersuites` jednotlivé šifrové sady z verze TLS 1.3:

```
TLS_AES_128_GCM_SHA256,
TLS_AES_128_CCM_8_SHA256,
TLS_AES_128_CCM_SHA256
```

a dále v rozhraní `cipher_list` použít šifrový alias `AES128`.

V navrhované verzi stačí pouze zadat alias a specifikovat příslušné verze, v tomto případě všechny, tedy zadat `AES128|ALL`.

## 3.4 Posunutí na začátek

OpenSSL API má implementovaný operátor pro odsunutí šifrové sady nebo aliasu na konec seznamu. Není zde ale možnost, jak posunout šifrovou sadu nebo alias na začátek. Proto bude implementován operátor `^`, který bude plnit tuto funkci.

V případě zadání následujícího řetězce:

`TLSv1.3:^AES|ALL:RSA-PSK-CHACHA20-POLY1305`,

bude pořadí šifrových sad v jednotlivých krocích následující:

- `TLSv1.3` — Povolí se všechny šifrové sady z verze TLS 1.3, jejich pořadí bude následující:

  ```
  TLS_AES_256_GCM_SHA384,
  TLS_CHACHA20_POLY1305_SHA256,
  TLS_AES_128_GCM_SHA256,
  TLS_AES_128_CCM_8_SHA256,
  TLS_AES_128_CCM_SHA256.
  ```

- `^AES|ALL` — Poté dojde k přesunutí šifrových sad používajících šifru AES ze všech verzí na začátek. Po této operaci bude pořadí šifrových sad následující:

  ```
  TLS_AES_256_GCM_SHA384,
  TLS_AES_128_GCM_SHA256,
  TLS_AES_128_CCM_8_SHA256,
  TLS_AES_128_CCM_SHA256,
  TLS_CHACHA20_POLY1305_SHA256.
  ```

- `RSA-PSK-CHACHA20-POLY1305` — Na závěr dojde k povolení následující šifrové sady, která se umístí na konec seznamu:

  ```
  RSA-PSK-CHACHA20-POLY1305.
  ```

## 3.5 Prioritizace šifrové sady podle klienta

Aby bylo možné při výběru šifrové sady brát v úvahu nejen preferenci serveru, ale i klienta bude do API implementován nový operátor `*`, který se bude chovat podobně jako volba `-prioritize_chacha`, ale obecněji. Pokud tedy bude u šifrové sady nebo aliasu použit tento operátor a pokud se tato šifrová sada bude nacházet na začátku seznamu zaslaného od klienta, bude tato sada použita. V opačném případě nebude mít tento operátor žádný vliv při navazování spojení.

Budou-li mít server a klient nastaveny následující šifrové řetězce:

```
TLSv1.3:+CHACHA20|ALL:*CHACHA20|ALL:!TLS_AES_128_CCM_8_SHA256,
                TLSv1.3:^CHACHA20|ALL:!AES256|ALL,
```

tak význam jednotlivých částí pro server bude:

- `TLSv1.3` — Povolení všech šifrových sad z verze TLS 1.3.
- `+CHACHA20|ALL` — Přesunutí šifrových sad ze všech verzí, které používají algoritmus ChaCha20 na konec seznamu.

- `*CHACHA20|ALL` — Nastavení příznaku preference šifrovým sadám ze všech verzí, které používají algoritmus ChaCha20.
- `!TLS_AES_128_CCM_8_SHA256` — Trvalé zakázání šifrové sady `TLS_AES_128_CCM_8_SHA256`.

Význam částí klientova řetězce:

- `TLSv1.3` — Povolení všech šifrových sad z verze TLS 1.3.
- `^CHACHA20|ALL` — Přesunutí šifrových sad ze všech verzí, které používají algoritmus ChaCha20 na začátek seznamu.
- `!AES256|ALL` — Trvalé zakázání šifrových sad ze všech verzí, které používají AES s délkou klíče 256 bitů.

Tyto řetězce po zpracování vytvoří následující seznamy šifrových sad (pro jednoduchost jsou vybrány pouze šifry z verze TLS 1.3). Pro názornost jsou červeně označené šifrové sady, které jsou povoleny v obou seznamech a hvězdičkou znázorněn příznak preference.

Server:

```
  TLS_AES_256_GCM_SHA384,
  TLS_AES_128_GCM_SHA256,
  TLS_AES_128_CCM_SHA256,
* TLS_CHACHA20_POLY1305_SHA256.
```

Klient:

```
  TLS_CHACHA20_POLY1305_SHA256,
  TLS_AES_128_CCM_8_SHA256,
  TLS_AES_128_CCM_SHA256.
```

Při navázání spojení dojde, v závislosti na nastavení, k následující volbě šifer:

- ve výchozím nastavení se vybere šifrová sada s ohledem na preference klienta, tedy `TLS_CHACHA20_POLY1305_SHA256`, protože z průniku obou množin je v klientově seznamu nejvýše.
- V původním rozhraní, když by byl preferován pouze serverový seznam, by se vybrala šifrová sada `TLS_AES_256_GCM_SHA384`.
- V novém rozhraní, v případě preferování serverového seznamu, se využije příznak `*` a díky němu se vybere šifrová sada `TLS_CHACHA20_POLY1305_SHA256`, i když je na konci seznamu serveru. V tomto případě je výsledek stejný, jako při použití již existující volby `-prioritize_chacha` ze starého rozhraní, ale takto lze upřednostnit libovolné šifrové sady.

## 3.6 Skupiny o stejné preferenci

Další možností, jak brát v potaz preference klienta, jsou skupiny šifer o stejné preferenci. Jednotlivé skupiny budou reprezentovány hranatými závorkami a bude možné do nich zadávat šifrové sady nebo aliasy a nastavit jim příznak `*` pro upřednostnění v závislosti na klientovi. Zadání příznaku ve skupině (například `*ALL|ALL`) ovlivní pouze šifry ve skupině. Naopak bude-li zadán mimo skupinu, nebude mít vliv na šifry ve skupině. Případné odstraňování šifer pomocí operátoru `!` bude možné mimo skupinu. V případě použití skupin nebude v celém seznamu šifer možné používat příkazy `@STRENGTH` ani `@SECLEVEL`.

Protože používání operátorů pro přesouvání šifer na začátek nebo konec šifrového seznamu ve skupině nedává smysl, nebudou tyto operátory `^,+` uvnitř skupin povoleny. V případě použití mimo skupinu je budou šifry ve skupinách ignorovat.

Skupina, ve které budou zadány šifrové sady z více verzí, se bude chovat jako jedna skupina. Vzhledem k odlišnostem mezi verzemi TLS 1.2 a 1.3 bude vždy novější verze preferována. Pokud obě strany podporují TLS 1.3 a nenajdou žádnou společnou šifru z této verze, nemohou spolu z bezpečnostních důvodů spojení navázat. Proto tedy skupiny obsahující různé verze šifrových sad jsou pomyslně rozděleny a prochází se buď pouze ty, které obsahují TLS 1.3, nebo ty s verzemi do TLS 1.2.

Bude-li nastavena maska verzí na `ALL` a šifrový seznam:

```
[TLSv1.3:-CHACHA20]:AES256+RSA:CHACHA20:*CHACHA20:+AES256
```

dojde k následujícím krokům:

- `[ ]` — Vytvoření skupiny o stejné preferenci, ve které jsou přidány:

  `TLSv1.3` — Přidány všechny šifrové sady z verze TLS 1.3.

  `-CHACHA20` — Odebrání všech šifrových sad používajících algoritmus ChaCha20, v tomto případě bude odebrána pouze šifrová sada `TLS_CHACHA20_POLY1305_SHA256`.

- `AES256+RSA` — Přidání šifrových sad používajících AES256 a RSA.
- `CHACHA20` — Přidání všech šifrových sad používajících algoritmus ChaCha20.
- `*CHACHA20` — Přidání příznaku preference šifrovým sadám z předchozího bodu.
- `+AES256` — Posunutí šifrových sad používajících AES256 na konec seznamu. Tato položka ale neovlivní šifrovou sadu `TLS_AES_256_GCM_SHA384`, která je ve skupině.

Výsledný seznam lze zobrazit například pomocí nástroje `openssl ciphers -v`, jehož výstup se v upravené verzi bude skládat ze sedmi sloupců:

- příznaky — Grafické znázornění příslušnosti jednotlivých šifrových sad do skupin a příznaků preference.
- Název šifrové sady — Název šifrové sady, který používá knihovna OpenSSL, může se lišit od názvu uvedeného v RFC.
- Verze protokolu — Minimální verze protokolu, ve které byla šifrová sada definována.
- `Kx` — Algoritmus výměny klíče.
- `Au` — Algoritmus použitý pro autentizaci.
- `Enc` — Algoritmus pro samotné šifrování dat uvedený včetně velikosti klíče.
- `Mac` — Algoritmus pro ověření zpráv.

Seznam šifrových sad podle výše zadaného řetězce bude vypadat takto (poslední dva sloupce vynechány):

```
┌  TLS_AES_256_GCM_SHA384         TLSv1.3 Kx=any      Au=any
│  TLS_AES_128_GCM_SHA256         TLSv1.3 Kx=any      Au=any
│  TLS_AES_128_CCM_8_SHA256       TLSv1.3 Kx=any      Au=any
└  TLS_AES_128_CCM_SHA256          TLSv1.3 Kx=any      Au=any
*  TLS_CHACHA20_POLY1305_SHA256   TLSv1.3 Kx=any      Au=any
*  ECDHE-ECDSA-CHACHA20-POLY1305  TLSv1.2 Kx=ECDH     Au=ECDSA
*  ECDHE-RSA-CHACHA20-POLY1305    TLSv1.2 Kx=ECDH     Au=RSA
*  DHE-RSA-CHACHA20-POLY1305      TLSv1.2 Kx=DH       Au=RSA
*  RSA-PSK-CHACHA20-POLY1305      TLSv1.2 Kx=RSAPSK   Au=RSA
*  DHE-PSK-CHACHA20-POLY1305      TLSv1.2 Kx=DHEPSK   Au=PSK
*  ECDHE-PSK-CHACHA20-POLY1305    TLSv1.2 Kx=ECDHEPSK Au=PSK
*  PSK-CHACHA20-POLY1305          TLSv1.2 Kx=PSK      Au=PSK
   AES256-GCM-SHA384              TLSv1.2 Kx=RSA      Au=RSA
   AES256-CCM8                    TLSv1.2 Kx=RSA      Au=RSA
   AES256-CCM                     TLSv1.2 Kx=RSA      Au=RSA
   AES256-SHA256                  TLSv1.2 Kx=RSA      Au=RSA
   AES256-SHA                     SSLv3   Kx=RSA      Au=RSA
```

Kapitola 4

# Implementace

Tato kapitola popisuje detaily ze stávající implementace, které jsou důležité pro pochopení současného stavu a plánovaných úprav knihovny. Následně je popsána implementace nových funkcionalit, které jsou navrženy v kapitole 3. Dále se kapitola zabývá změnami, které s těmito funkcionalitami souvisí. Části kapitoly odpovídají jednotlivým logickým celkům implementace.

## 4.1 Použití knihovny

Zdrojové kódy knihovny OpenSSL je možné stáhnout z oficiálních stánek[7] nebo z repozitáře dostupného na stránkách GitHub[8]. Po stažení je potřeba knihovnu zkompilovat a volitelně spustit přiložené testy. Pokud vše proběhne v pořádku, je možné zkompilované soubory přesunout do systémových adresářů, aby mohly být spuštěny pomocí příkazu `openssl`. Výše uvedeného lze docílit následujícími příkazy:

```
git clone https://github.com/openssl/openssl.git
cd openssl
./config
make
make test
sudo make install
```

Následně je možné knihovnu začít používat. Prvním způsobem je využít jejího API/ABI rozhraní a volat příslušné funkce. Toto rozhraní je pro každou verzi podrobně popsané na stránkách OpenSSL[9]. Druhým možným použitím jsou nástroje příkazového řádku[10]. Tyto příkazy mají následující strukturu:

[7] `https://www.openssl.org/source/`
[8] `https://github.com/openssl/openssl.git`
[9] `https://www.openssl.org/docs/manmaster/man3/`
[10] `https://wiki.openssl.org/index.php/Command_Line_Utilities`

`openssl command [ command_options ] [ command_arguments ]`

Pro implementaci výše uvedených funkcionalit se stačí zaměřit pouze na následující nástroje:

- `openssl ciphers` — Jednoduchý nástroj pro převedení konfigurace šifer do seznamu šifer, který je seřazený podle preference. Lze jej použít jako první krok při ladění nastavení.
- `openssl s_server` — Základní SSL/TLS server, který je možné nastavit tak, aby poslouchal na daném portu.
- `openssl s_client` — Základní SSL/TLS klient, který se připojí na zadaný vzdálený nebo lokální server a umožní ladit nastavení serveru.
- `openssl s_time` — Klient, který otestuje TLS spojení s daným serverem.

U API bude potřeba upravit zejména funkce, které dostávají na vstupu řetězec s šifrovým seznamem. Mezi tyto funkce patří:

- `SSL_CTX_set_cipher_list`,
- `SSL_set_cipher_list`,
- `SSL_CTX_set_ciphersuites`,
- `SSL_set_ciphersuites`.

První dvě nastavují šifrové sady do TLS 1.2 a poslední dvě nastavují šifrové sady z TLS 1.3.

## 4.2 Detaily stávající implementace

Pro ukládání informací o jednotlivých šifrách slouží struktura `ssl_cipher_st`:

```
typedef struct ssl_cipher_st SSL_CIPHER;
struct ssl_cipher_st {
 uint32_t valid;
 const char *name;        /* text name */
 const char *stdname;     /* RFC name */
 uint32_t id;             /* id, 4 bytes, first is version */
 /*
  * changed in 1.0.0: these four used to be portions of a single
  * value 'algorithms'
  */
 uint32_t algorithm_mkey; /* key exchange algorithm */
```

```
 uint32_t algorithm_auth; /* server authentication */
 uint32_t algorithm_enc;  /* symmetric encryption */
 uint32_t algorithm_mac;  /* symmetric authentication */
 int min_tls;             /* minimum SSL/TLS protocol version */
 int max_tls;             /* maximum SSL/TLS protocol version */
 int min_dtls;            /* minimum DTLS protocol version */
 int max_dtls;            /* maximum DTLS protocol version */
 uint32_t algo_strength;  /* strength and export flags */
 uint32_t algorithm2;     /* Extra flags */
 int32_t strength_bits;   /* Number of bits really used */
 uint32_t alg_bits;       /* Number of bits for algorithm */
};
```

Záznam pro konkrétní šifru pak vypadá takto:

```
static SSL_CIPHER tls13_ciphers[] = {
  {
    1,
    TLS1_3_RFC_AES_128_GCM_SHA256,
    TLS1_3_RFC_AES_128_GCM_SHA256,
    TLS1_3_CK_AES_128_GCM_SHA256,
    SSL_kANY,
    SSL_aANY,
    SSL_AES128GCM,
    SSL_AEAD,
    TLS1_3_VERSION, TLS1_3_VERSION,
    0, 0,
    SSL_HIGH,
    SSL_HANDSHAKE_MAC_SHA256,
    128,
    128,
  },
...
}
```

První položkou je příznak, zda je se jedná o platnou šifru. Tato struktura se využívá i pro signalizační účely a v takovém případě je tento příznak nastaven na 0. Následuje jméno, pomocí kterého se šifrová sada zadává, a jméno podle RFC, u TLS 1.3 jsou tato jména shodná. Další položkou je identifikátor, který je složený z označení verze a následně unikátního čísla definovaného v RFC. Dále následují čtyři položky pro použité algoritmy. Díky tomu, že každý algoritmus má rezervovaný samostatný bit, je možné velice snadno vytvářet aliasy pomocí pouhého logického součtu požadovaných algoritmů. V dalších čtyřech položkách je rozsah verzí pro TLS a DTLS (protokol založený na TLS,

Tabulka 4.1: Číslování jednotlivých verzí protokolů

| Verze | Číselná hodnota |
|---|---|
| SSL 3 | 0x0300 |
| TLS 1 | 0x0301 |
| TLS 1.1 | 0x0302 |
| TLS 1.2 | 0x0303 |
| TLS 1.3 | 0x0304 |

fungující přes UDP). Poslední čtyři položky představují příznaky související s bezpečností dané šifrové sady a počet bitů klíče, které se používají.

Jednotlivé šifrové sady jsou pak v tomto formátu uloženy v poli struktur `SSL_CIPHER`. Aktuálně jsou pole rozdělena na šifrové sady pro verze do TLS 1.2 a samostatné pole pro verzi TLS 1.3.

## 4.3 Operátor pro výběr verze

Tato sekce je realizována následujícími commity:

- *Add function for parsing version string.*
  `https://github.com/OttoHollmann/openssl/commit/3c293d362d62de4d9ad29bcddc2598e60421be97`
- *Add parameter version_mask to function ssl_cipher_apply_rule.*
  `https://github.com/OttoHollmann/openssl/commit/f0e23ddad6ee48a634d2d01fd2ea89bcd11a859e`
- *Modify cipher_list parser to parse version specification.*
  `https://github.com/OttoHollmann/openssl/commit/1be0365fc33e6186323ea8d1ea7acd1ad89b0ff8`

Než bude implementován tento operátor, je potřeba připravit proměnnou, do které budou vybrané verze ukládány.

### 4.3.1 Ukládání seznamu verzí

Verze protokolů v knihovně OpenSSL jsou definovány pomocí maker, jejich hodnoty jsou zobrazeny v tabulce 4.1. Tyto číselné reprezentace verzí jsou zakotveny v příslušných RFC standardech a používají se například při navazování spojení. Pro uložení více verzí do jedné proměnné, například formou bitové masky, se ale nehodí.

Proto je pro uchování informací o verzích vytvořena nová struktura s názvem `ssl_version_st`, která umožní ke každé verzi uložit i dodatečné informace:

```
struct ssl_version_st {
    uint8_t is_default;
    const char *name;
    int ssl_version;
};

typedef struct ssl_version_st SSL_VERSION;
```

Příznak `is_default` určuje, zda bude výchozí maska tuto verzi obsahovat. Položky `name` a `ssl_version` pak obsahují řetězec se jménem a číselnou hodnotou dané verze. Pro snazší povolení všech verzí je vytvořena verze s názvem `ALL`, jejíž číslo verze je nastaveno na nulu. To umožní identifikovat, že se má verze ignorovat a mají se povolit šifry ze všech verzí.

Seznam všech verzí je uložen v poli struktur `ssl_version_st`:

```
static SSL_VERSION ssl_version_table[] = {
    {
        0,
        SSL_TXT_ALL,
        0,
    }, {
        1,
        SSL_TXT_SSLV3,
        SSL3_VERSION,
    },
    ...
};
```

Díky tomu pak má každý záznam s verzí přidělené číslo odpovídající pořadí v poli `ssl_version_table` a podle tohoto čísla lze jednoznačně nastavit příslušný bit v závislosti na tom, zda je verze vybraná. K takovému nastavení slouží bitový posun doleva o $i$, kde $i$ je index v poli:

```
*version_mask |= (1 << i);
```

Pokud tedy budeme chtít povolit verzi na nulté a druhé pozici, bude hodnota proměnné `version_mask` rovna číslu `5`, což ve dvojkové soustavě odpovídá hodnotě `0000 0101`.

### 4.3.2 Výběr verze

Nyní je možné přejít k samotnému výběru verze, tedy k vytvoření funkce, která se postará o zpracování textového řetězce zadaného uživatelem a převede jej na příslušnou masku verzí. Funkce se jmenuje `OPENSSL_version_list` a má dva argumenty. Prvním z nich je ukazatel na řetězec, který má být zpracován, druhým pak ukazatel na masku verzí, kam má být uložen výsledek. Návratovou hodnotou funkce je datový typ `int`, který udává počet zpracovaných znaků. V případě že došlo k chybě nebo nebyl nalezen žádný řetězec reprezentující verzi, je vrácena hodnota `0` a chyba je tak ošetřena volající funkcí.

Všechny operace s šiframi (přidání, přesunutí, ...) provádí funkce `ssl_cipher_apply_rule`, které je nyní pomocí parametru předána i maska verzí. Ne vždy je ale potřeba se rozhodovat podle všech parametrů, proto pokud je maska nulová, bude se ignorovat. Pokud je nenulová, pak se postupně prochází a testuje se, zda některé z povolených verzí odpovídá minimální verzi (uložené v `min_tls`) dané šifrové sady. Pokud není nalezena žádná shoda, operace se pro danou šifrovou sadu neprovede.

## 4.4 Výchozí verze

Tato sekce je realizována následujícími commity:

- *Add parameter for specifying default version mask in cipher_list.* `https://github.com/OttoHollmann/openssl/commit/a30ff7d3258e5bea1caa099f2511a4e443159755`
- *Add a condition to check order of configuration parameters.* `https://github.com/OttoHollmann/openssl/commit/9c24855de52b97e19705897875494749b8431e8e`

Aby bylo možné pracovat s výchozí verzí, je potřeba najít způsob pro uložení nebo předání této hodnoty. Následně se musí přidat možnost specifikace této výchozí verze všude, kde je možné šifrové sady zadávat.

### 4.4.1 Předání hodnoty výchozí verze

Jednou z možností by bylo přidat výchozí masku do globální struktury `SSL_CTX`, která v sobě udržuje informace o povolených šifrových sadách, verzích protokolů a výchozí hodnoty, ze kterých jsou později vytvářeny další SSL struktury potřebné pro navázání spojení. Tato možnost by byla jednoduchá na implementaci, ale z logického pohledu nedává smysl, aby v takové struktuře byl parametr, který se použije pouze jednou při zadávání nebo změně šifer. Z toho důvodu je parametr výchozí masky předáván jako argument funkcím `SSL_set_cipher_list` a `SSL_CTX_set_cipher_list`, které se starají o převedení textového řetězce na seznam šifrových sad.

### 4.4.2 Aliasy

Výběr verzí u aliasů řeší předchozí sekce, ale alias `DEFAULT` se od ostatních liší svým fungováním. Může být použit pouze na začátku šifrového seznamu a v případě jeho zadání se zavolá funkce `OSSL_default_cipher_list()`, která vrátí následující textový řetězec: `ALL:!COMPLEMENTOFDEFAULT:!eNULL`. Díky tomu povolí všechny výchozí šifrové sady a v aktuálním šifrovém seznamu trvale zakáže používání jiných šifrových sad než těch výchozích.

Pro specifikování verze by bylo potřeba zkopírovat specifikátor verzí za každou položku tohoto seznamu, ideálně ještě ve funkci `OSSL_default_cipher_list()`. Vzhledem k tomu, že se jedná o API funkci, tak tato změna není žádoucí. Místo toho je při načtení řetězce `DEFAULT` načten i jeho případný specifikátor verze, který je pak jako výchozí verze předán funkci starající se o samotné zpracování řetězce.

### 4.4.3 Upravení API

Pro použití API je potřeba příslušným funkcím přidat parametr. Protože je ale knihovna psaná v jazyku C a nikoliv v C++, není možné přidat parametr funkce, který by měl nastavenou výchozí hodnotu a nemusel se při volání funkce zadávat. Pouhé přidání parametru by tedy porušilo kompatibilitu API rozhraní a knihovna by tak nemusela být použitelná s existujícími programy.

Proto je funkcionalita současných funkcí přesunuta do nových, které mají podobný název a o parametr víc. Původní funkce pak zavolají ty nové a na místo nového parametru dosadí nulovou hodnotu:

```
int SSL_CTX_set_cipher_list(SSL_CTX *ctx, const char *str)
{
    return SSL_CTX_set_cipher_list_and_mask(ctx, str, 0);
}

int SSL_CTX_set_cipher_list_and_mask(SSL_CTX *ctx,
                                     const char *str,
                                     int default_version_mask)
{
    ...
}
```

### 4.4.4 Upravení nástrojů příkazové řádky

Přidání parametru do nástrojů příkazové řádky je poněkud složitější. U nástrojů `s_time` a `ciphers` stačí přidat do přepínače switch návěstí a v něm načíst zadaný argument. Nástroje `s_client` a `s_server` používají ke konfiguraci funkci `SSL_CONF_cmd`, jejíž cílem je zjednodušit a sjednotit načítání parametrů

z příkazové řádky nebo konfiguračních souborů. Interně tato konfigurace funguje tak, že pro každý parametr ukládá adresu funkce, řetězec pro nastavení z konfiguračního souboru, řetězec, pomocí kterého lze nastavit hodnotu parametru z příkazové řádky, příznaky a typ — textový řetězec, název souboru, název adresáře. Tyto položky jsou uloženy v poli struktur `ssl_conf_cmd_tbl`, které se při načítání parametrů postupně prochází. V případě, že zadaný název parametru odpovídá tomu uloženému ve struktuře, zavolá se příslušná funkce, která jej zpracuje.

Samotné přidání parametru probíhá pomocí následujících maker:

```
#define SSL_CONF_CMD(name, cmdopt, flags, type) \
        {cmd_##name, #name, cmdopt, flags, type}

#define SSL_CONF_CMD_STRING(name, cmdopt, flags) \
        SSL_CONF_CMD(name, cmdopt, flags, SSL_CONF_TYPE_STRING)
```

Pro přidání parametru tedy je potřeba řádek:

```
SSL_CONF_CMD_STRING(VersionMask, "version_mask", 0)
```

Ten po zpracování makry rovnou inicializuje příslušnou strukturu. To zajistí, že se při zpracovávání parametrů zavolá funkce `int cmd_VersionMask(SSL_CONF_CTX *cctx, const char *value)`, která uloží zadanou masku verzí a až dojde k nastavování šifer, předá ji této funkci.

To s sebou nese i fakt, že pokud budou parametry zadány ve špatném pořadí, nebude mít maska verzí žádný efekt. Proto je při nastavování šifer nastaven příznak, který signalizuje, že již je šifrový seznam nastaven, a pokud se v takovém případě uživatel pokusí o nastavení masky verzí, je to považováno za chybu.

### 4.4.5 Výchozí hodnota

Pokud parametr není zadán, je potřeba jej nastavit na výchozí hodnotu. To se děje prostřednictvím funkce `OPENSSL_version_list`, které se místo řetězce, jenž má být zpracován, předá hodnota `NULL`. Tato funkce pak do masky verzí uloží všechny výchozí verze — tedy všechny, které mají nastavený příznak `is_default`.

## 4.5 Sjednocení rozhraní

Tato sekce je realizována následujícími commity:

- *Replaced cipher_list with structure containing cipher_list and flags for each cipher.*
  `https://github.com/OttoHollmann/openssl/commit/7bfd232b45646bceee5a54f5593389624edd846d`
- *Add function for counting ciphers by version, which were entered via cipher list interface.*
  `https://github.com/OttoHollmann/openssl/commit/99bfc9acaf909a61b84cf19d6bc4ac1150c5ccd7`
- *Add TLSv1.3 ciphers to the cipher list configuration interface.*
  `https://github.com/OttoHollmann/openssl/commit/ac095cab48568078533e22d990b5c6a4f9ade1c6`
- *Remove ciphers entered from ciphersuites if they are not needed.*
  `https://github.com/OttoHollmann/openssl/commit/d21d10baa7be2c7a86395fdc57a2788536960ba5`
- *Merge cipher table tls13_ciphers with ssl3_ciphers.*
  `https://github.com/OttoHollmann/openssl/commit/fb9844d781b1b2d37c0afe52b6ca26e487cbe2e0`

Pro možnost sloučení rozhraní je nutné se seznámit s aktuálním způsobem přidávání šifrových sad.

### 4.5.1 Přidání šifrové sady

Šifrové sady zadané z jednoho nebo druhého rozhraní se interně ukládají do jednoho seznamu. Šifrový kontext `SSL_CTX` používá k uložení šifrových sad dvě členské proměnné `tls13_ciphersuites` a `cipher_list`. Při jeho vytváření je nejprve pomocí funkce `OSSL_default_ciphersuites` inicializována proměnná `tls13_ciphersuites` na první tři šifrové sady z TLS verze 1.3. Následně se zavolá funkce `ssl_create_cipher_list`, které se jako jeden z argumentů předá seznam TLS 1.3 šifrových sad a textový řetězec obsahující šifrové sady z verzí TLS 1.2 nebo starších. V případě vytváření kontextu jsou použity výchozí hodnoty, tedy ty, které vrací funkce `OSSL_default_cipher_list`. Následně funkce vytvoří šifrový seznam, na jeho začátek umístí šifrové sady z verze TLS 1.3 a pak až přidá šifrové sady specifikované podle textového řetězce zadaného jako argument.

Při následném zadání řetězce pomocí rozhraní `cipher_list` dojde opět k zavolání funkce `ssl_create_cipher_list`. Pokud ale uživatel zadá šifry pomocí rozhraní `ciphersuites`, je nutné nejprve zavolat funkci

`set_ciphersuites`, která zpracuje zadaný textový řetězec a uloží příslušné šifrové sady do členské proměnné `tls13_ciphersuites`. Následně se zavolá funkce `update_cipher_list`, která vezme aktuální seznam šifer, z jeho začátku odstraní všechny šifrové sady z verze TLS 1.3 a na začátek zkopíruje ty z `tls13_ciphersuites`.

Mezi voláním funkce `ssl_create_cipher_list` pro zpracování šifrového řetězce a výsledným šifrovým seznamem je několik kroků. Jak bylo zmíněno na začátku kapitoly, šifrová sada je nejprve uložena v poli struktur. Poté je ve funkci `ssl_cipher_collect_ciphers` toto pole procházeno a podle zakázaných algoritmů se kontroluje, zda má být šifra povolena. Pokud ano, je přidána do obousměrně zřetězeného seznamu, který je tvořen následující strukturou:

```
typedef struct cipher_order_st {
    const SSL_CIPHER *cipher;
    int active;
    int dead;
    struct cipher_order_st *next, *prev;
} CIPHER_ORDER;
```

Položka `active` signalizuje, že je daná šifra vybrána a `dead` že byla trvale zakázána. Výstupem této funkce jsou pak ukazatele na začátek a konec tohoto seznamu, ve kterém jsou pouze šifrové sady, které nebyly zakázány. V tuto chvíli mají všechny šifrové sady nastavený příznak `active` na hodnotu nula a až na základě výchozího nebo uživatelem zadaného šifrového řetězce je tento příznak změněn. Následně dojde k seřazení tohoto seznamu podle sily šifer a použitých algoritmů.

Nyní již přichází na řadu zadaný šifrový řetězec, který se postupně prochází po jednotlivých šifrových sadách nebo aliasech a ty se hledají v původním poli. Pokud je nalezena shoda (šifrové sady nebo šifrového aliasu), zavolá se funkce `ssl_cipher_apply_rule`, která zajistí „přidání“ příslušných šifrových sad. Pokud má šifrová sada nastavený příznak `active`, neděje se nic. V opačném případě je nastaven tento příznak a šifrová sada je v seznamu zařazena na konec. Podobně funguje i operátor `+` pro posunutí šifrových sad na konec seznamu. Operátor `-` pro odstranění šifrových sad funguje opačně, šifrám které mají nastavený příznak `active`, tento příznak odebere a přesune je na začátek seznamu.

Na závěr se projde tento seznam a šifry, které mají nastavený příznak `active`, se přidají do výsledného seznamu typu `STACK_OF(SSL_CIPHER)` pomocí funkce `sk_SSL_CIPHER_push`.

### 4.5.2 Nahrazení šifrového seznamu strukturou

Aby bylo možné rozlišit, které šifry byly zadány přes které rozhraní, a obecně bylo možné přidat k šifrám dodatečnou informaci, je šifrový seznam `cipher_list` nahrazen strukturou:

```
struct ssl_cipher_list_flags_st {
    STACK_OF(SSL_CIPHER) *cipher_list;
    uint8_t *flags;
};
typedef struct ssl_cipher_list_flags_st SSL_CIPHER_FLAGS;
```

Tato struktura obsahuje původní šifrový seznam a pole typu `uint8_t` umožňující ke každé šifře z šifrového seznamu uložit příznaky.

### 4.5.3 Příznaky šifrových sad

Přidání příznaků je nejvhodnější udělat pomocí funkce `ssl_cipher_apply_rule`. Aby tato funkce mohla ovlivnit výsledné pole příznaků, je do struktury `cipher_order_st` přidán příznak `uint8_t flags`, který se pak při vytváření zásobníku šifer průběžně kopíruje do pole příznaků.

### 4.5.4 Odlišení rozhraní

Pokud bude umožněno zadávat šifrové sady TLS 1.3 pomocí rozhraní `cipher_list`, je potřeba rozlišit, ze kterého rozhraní byly zadány. K tomu slouží výše uvedení pole příznaků, kde všem šifrám zadaným pomocí rozhraní `ciphersuites` je nastaven příznak `CIPHER_FLAG_FROM_CIPHERRSUITE`. Podle tohoto příznaku pak funkce `update_cipher_list` pozná, zda byla daná šifra zadaná pomocí `ciphersuites` a má být při aktualizaci šifrového seznamu odstraněna, nebo ne.

### 4.5.5 Sloučení polí s šifrovými sadami

Spolu s rozdělením rozhraní pro zadávání šifrových sad byla rozdělena i pole, ve kterých jsou tyto sady uloženy. Šifrové sady zadané pomocí `ciphersuites` se pomocí zpětně volané funkce hledají v obou polích, ale následně se provádí kontrola, zda jsou tyto sady z poslední verze TLS 1.3. Původní šifry z rozhraní `cipher_list` se hledají pouze v poli `ssl3_ciphers` a kontrola verze se neprovádí. Pokud je nyní implementovaná maska verzí, která ve výchozím nastavení odfiltruje verzi TLS 1.3, nic nebrání opětovnému sloučení těchto dvou polí do jednoho.

### 4.5.6 Odstranění šifrových sad zadaných pomocí ciphersuites

Problém ponechání druhého rozhraní spočívá v tom, že pokud není zadán seznam šifrových sad, povolí se všechny výchozí z verze TLS 1.3. Jak ale rozlišit, jestli uživatel zapomněl toto rozhraní nastavit, nebo jej záměrně nechal prázdné a chce použít jeho výchozí hodnoty?

Proto je implementována funkce `SSL_CTX_count_ciphers_by_version`, která má dva parametry — šifrový kontext a číslo verze. Podle nich pak spočítá, kolik šifrových sad dané verze bylo pro konkrétní kontext zadáno pomocí rozhraní `cipher_list`. To je možné díky výše uvedeným příznakům.

Pokud tedy bude pomocí rozhraní `cipher_list` zadána alespoň jedna šifrová sada z verze TLS 1.3 nebo uživatel zadal nenulovou masku, je zřejmé, že uživatel chce použít toto vylepšené rozhraní. V tomto případě je rozhraní `ciphersuites` nastaveno na prázdný řetězec, což má za následek smazání šifrových sad z proměnné `tls13_ciphersuites`.

## 4.6 Prioritizace šifrové sady podle klienta

Tato sekce je realizována následujícími commity:

- *Move code for checking ciphers into separate function.*
  `https://github.com/OttoHollmann/openssl/commit/8f6dc7f956b62606a81c43255bc9c529540efdc4`

- *Add operator for prioritizing ciphers depending on client's preference.*
  `https://github.com/OttoHollmann/openssl/commit/506f5f98d31b7a33f43d16196c14feb6528cd418`

- *Check preference flag for client's first common cipher instead of first cipher.*
  `https://github.com/OttoHollmann/openssl/commit/031beddf3bac1144b8657740a4ea421ab23e09bf`

Nejvhodnějším místem pro zpracování příznaků preference je opět funkce `ssl_cipher_apply_rule`, která využije již existující proměnné `flags` a pomocí bitové operace `OR` přičte příznak `CIPHER_PREFER`. Následně se výše popsaným způsobem tato hodnota příznaku dostane až k samotným funkcím, které vybírají výslednou šifrovou sadu.

Před samotným výběrem šifrové sady se na základě nastavení serveru přiřadí do proměnných `prio` a `allow` seznamy serveru a klienta podle toho, který má být preferovaný. Dále se do proměnné `flags` přiřadí ukazatel na pole příznaků nebo hodnota `NULL` podle toho, který seznam je preferován.

Proces výběru (pokud existuje pole příznaků) pokračuje zkontrolováním, zda šifrová sad nacházející se v klientově seznamu na prvním místě je mezi povolenými na serveru. Pokud ano, zkontroluje se, jestli má nastaven příznak pro preferenci a jestli vyhoví všechny dílčí algoritmy. V případě, že šifrová sada projde i těmito testy, je vybrána. Následně byla tato část upravena tak, aby v seznamu klienta nalezla místo první šifrové sady první společnou šifrovou sadu a u té zkontrolovala příznaky.

## 4.7 Posunutí na začátek

Tato sekce je realizována následujícím commitem:

- *Add operator for moving ciphers to the beginning of the cipher_list.*
  `https://github.com/OttoHollmann/openssl/commit/e627da2ef6ea2579fad3da6149819e8dfbb4c4c1`

Funkcionalita pro posunutí šifrových sad na začátek seznamu již je v knihovně implementovaná. Používá se například pro seřazení seznamu všech šifer před tím, než dojde k samotnému výběru šifrových sad. To vede k tomu, že v případě zadání šifrového aliasu budou ty bezpečnější výše v seznamu.

Aby bylo možné používat operátor pro posunutí šifrové sady na začátek seznamu, je potřeba tento operátor definovat a v případě jeho zadání nastavit příslušné pravidlo, které pak zajistí samotné vykonání operace.

## 4.8 Skupiny o stejné preferenci

Tato sekce je realizována následujícími commity:

- *Add parser and errors for equal-preference groups.*
  `https://github.com/OttoHollmann/openssl/commit/3810d3028b9417750b7eafb3e70b46501ce29ea3`
- *Add support for the equal-preference groups to the cipher selection function.*
  `https://github.com/OttoHollmann/openssl/commit/7c2062e2f653bf3319aba39a99cbb3fcea2dfb60`
- *Fixed behavior of operators in equal-preference group.*
  `https://github.com/OttoHollmann/openssl/commit/f5275ffe79d30ab805d2c59e9714c9181ea1f35b`
- *Add a function for cipher flags description.*
  `https://github.com/OttoHollmann/openssl/commit/e396459488a992e1c26a63c2a663629c3b64d6ae`

Se skupinami o stejné preferenci souvisí ošetření vstupu a také grafické znázornění skupin a příznaků jednotlivých šifrových sad.

### 4.8.1 Přidání chybových hlášek

Chybové hlášky jsou definované v souborech `include/openssl/sslerr.h`, kde jsou pomocí maker jednotlivým chybovým hláškám definovány číselné konstanty, a `ssl/ssl_err.c`, kde pomocí této konstanty, kódu knihovny a

jiného makra je vytvořen unikátní kód chyby, ke kterému je přiřazen i textový popis. Kvůli zachování konzistence by se ale tyto soubory neměly manuálně upravovat. Preferovaným způsobem je provést příslušné změny v souboru `crypto/err/openssl.txt` a pak pomocí generátoru `util/mkerr.pl` vygenerovat nové verze chybových souborů.

### 4.8.2 Označení skupin

Dokud jsou šifrové sady uložené v obousměrně zřetězeném seznamu, nelze spoléhat na jejich pořadí, protože může velice snadno dojít k jejich přesouvání. Proto bylo potřeba vymyslet jednoznačný způsob identifikace, do které skupiny jednotlivé šifrové sady patří. Protože obyčejný příznak k tomu použít nelze, je do struktury `cipher_order_st` přidána položka `int group_id` ve které budou uloženy unikátní identifikátory skupin.

### 4.8.3 Úprava parseru

Dále je potřeba upravit parser, který převádí textový řetězec na seznam šifrových sad. Jeho úkolem mimo jiné je zkontrolovat vstup, jestli nebyly zadané vnořené skupiny nebo nedochází k používání speciálních operátorů pro řazení, když jsou použity skupiny o stejné preferenci. Dalším jeho úkolem je generovat unikátní identifikátor skupin, který pro šifrové sady, které nejsou ve skupině, bude nabývat hodnoty nula, a ty, které ve skupině jsou, budou mít hodnotu podle toho, o kolikátou skupinu se jedná. Při každém načtení znaku `[` pak dojde ke zvýšení tohoto čísla.

### 4.8.4 Omezení funkčnosti operátorů ve skupině

Pro omezení platnosti operátoru volby preference byl funkci `ssl_cipher_apply_rule` přidán parametr s aktuální hodnotou identifikátoru skupiny. Na základě toho se pak funkce může rozhodnout, zda šifrová sada patří do aktuální skupiny a má se jí přidat příznak pro preferenci.

Aby bylo zabráněno přesouvání operátorů na začátek nebo konec seznamu u šifrových sad, které jsou ve skupině, je do této funkce přidána podmínka, která zkontroluje, zda identifikátor skupiny u šifrové sady je roven nule, a tedy nepatří do žádné skupiny.

### 4.8.5 Příznaky skupin

Způsob uložení příslušnosti do skupiny byl inspirován implementací v knihovně BoringSSL, kde je příznak použit pro všechny šifry ve skupině kromě poslední. Takto může vypadat zadaný šifrový seznam a jeho příznaky:

```
šifrový seznam:     [ A : B : C : *ALL ] : D : [ E : F ]
příznaky skupin:      1   1   0            0     1   0
příznaky preference:  1   1   1            0     0   0
```

S tímto řešením je tedy možné pomocí jednobitového příznaku jednoznačně určit, kde začíná a končí skupina o stejné preferenci. Ve výše uvedeném příkladu je v první skupině použit operátor pro preferenci, ale protože je použit ve skupině, aplikuje se pouze na šifry v této skupině.

### 4.8.6 Vybrání šifrové sady

Nyní je na řadě úprava samotné funkce, která vybírá výslednou šifrovou sadu. Pokud je preferován serverový seznam, nejprve se zkontroluje klientova první společná šifrová sada. Pokud tato šifrová sada nemá nastaven příznak, dojde k výběru podle skupin stejné preference. Začne se postupně procházet serverový seznam a zjišťuje se, zda jsou tyto šifrové sady povolené. Pokud ano, je uložen index v klientově seznamu a procházení pokračuje dál. Na konci každé skupiny se pak nalezne nejmenší index (pokud byla nalezena nějaká společná šifrová sada) a skupina s tímto indexem se pak povolí. Pokud ne, prochází se serverový seznam dále.

### 4.8.7 Funkce pro popis příznaků

Je důležité také nezapomenout na nástroj `openssl ciphers`, který se používá pro ověření toho, jak knihovna interpretuje zadaný řetězec. Tento nástroj načte seznam povolených šifrových sad a poté pro každou šifrovou sadu o ní vypíše informace pomocí funkce `SSL_CIPHER_description`. Protože pro správnou interpretaci příznaků je potřeba znát alespoň příznak u předchozí šifrové sady, je implementována nová funkce `SSL_CIPHER_flags_description`, která při volání dostane strukturu `SSL`, ze které získá pole příznaků, index, pro kterou šifru má vrátit příznaky, a ukazatel na buffer, kam má zapsat výsledek. Pro výše uvedený příklad by nástroj `openssl ciphers` vrátil následující výstup:

```
$ openssl ciphers -v '[ A : B : C : *ALL ] : D : [ E : F ]'
┌ * A   TLSv1.?   Kx=?      Au=?      Enc=?     Mac=?
│ * B   TLSv1.?   Kx=?      Au=?      Enc=?     Mac=?
└ * C   TLSv1.?   Kx=?      Au=?      Enc=?     Mac=?
    D   TLSv1.?   Kx=?      Au=?      Enc=?     Mac=?
┌   E   TLSv1.?   Kx=?      Au=?      Enc=?     Mac=?
└   F   TLSv1.?   Kx=?      Au=?      Enc=?     Mac=?
```

Jednoprvkové skupiny zde vzhledem k fungování příznaků znázorněny nejsou. Takové skupiny totiž ve výsledném seznamu nemají žádný význam. Mají vliv pouze při nastavování preferencí nebo přesouvání šifrových sad.

## 4.9 Dokumentace

Tato sekce je realizována následujícím commitem:

- *Documentation update for equal-preference groups and improved cipher list format.*
  `https://github.com/OttoHollmann/openssl/commit/727e2360e0685f66acb58634ded9ea98c2b6e18a`

Poslední částí implementace je úprava dokumentace. Ta je psaná ve formátu `.pod`[11] a jedná se o jednoduchý značkovací jazyk, jehož cílem je především jednoduchost, snadná čitelnost a převoditelnost do jiných formátů jako je TEX nebo `Markdown`. Z těchto souborů jsou pak pomocí `Perl` skriptů vygenerovány manuálové nebo webové stánky.

[11] Plain Old Documentation

Kapitola 5

# Testování

Tato kapitola se zabývá analýzou kódu, testy před nainstalováním knihovny, vlivem změn na rychlost navázání spojení, testováním vestavěnými nástroji knihovny a dále integraci s vybranými aplikacemi třetích stran.

## 5.1 Statická analýza kódu

Pro statickou analýzu byl použit nástroj `coverity scan`, dostupný z [16]. Tento nástroj dokáže v kódu odhalit slabiny, zranitelnosti a chyby popsané na stránkách [17] včetně jejich CWE označení. Při testování upravené verze knihovny byly všechny chyby způsobené těmito úpravami odstraněny.

## 5.2 Dynamická analýza kódu

Knihovna byla otestována pomocí nástroje Valgrind [18]. Při jejím běhu ani po jejím skončení nebyly detekovány žádné chyby ani neuvolněné paměťové bloky.

## 5.3 Interní testy

Pro ověření, že kompilace proběhla v pořádku a že vše funguje jak má jsou v knihovně implementovány rozsáhlé testy. Po zkompilování je možné tyto testy spustit pomocí příkazu `make test`. Aby i ostatní vývojáři mohli ověřit, že neporušili funkčnost žádné části, byly napsány testy pro všechny přidané funkcionality.

## 5.4 Srovnání časové náročnosti

Pro porovnání časové náročnosti původní implementace a nové implementace byla původní a upravená knihovna změřena pomocí nástroje `s_time`. Prů-

Tabulka 5.1: Porovnání knihoven pomocí nástroje s_time

| **Knihovna** | **#spojení/vteřinu** | **#spojení/vteřinu*** |
|---|---|---|
| OpenSSL — původní | 1198,48 | 1197,49 |
| OpenSSL — upravená | 1182,66 | 1187,73 |

*Opětovné použití identifikátoru relace.

měrná doba pro navázání spojení byla změřena pomocí nástrojů `s_server` a `s_client`.

Výstup nástroje `s_time` je v příloze C a porovnání časů v tabulce 5.1. Tyto hodnoty ukazují, že nová implementace mírně snížila počet navázaných spojení za jednu vteřinu, doba navázání jednoho spojení prodloužila přibližně o 1,5 %. V případě znovupoužití identifikátoru relace došlo k prodloužení doby potřebné pro navázání spojení o méně než 1 %, což je v obou případech zanedbatelné prodlení a může být ovlivněné chybou měření.

Dále byla změřena doba potřebná k navázání a ukončení 10 000 spojení pomocí `s_client` s `s_server`. Tyto časy ale vyšly stejné, protože zde převládla režie operačního systému nad samotným navázáním spojení.

## 5.5 Testování nástroji OpenSSL

Běžně dostupné nástroje nejsou vhodné pro ověření funkčnosti skupin stejné preference nebo preferování jednotlivých šifrových sad. Proto byly k tomuto účely využity nástroje `s_client` a `s_server`. Tyto nástroje byly spouštěny s různými šifrovými seznamy uvedenými v příloze B. Vzhledem k tomu, jak odlišné jsou šifrové sady jednotlivých verzích, jsou v těchto testech zvlášť seznamy pro TLS 1.3 a TLS 1.2. Ve všech případech došlo k vybrání očekávané šifrové sady.

## 5.6 Webový server Nginx

Upravené verze byla vyzkoušena s webovým serverem Nginx dostupným z [19]. Postup pro jeho stažení a zkompilování je následující:

```
wget https://nginx.org/download/nginx-1.16.1.tar.gz
tar -xvf nginx-1.16.1.tar.gz
cd nginx-1.16.1
./configure --with-http_ssl_module
make
sudo make install
```

Tento postup vyžaduje nainstalovanou OpenSSL knihovnu. Pokud knihovna není nainstalovaná, je možné pomocí volby `--with-openssl=/path/to/openssl` při konfiguraci nastavit umístění zdrojových souborů knihovny a při následném spuštění příkazu `make` bude spolu s webovým serverem zkompilována i knihovna.

V aktuální verzi se šifrové sady nastavují v konfiguračním souboru pomocí `ssl_ciphers`, ale je možné takto nastavit pouze šifrové sady do verze TLS 1.2. Pokud chceme povolit protokol TLS 1.3, je nutné upravit proměnnou `ssl_protocols`. Pokud takto povolíme protokol TLS 1.3, dojde k povolení všech tří výchozích šifrových sad, není možné jednotlivé sady zakázat nebo povolit ty, které nejsou výchozí.

S upravenou verzí knihovny to možné je. Nejprve je nutné povolit tento protokol v proměnné `ssl_protocols` a následně je možné pomocí `ssl_ciphers` zadávat libovolné šifrové sady.

Pokud budeme chtít povolit všechny šifrové sady z aliasu `HIGH` z protokolů TLS 1.2 a TLS 1.3 a preferovat serverové šifrové sady, bude konfigurace následující:

```
ssl_prefer_server_ciphers on;
ssl_protocols TLSv1.2 TLSv1.3;
ssl_ciphers 'HIGH|TLSv1.2|TLSv1.3';
```

V tomto případě není nutné uvádět za aliasem `HIGH` konkrétní verze, protože je zde použita volba `ssl_protocols` nutná k povolení příslušných protokolů. Stačilo by do šifrového seznamu napsat `HIGH|ALL`.

Zkoušené kombinace serverových a klientských šifrových seznamů byly stejné jako v příloze B a pro komunikaci byly vybrány stejné šifrové sady. Preference šifrových sad tedy funguje dle očekávání.

## 5.7 Webový server Apache

Druhým webovým serverem, se kterým byla knihovna otestována, je Apache HTTP Server (httpd) dostupný z [20]. Postup pro jeho zkompilování je následující:

```
wget https://www-us.apache.org/dist//httpd/httpd-2.4.41.tar.gz
tar -xvf httpd-2.4.41.tar.gz
cd httpd-2.4.41
./configure --enable-ssl
make
sudo make install
```

V dřívějších verzích se k nastavování šifrových sad používala volba `SSLCipherSuite`. S rozdělením zadávacích rozhraních v OpenSSL bylo nutné rozdělit tato rozhraní i tady. Konfigurace tedy vypadá takto:

```
SSLCipherSuite    TLSv1.3   TLS_AES_256_GCM_SHA384
SSLCipherSuite    SSL       HIGH
```

Pomocí prvního řádku je možné nastavit šifrové sady z verze TLS 1.3 a pomocí druhého řádku ty starší.

S upravenou verzí knihovny opět stačí původní rozhraní. Stejná konfigurace jako u webového serveru Nginx:

```
SSLEngine on
SSLHonorCipherOrder on
SSLProtocol -all +TLSv1.3 +TLSv1.2
SSLCipherSuite 'HIGH|ALL'
```

## 5.8 Nástroj curl

Nástroj curl, dostupný z [21], je klientská aplikace určená k odeslání nebo přijetí dat ze serveru. Podporuje široké spektrum protokolů, například HTTPS, LDAPS nebo IMAPS. Dále podporuje různé kryptografické knihovny. Pokud máme knihovnu nainstalovanou, je možné tento nástroj stáhnout a zkompilovat následující posloupností příkazů:

```
git clone http://github.com/curl/curl.git
cd curl
./buildconf
./configure --with-ssl
make
sudo make install
```

Přesto, že podporuje různé kryptografické knihovny, nabízí k nastavení šifrových sad společné rozhraní `--ciphers`. V případě knihovny OpenSSL přes něj ale není možné zadávat šifrové sady z verze TLS 1.3. K tomu slouží rozhraní `--tls13-ciphers`, které funguje pouze pokud je curl zkompilován s touto knihovnou.

Pokud je zkompilován s upravenou knihovnou OpenSSL, je možné zadávat šifrové sady z různých verzí pomocí jednoho rozhraní. Stejně jako u předchozích programů, i zde byly vyzkoušeny stejné kombinace serverových a klientských šifrových seznamů z přílohy B se stejným výsledkem.

Se stávající verzí knihovny OpenSSL je pro nastavení šifrových sad ze všech verzí nutné použít oba přepínače:

```
curl --ciphers 'HIGH|ALL' \
--tls13-ciphers 'TLS_AES_256_GCM_SHA384:\
TLS_CHACHA20_POLY1305_SHA256:TLS_AES_128_GCM_SHA256:\
TLS_AES_128_CCM_8_SHA256:TLS_AES_128_CCM_SHA256'
```

S upravenou verzí pak stačí využít pouze jednoho rozhraní a využít šifrových aliasů:

```
curl --ciphers 'HIGH|ALL'
```

## 5.9 Emailový server

Zástupcem e-mailových serverů je Dovecot, dostupný z [22], který nabízí protokoly IMAP a POP3 včetně jejich šifrovaných alternativ (IMAPS a POP3S). Postup pro jeho stažení a zkompilování je následující:

```
wget https://dovecot.org/releases/2.3/dovecot-2.3.9.2.tar.gz
tar -xvf dovecot-2.3.9.2.tar.gz
cd dovecot-2.3.9.2
./configure --with-ssl=openssl
make
sudo make install
```

Dovecot dříve podporoval GnuTLS i OpenSSL, ale aktuálně funguje pouze s OpenSSL. Šifrové sady lze ovlivnit dvěma způsoby. Zadáním minimální povolené verze protokolu `ssl_min_protocol` a nebo specifikováním šifrových sad pomocí `ssl_cipher_list`, kde je možné použit šifrový řetězec používaný v rozhraní `cipher_list` v knihovně OpenSSL. Seznam šifrových sad z protokolu TLS 1.3 není možné změnit, bude obsahovat pouze výchozí šifrové sady.

Díky upravené verzi knihovny pak je možné specifikovat šifrové sady ze všech verzí včetně nových funkcionalit. Konfigurace, která zapne preferenci serverových šifer, zakáže protokoly starší než TLS 1.2 a povolí šifrové sady z aliasu `HIGH` ze všech verzí, vypadá následovně:

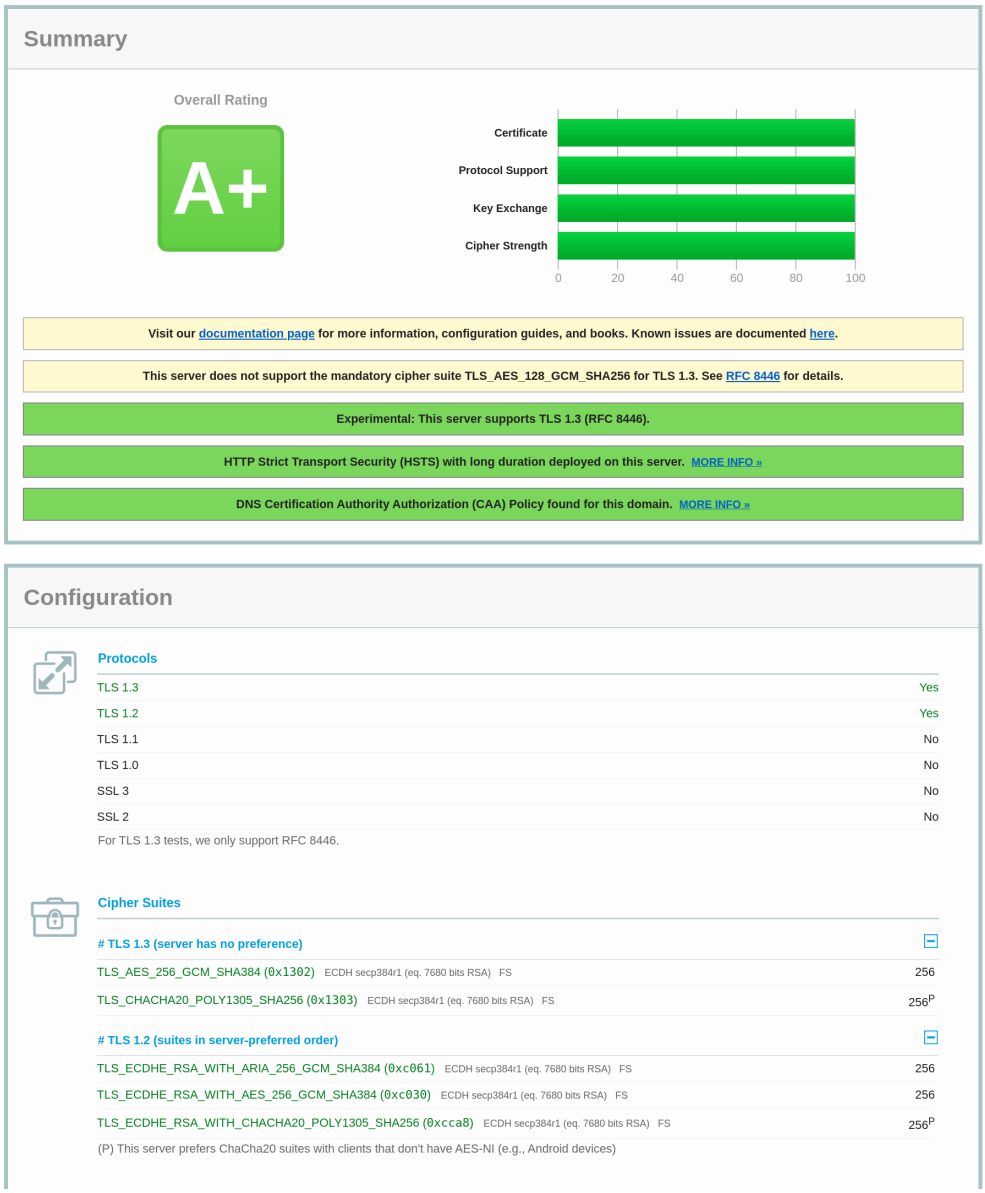
```
ssl_cipher_list='HIGH|ALL'
ssl_prefer_server_ciphers=yes
ssl_min_protocol=TLSv1.2
```

## 5.10 Hodnocení SSL Labs

Další možností, jak ověřit povolené šifrové sady a celkovou bezpečnost konfigurace, je webový nástroj, který nabízí bezplatně společnost Qualys, Inc. Na jejích stránkách `https://www.ssllabs.com/` je možné otestovat vlastní prohlížeč nebo zadat adresu webové stránky kterou chceme otestovat.

Pro dosažení hodnocení 100 % u síly šifer je potřeba povolit pouze ty, které mají velikost klíče alespoň 256 bitů a nepoužívají slabé algoritmy. Výsledky testů znázorňuje obrázek 5.1 a v jeho popisku se pak nachází šifrový seznam použitý při konfiguraci webového serveru. Pro názornost byl vybrán pouze vzorek šifrových sad. Za zmínku ale stojí detekce preference šifrových sad používajících algoritmus ChaCha20.

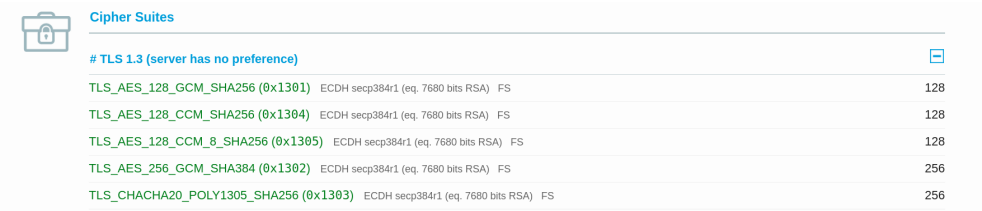
Pokud se rozhodneme použít skupiny o stejné preferenci, stránka to vyhodnotí jako preferenci klientova seznamu. V tomto případě také nedokáže identifikovat preferenci šifrové sady. Výstup testu je zachycen na obrázku 5.2 a v jeho popisku je znázorněn zadaný šifrový seznam včetně zadaných skupin a příznaků.




Obrázek 5.1: Hodnocení SSLlabs — síla šifrových sad

Zadaný šifrový seznam:

`TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256:`
`ECDHE-ARIA256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:`
`ECDHE-RSA-CHACHA20-POLY1305:+CHACHA20|ALL:*CHACHA20|ALL`




Obrázek 5.2: Hodnocení SSLlabs — skupiny o stejné preferenci

```
  TLS_AES_256_GCM_SHA384          TLSv1.3 Kx=any      Au=any
  TLS_AES_128_CCM_8_SHA256        TLSv1.3 Kx=any      Au=any
  TLS_AES_128_CCM_SHA256          TLSv1.3 Kx=any      Au=any
* TLS_CHACHA20_POLY1305_SHA256    TLSv1.3 Kx=any      Au=any
  TLS_AES_128_GCM_SHA256          TLSv1.3 Kx=any      Au=any
```

# Závěr

Cílem této diplomové práce bylo prostudovat problematiku protokolů SSL/TLS se zaměřením na různé verze protokolů v jednotlivých kryptografických knihovnách a následně navrhnout a implementovat změny v knihovně OpenSSL. Prvním krokem bylo seznámení se s problematikou a samotnými protokoly. Dále byly prozkoumány existující kryptografické knihovny, zejména podpora protokolů a princip specifikace šifrových sad. Na základě těchto zjištěných nedostatků byly navrženy úpravy a jejich následná implementace v knihovně OpenSSL. Poslední částí bylo testování těchto úprav. Důraz byl kladen především na minimum změn a zachování zpětné kompatibility.

Při testování navázání spojení se upravená verze knihovny chovala dle očekávání. Vliv na výkon byl velmi malý a v porovnání s režií operačního systému a samotného přenosu dat ho lze zanedbat. Dále s touto knihovnou byly zkompilovány vybrané aplikace třetích stran (Nginx, Apache, curl a Dovecot) a všechny dokázaly využít vylepšené rozhraní knihovny pro zadávání šifrových sad.

Přestože některé aplikace se již vypořádaly se dvěma rozhraními pro zadávání šifrových sad, najde tato knihovna uplatnění i u nich. Upravené rozhraní pro zadávání šifrových sad přináší nejenom zjednodušení práce správcům serverů, ale také přináší více možností nastavení pro výběr šifrové sady. U aplikací, které nyní podporují pouze jedno rozhraní, je přínos ještě větší, tato verze jim totiž umožní modifikovat šifrové sady z verze TLS 1.3.

Knihovna s upraveným rozhraním pro zadávání šifrových sad je zveřejněna v repozitáři[12] ve větvi `Improvement-of-the-ciphersuite-selection` na stránkách GitHub odkud je možné ji také stáhnout a zkompilovat.

V době psaní této práce byl na stránkách GitHub v projektu OpenSSL otevřen pull request[13] ve kterém se řeší přidání těchto úprav do hlavní větve. Pokud by k tomu došlo, upravená verze by se časem dostala do repozitářů

[12] `https://github.com/OttoHollmann/openssl/tree/Improvement-of-the-ciphersuite-selection`

[13] `https://github.com/openssl/openssl/pull/10590`

linuxových distribucí a aplikací třetích stran. Takto vylepšeného rozhraní by pak mohli využívat i běžní uživatelé bez nutnosti kompilovat celou knihovnu nebo aplikace, které ji využívají.

# Literatura


[1] Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, RFC Editor, August 2018, [cit. 2019-05-07]. Dostupné z: https://www.rfc-editor.org/rfc/rfc8446.txt

[2] Dierks, T.; Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, RFC Editor, August 2008, [cit. 2019-09-25]. Dostupné z: https://www.rfc-editor.org/rfc/rfc5246.txt

[3] Krawczyk, H.; Bellare, M.; Canetti, R.: HMAC: Keyed-Hashing for Message Authentication. RFC 2104, RFC Editor, February 1997, [cit. 2019-10-17]. Dostupné z: https://www.rfc-editor.org/rfc/rfc2104.txt

[4] Krawczyk, H.; Eronen, P.: HMAC-based Extract-and-Expand Key Derivation Function (HKDF). RFC 5869, RFC Editor, May 2010, [cit. 2019-06-24]. Dostupné z: https://www.rfc-editor.org/rfc/rfc5869.txt

[5] Barker, E.: NIST Special Publication 800-57: Recommendation for Key Management. January 2016, [cit. 2019-06-10]. Dostupné z: https://dx.doi.org/10.6028/NIST.SP.800-57pt1r4

[6] *OpenSSL, Release Strategy.* [online] [cit. 2019-05-21]. Dostupné z: https://www.openssl.org/policies/releasestrat.html

[7] *wolfSSL Embedded SSL/TLS Library.* [online] [cit. 2019-05-25]. Dostupné z: https://www.wolfssl.com/products/wolfssl/

[8] *BoringSSL.* [online] [cit. 2019-06-19]. Dostupné z: https://opensource.google.com/projects/boringssl

[9] *About us — mbed TLS (Previously PolarSSL).* [online] [cit. 2019-04-20]. Dostupné z: https://tls.mbed.org/about-us

[10] *TLS Toolkit 4.0, a complete and compact TLS implementation supporting TLS 1.3*. [online] [cit. 2019-06-06]. Dostupné z: https://www.insidesecure.com/Products/Data-Communication/Secure-Communication-Toolkits/Inside-Secure-TLS-Toolkit

[11] *Network Security Services — Mozilla*. [online] [cit. 2019-06-10]. Dostupné z: https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS

[12] *JSSE Reference Guide*. [online] [cit. 2019-06-14]. Dostupné z: https://docs.oracle.com/javase/8/docs/technotes/guides/security/jsse/JSSERefGuide.html#Introduction

[13] *The GnuTLS Transport Layer Security Library*. [online] [cit. 2019-06-15]. Dostupné z: https://gnutls.org/

[14] *The Legion of the Bouncy Castle*. [online] [cit. 2019-06-16]. Dostupné z: https://bouncycastle.org/

[15] Birr-Pixton, J.: *Crate rustls*. [online] [cit. 2019-06-16]. Dostupné z: https://docs.rs/rustls/0.13.1/rustls/

[16] *COVERITY SCAN STATIC ANALYSIS*. [online] [cit. 2019-12-05]. Dostupné z: https://scan.coverity.com/

[17] *Common Weakness Enumeration, A Community-Developed List of Software Weakness Types*. [online] [cit. 2019-12-05]. Dostupné z: https://cwe.mitre.org/index.html

[18] *Valgrind*. [online] [cit. 2019-12-06]. Dostupné z: https://valgrind.org/

[19] *nginx*. [online] [cit. 2019-12-07]. Dostupné z: https://nginx.org/en/

[20] *The Apache HTTP Server Project*. [online] [cit. 2019-12-07]. Dostupné z: https://httpd.apache.org/

[21] *curl*. [online] [cit. 2019-12-07]. Dostupné z: https://curl.haxx.se/

[22] *Dovecot*. [online] [cit. 2019-12-16]. Dostupné z: https://www.dovecot.org/

Příloha A

# Seznam použitých zkratek

**ABI** Application Binary Interface

**AEAD** Authenticated Encryption with Associated Data

**API** Application Programming Interface

**CWE** Common Weakness Enumeration

**DTLS** Datagram Transport Layer Security

**FIPS** Federal Information Processing Standards

**GPL** GNU General Public License

**HKDF** HMAC-based Extract-and-Expand Key Derivation Function

**HMAC** Keyed-hash Message Authentication Code

**HTTPS** Hypertext Transfer Protocol Secure

**IETF** Internet Engineering Task Force

**IMAP** Internet Message Access Protocol

**IMAPS** IMAP Secure

**ISC** Internet Systems Consortium

**LDAPS** Lightweight Directory Access Protocol Secure

**LGPL** GNU Lesser General Public License

**MAC** Message Authentication Code

**MIT** Massachusetts Institute of Technology

**MPL** Mozilla Public License

**NIST** National Institute of Standards and Technology

**POP3** Post Office Protocol version 3

**POP3S** POP3 Secure

**RFC** Request for Comments

**SSL** Secure Socket Layer

**TCP** Transmission Control Protocol

**TLS** Transport Layer Security

**UDP** User Datagram Protocol

Příloha B

# Výběr šifrové sady

Tato příloha popisuje různá nastavení serveru a klienta použitých při testování a pro každou kombinaci popisuje výběr konkrétní šifrové sady. Vzhledem k odlišnostem mezi verzemi protokolů jsou nejprve uvedeny kombinace šifrových sad pro verzi TLS 1.3 a v druhé části pro starší verze.

## B.1 Výběr šifrových sad ve verzi TLS 1.3

Bude-li mít server nastavenou masku verzí na hodnotu `ALL` a následující šifrový řetězec `[AESGCM]:[AESCCM]:[CHACHA20:*ALL]` bude jeho šifrový seznam včetně příznaků a skupin, vypadat takto (budeme-li uvažovat pouze šifrové sady z verze TLS 1.3):

```
[   TLS_AES_256_GCM_SHA384
[   TLS_AES_128_GCM_SHA256
[   TLS_AES_128_CCM_8_SHA256
    TLS_AES_128_CCM_SHA256
  * TLS_CHACHA20_POLY1305_SHA256
```

Dále následují různé seznamy šifrových sad použité u klienta. Červeně označené šifrové sady budou vybrány pro komunikaci.

### B.1.1 Konfigurace klienta

```
TLS_CHACHA20_POLY1305_SHA256
TLS_AES_256_GCM_SHA384
TLS_AES_128_GCM_SHA256
TLS_AES_128_CCM_8_SHA256
TLS_AES_128_CCM_SHA256
```

Bude vybrána šifrová sada TLS_CHACHA20_POLY1305_SHA256, protože v seznamu klienta je na prvním místě a server má u této sady nastavený příznak preference.

### B.1.2 Konfigurace klienta

TLS_AES_128_GCM_SHA256
TLS_AES_256_GCM_SHA384
TLS_AES_128_CCM_8_SHA256
TLS_AES_128_CCM_SHA256
TLS_CHACHA20_POLY1305_SHA256

Bude vybrána šifrová sada TLS_AES_128_GCM_SHA256, protože ze dvou šifrových sad v první skupině o stejné preferenci se tato nachází v klientově seznamu výš.

### B.1.3 Konfigurace klienta

TLS_AES_128_CCM_8_SHA256
TLS_AES_128_CCM_SHA256
TLS_AES_256_GCM_SHA384
TLS_AES_128_GCM_SHA256
TLS_CHACHA20_POLY1305_SHA256

Bude vybrána šifrová sada TLS_AES_256_GCM_SHA384, protože ze dvou šifrových sad v první skupině o stejné preferenci se tato nachází v klientově seznamu výš. Šifrové sady z aliasu AESCCM jsou sice v seznamu klienta výš, ale v serverovém seznamu se nacházejí až ve druhé skupině o stejné preferenci.

### B.1.4 Konfigurace klienta

TLS_AES_128_CCM_8_SHA256
TLS_CHACHA20_POLY1305_SHA256
TLS_AES_128_CCM_SHA256

Z první skupiny o stejné preferenci není žádná společná šifrová sada. Proto dojde k vybrání sady TLS_AES_128_CCM_8_SHA256, která se nachází až ve druhé skupině.

### B.1.5 Konfigurace klienta

TLS_AES_128_CCM_SHA256
TLS_AES_128_CCM_8_SHA256

Podobná situace jako v předchozím příkladu, pouze je v seznamu klienta výš šifrová sada TLS_AES_128_CCM_SHA256 a proto bude vybrána.

## B.2 Výběr šifrových sad ve verzích do TLS 1.2

Pro testování starších šifrových sad bude mít server nastaven šifrový řetězec: `[AES+RSA]:[AESCCM+DH]:[CHACHA20:*ALL]:CAMELLIA256-SHA256` a po vynechání šifrových sad, které jsou ve výchozím nastavení zakázány při kompilaci, jeho šifrový seznam bude vypadat následovně:

```
   AES256-GCM-SHA384
   AES256-CCM8
   AES256-CCM
   AES128-GCM-SHA256
   AES128-CCM8
   AES128-CCM
   AES256-SHA256
   AES128-SHA256
   DHE-RSA-AES256-CCM8
   DHE-RSA-AES256-CCM
   DHE-RSA-AES128-CCM8
   DHE-RSA-AES128-CCM
* ECDHE-ECDSA-CHACHA20-POLY1305
* ECDHE-RSA-CHACHA20-POLY1305
* DHE-RSA-CHACHA20-POLY1305
   CAMELLIA256-SHA256
```

Opět následují různé seznamy šifrových sad použité u klienta. Červeně označené šifrové sady budou vybrány pro komunikaci.

### B.2.1 Konfigurace klienta

```
ECDHE-ECDSA-AES128-CCM
ECDHE-RSA-CHACHA20-POLY1305
AES256-CCM
AES128-CCM
DHE-RSA-AES128-CCM8
DHE-RSA-AES128-CCM
```

Šifrová sada `ADH-AES256-SHA256` není na serveru povolena, proto první společnou šifrovou sadou je sada `ECDHE-RSA-CHACHA20-POLY1305`. Protože má v serverovém seznamu nastavený příznak preference a z pohledu klienta je první společná, bude vybrána.

### B.2.2 Konfigurace klienta

```
DHE-RSA-AES128-CCM8
DHE-PSK-CHACHA20-POLY1305
ECDHE-ECDSA-CHACHA20-POLY1305
AES256-CCM
AES128-CCM
```

V tomto případě je první společnou šifrovou sadou `DHE-RSA-AES128-CCM8`, která nemá nastavený příznak preference, takže server nejprve vyzkouší všechny šifrové sady z první skupiny (v tomto případě ty, které používají AES). V této skupině se nachází hned dvě šifrové sady, které podporuje i klient. Sada `AES256-CCM` je ale v klientově seznamu výše a proto bude vybrána.

### B.2.3 Konfigurace klienta

Skupiny o stejné preferenci nebo příznaky preference lze zadat i v klientově seznamu. V takovém případě budou mít vliv pouze při vytváření seznamu (při případném posouvání šifrových sad na začátek nebo na konec), ale samotné navazování spojení nijak neovlivní. Pokud bude mít klient nastaven řetězec `[AESCCM+DH|TLSv1.2]:CAMELLIA256-SHA256:CHACHA20+ECDH:^ALL`, jeho seznam bude vypadat takto:

```
CAMELLIA256-SHA256
ECDHE-ECDSA-CHACHA20-POLY1305
ECDHE-RSA-CHACHA20-POLY1305
DHE-RSA-AES256-CCM8
DHE-RSA-AES256-CCM
DHE-RSA-AES128-CCM8
DHE-RSA-AES128-CCM
```

Jak již bylo psáno výše, při navazování spojení se skupiny a příznaky klienta ignorují, protože za výběr šifrové sady zodpovídá server a ten dostane pouze seznam šifrových sad bez příznaků. Vybrána bude šifrová sada `DHE-RSA-AES256-CCM8`.

Příloha C

# Měření výkonu

Výstup nástroje `s_time` s původní verzí knihovny:

```
Collecting connection statistics for 30 seconds

3164 connections in 2.64s; 1198.48 connections/user sec, bytes read 0
3164 connections in 31 real seconds, 0 bytes read per connection

Now timing with session id reuse.
starting

3341 connections in 2.79s; 1197.49 connections/user sec, bytes read 0
3341 connections in 31 real seconds, 0 bytes read per connection
```

Výstup nástroje `s_time` s upravenou verzí knihovny:

```
Collecting connection statistics for 30 seconds

3205 connections in 2.71s; 1182.66 connections/user sec, bytes read 0
3205 connections in 31 real seconds, 0 bytes read per connection

Now timing with session id reuse.
starting

3290 connections in 2.77s; 1187.73 connections/user sec, bytes read 0
3290 connections in 31 real seconds, 0 bytes read per connection
```

Příloha D

# Obsah přiloženého CD

- `README.txt` ..... stručný popis obsahu CD
- `openssl-3.0.0-src.tar.gz` ..... původní archiv OpenSSL
- `openssl-3.0.0.tar.gz` ..... archiv OpenSSL s implementací
- `text` ..... text práce
  - `DP_Hollmann_Otto_2020.pdf` ..... text práce ve formátu PDF
  - `src` ..... zdrojová forma práce ve formátu LaTeX