Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

# Bakalářská práce

# KIVFS
# Zabezpečení serveru

Plzeň 2014 Karel Vlček

# Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 2. července 2014

Karel Vlček

# Abstract

The thesis is about designing and creatig security model for KIVFS distributed file system. It starts with explanation what the distributed systems are and the KIVFS is described there. The thesis continues with informations about security algorithms used for encryption and security protokols used for negotation of identity. There is presented security model and it is described how it is implemented into KIVFS.

# Obsah

**1 Úvod** **1**

**2 Teoretická část** **2**
2.1 Centralizované systémy . . . . . 2
2.2 Distribuovaný systém . . . . . 3
2.2.1 Sdílení prostředků, škálování . . . . . 4
2.2.2 Transparentní přístup . . . . . 4
2.2.3 Konkurenční přístup a synchronizace . . . . . 5
2.3 DFS . . . . . 6
2.3.1 Bezpečnost . . . . . 7
2.3.2 Dostupná řešení . . . . . 7
2.4 KIVFS . . . . . 8
2.4.1 Struktura KIVFS . . . . . 9
2.4.2 Komunikační protokol . . . . . 10
2.4.3 Bezpečnost - aktuální slabá místa . . . . . 11
2.5 Šifrování komunikace . . . . . 11
2.5.1 Symetrické šifrování . . . . . 12
2.5.2 Asymetrické šifrování . . . . . 16
2.5.3 Jednocestné funce - HASH . . . . . 17
2.5.4 Existující knihovny a řešení - Protokol SSL . . . . . 19
2.6 Ověření uživatele . . . . . 21
2.6.1 Kerberos . . . . . 22
2.7 Pověření uživatele . . . . . 26

**3 Praktická část** **28**
3.1 Ověření . . . . . 28
3.2 Zabezpečení přenosu . . . . . 29
3.2.1 Kód . . . . . 31
3.3 Ověření uživatele . . . . . 33
3.3.1 Instalace . . . . . 33
3.3.2 Nastavení konfiguračních souborů . . . . . 33

3.3.3 Nastavení databáze KDC . . . . . 34
3.3.4 Kód . . . . . 36
3.4 Oprávnění uživatele . . . . . 38
3.4.1 Implementace ACL v bezpečnostní vrstvě . . . . . 41
3.5 Překlad a spuštění . . . . . 43
3.5.1 Překlad . . . . . 44
3.5.2 Spuštění . . . . . 45
3.6 Ověření výkonu . . . . . 46

**4 Závěr** **49**

# 1 Úvod

Nacházíme se v době velkého rozmachu internetu a s tím souvisejícího růstu množství dat. Data je třeba uchovávat na serverech s velkou úložnou kapacitou, která na osobních počítačích a zvláště pak na mobilních přístrojích chybí. S potřebou přistupovat k velkým objemům dat pomocí mobilních stanic přes internet roste riziko zcizení těchto dat během přenosu nebo zcizení přístupu k nim.

Ideálním řešením v obecné síti může být DFS (distribuovaný souborový systém). DFS je systém pro uchovávání dat, který nejenom zabezpečuje data samotná, ale i přístup k nim.

Na katedře informatiky Západočeské univerzity v Plzni je vyvíjen distribuovaný souborový systém KIVFS. Úkolem mé práce je zabezpečení KIVFS.

# 2 Teoretická část

V teoretické části jsou vysvětleny pojmy centralizovaný a distribuovaný systém. Jsou zde vyjmenovány funkce distribuovaného souborového systému, který je speciálním případem distribuovaného systému. Je zde popsán distribuovaný souborový systém KIVFS. V dalších kapitolách je řešena bezpečnost. Jsou zde popsány symetrické šifry, asymetrické šifry a hashovací funkce a nakonec bezpečnostní protokoly SSL a Kerberos.

## 2.1 Centralizované systémy

Přístup k systému je poskytován jedním přístupovým bodem v síti (dále uzlem). Systém, ke kterému uzel zprostředkovává přístup je považován za jeho součást. Jednoduchost systému vyplývající ze správy jednoho uzlu má následující výhody:

- Snadné nasazení
- Jednoduchá správa a údržba

Nevýhodou použití jednoho uzlu je tzv. úzké hrdlo, které má tyto vlastnosti:

- Omezené škálovaní uzlu zahrnující přidávání nebo záměnu hardwarových komponent v serverech vedoucí ke zvýšení výkonnosti (škálování do výšky).
- Počáteční náklady mohou být vysoké, pokud budeme chtít nakoupit hardware, který bude poskytovat dostatečnou kapacitu po co nejdelší období.
- V případě výpadku datového spojení nebo celého uzlu jsou nedostupné všechny prostředky.
- Po úspěšném napadení uzlu má pachatel přístup ke všem prostředkům.

## 2.2 Distribuovaný systém

Definice: "Distribuovaný systém (dále DS) je kolekce nezávislých entit, která spolupracuje k řešení problémů, které nemohou být vyřešeny samostatně." [13] Distribuovaný systém lze charakterizovat jako soubor samostatných výpočetních prostředků, které jsou propojeny počítačovou sítí, pracujících za pomoci specializovaného software jako jednotný systém. Vlastnosti distribuovaných systémů:

- **Neobsahují jedny sdílené fyzické hodiny** Každý uzel v DS obsahuje vlastní lokální hodiny. Je třeba zajistit synchronizaci lokálních hodin na každém uzlu například pomocí NTP (Network Time Protocol).
- **Neobsahují sdílenou paměť'** Jednotlivé části spolu komunikují pomocí zpráv, kterými lze distribuovanou sdílenou paměť' realizovat, ale není to sdílená paměť', jak ji chápeme například z multiprocesorových architektur (distribuovaná paměť').
- **Geografická oddělenost** DS nemusí být celý umístěn v jedné lokalitě. Jeho části se mohou nacházet v různých geografických lokalitách s různými možnostmi připojení počítačové sítě. Nelze se proto spoléhat na kvalitu připojení a dostupnost uzlu. Může docházet ke zpožděním.
- **Nezávislé a různorodé prostředí** Prvky DS jsou samostatné jednotky, které mohou v systému vznikat, zanikat nebo se přeskupovat. DS může být složen z počítačů různých vlastností (architektura, výkonnost, operační systém) a musí být navržen tak, aby tyto vlastnosti bral v úvahu.

DS jsou konstruovány za účelem sdílení prostředků. K prostředkům je transparentní přístup (stejný jako k lokálním). Je řešen konkurenční přístup a škálování. Reálné systémy se zaměřují jen na vybrané vlastnosti na úkor jiných, protože se systém navrhuje s ohledem na poskytování služeb daných parametrů.

### 2.2.1 Sdílení prostředků, škálování

V současné době, kdy v prostředí internetu může přistupovat k jednomu prostředku nebo službě najednou tisíce uživatelů, výkon a kapacita jediného počítače i přes Mooreův zákon [5] nedostačuje. Sdílením prostředků mezi servery, které pro uživatele díky transparentnosti vystupují jako jeden systém, lze dosáhnout vyšší dostupnosti služeb a prostředků. Vyšší dostupnosti lze také dosáhnout přidáváním dalších uzlů (škálováním systému do šířky).

### 2.2.2 Transparentní přístup

Transparentní přístup znamená, že uživatel nebo služba přistupuje k DS jako, kdyby se jednalo o lokální systém. DS odděluje vlastnosti vnějšího světa od vnitřní architektury systému a vytváří dojem, že poskytuje služby centralizovaně. ANSA (Advanced Networked Systems Architecture) [2] popisuje oblasti, které je při návrhu DS třeba řešit. Transparentnost ideálního DS se dělí na následující oblasti:

- **Přístupová transparentnost** Se službou se pracuje, jako by byla lokální.
- **Místní transparentnost** Prostředky jsou k dispozici stejně, bez ohledu na to, kde se nachází poskytovatel nebo uživatel.
- **Migrační nebo relokační transparentnost** Změny struktury v DS, přidání prostředků, ubírání nebo přesun jsou uživateli skryty.
- **Transparentnost selhání a perzistence** Částečná selhání systému nesmí omezit uživatele v práci. Data má stále k dispozici. Po výpadku jsou data stále neporušená a beze ztrát.
- **Konkurenční a transakční transparentnost** Činnost ostatních uživatelů nesmí mít vliv na přístup k prostředkům daného uživatele. Konkurenční přístup je řešen pomocí transakcí splňujících požadavky na bezpečný transakční přístup: A - atomičnost, C - konzistence, I - izolace, D - trvanlivost.
- **Transparentnost replik** Uživatel nepozná, zda pracuje s replikou nebo s originálním prostředkem a zda je replika synchronní s původním prostředkem.

- **Bezpečnostní transparentnost** Správa přístupů a zabezpečení dat jsou jednotné.

### 2.2.3 Konkurenční přístup a synchronizace

Distribuované systémy mají za úkol poskytovat sdílené zdroje. U těchto zdrojů dochází k opakovaným konkurenčním přístupům, jak pro čtení dat, tak pro jejich zápis. Čtení data neovlivňuje, ale zápis je problém. O sdílené prostředky musí uzly DS nejprve požádat. Aby bylo možné určit, který uzel dostane zdroj přidělen, je nutné mít informaci o pořadí požadavků.

V DS se může pohybovat počet uzlů v řádu desítek až tisíců, jejich lokální hodiny jsou různě přesné a zpoždění při komunikaci není konstantní ani nulové. Pro synchronizaci a určování pořadí je nutné zavést logické hodiny a další synchronizační algoritmy (Lamport timestamps, Vector clock...).

Data jsou za účelem větší propustnosti a stability systému replikována. Replikovaná data musejí být konzistentní. To znamená, že nelze nabízet různá data pro stejný požadavek. Musí existovat způsob jak změny replikovaných dat propagovat. Změny se musí provést ve stejném pořadí na všech replikách. Modely konzistence rozlišujeme podle způsobu propagace dat a podle pohledu na systém (pro kterého aktéra v systému musí být pohled na data konzistentní). Z pohledu dat rozlišujeme modely konzistence:

- Striktní konzistence je nejsilnější konzistence. Všechny změny musí být zapsány všude v jeden čas a ve stejném pořadí (v případě chyby nikde). Čtecí operace na každém uzlu vrací vždy aktuální informace. Jakékoli čtení dat uživatelem vrací vždy stejnou hodnotu odpovídající poslednímu zápisu.
- Sekvenční konzistence řadí operace na všech uzlech ve stejném pořadí. Může vznikat zpoždění (oblast ne konzistence).
- Příčinná konzistence řadí operace v pořadí, v jakém na sebe vzájemně působí. Pokud se operace neovlivňují, není jejích pořadí zaručeno. Je třeba udržovat graf závislých operací.
- FIFO konzistence vidí zápisy každého uzlu v pořadí, v jakém byly prováděny, ale vzájemně mezi uzly pořadí není určeno. Je třeba číslovat operace z každého zdroje a provádět je ve stejném pořadí.

- Slabá konzistence využívá synchronizační operace, které se provádí sekvenčně a jsou řízené logickými hodinami nebo dalšími synchronizačními algoritmy. Změny se propagují až jako výsledek poslední operace.
- Uvolňovací konzistence zavádí kritickou sekci. Změny jsou propagovány až po opuštění kritické sekce (všechny operace zápisu musejí být ukončené). Do kritické sekce se vstupuje FIFO a pracuje se s ní stejně jako se sdílenou proměnnou při paralelním programování.
- Přístupová konzistence je podobná uvolňovací. Operace zápisu do sdílených proměnných musejí být dokončeny před získáním zámku pro výhradní přístup dalším procesem.

## 2.3 DFS

Zvláštní případ DS je distribuovaný souborový systém (dále DFS). Nabízí vzdálený přístup k souborům. Soubory mohou být fyzicky rozmístěny po různých uzlech. Uživatel tuto skutečnost nepozná, protože DFS soubory nabízí transparentně jako jeden zdroj (adresářový strom). Operace, které mohou v rámci uzlů probíhat jsou: migrace, replikace a zálohování dat. Pomocí databáze, která obsahuje strukturu a umístění dat, jsou řízeny další procesy a práce s daty. Tato databáze může být centralizovaná s replikami nebo distribuovaná. Mezi uzly DFS probíhá synchronizace replik souborů a požadavků systému.

DFS disponuje výše vyjmenovanými vlastnostmi distribuovaných systémů. Navíc se zaměřuje na funkce umožňující:

- Ukládat informace na datová úložiště.
- Prezentovat uložené informace v podobě souborů.
- Organizovat data do stromové struktury v podobě adresářů.
- Manipulovat s daty pomocí sady příkazů (operací).
- Zajištění bezpečnosti dat.

### 2.3.1 Bezpečnost

Nedílnou součástí systému zaměřeného na správu a poskytování dat je ochrana dat. V DFS jsou data chráněna omezením přístupu uživatelů k datům. Toho lze docílit pomocí následujícího modelu:

- **Ověřováním uživatelů.** Po navázání spojení se musí uživatel autentizovat. Po úspěšném ověření identity je uživateli povolen přístup k systému.
- **Šifrováním komunikace mezi serverem a klientem.** Šifrované spojení zajistí přenášená data před odcizením a nebo jejich poškozením.
- **Používáním přístupových práv k souborů a adresářům.** Pomocí seznamu přístupových práva (ACL) lze ke každému souboru přiřadit uživatele nebo skupinu a jejich práva pro manipulaci se souborem. Mezi základní práva patří: právo pro čtení a zápis nebo právo přidělovat práva.
- **Šifrováním uložených dat.** Pomocí šifrování dat uložených na serveru lze obsah ochránit před fyzickým útokem (krádež serveru).

### 2.3.2 Dostupná řešení

Uvedené vlastnosti zavádí hodně systémů, ale ne dokonale. Mezi nejpoužívanější řešení patří:

- **OpenAFS - Andrew File System** [6] Nastavený model konzistence je slabá konzistence. Replikace používá pouze Master write replikaci, neumí Multi Master replikaci. Všechny repliky jsou pouze pro čtení a jen na požádání. Pro komunikaci používá protokol UDP a má problémy s přístupem klientů přes NAT(Network Address Translation). Pro zabezpečení využívá protokolu Kerberos a ACL.
- **CODA - Constant Data Availability** [3] Používá optimistický model replikace a R/W replikační servery. Podporuje offline operace pro klientské stanice. Data jsou uchovávána v klientské cache.

Aktuální stav dat je udržován pomocí callbacků. Změny zpět na server jsou propagovány pomocí reintegračních požadavků a jsou zpracovávány přes CML (Client Modification Log). Pro autentizaci uživatelů používá vlastní protokol podobný protokolu Kerberos a pro autorizaci využívá ACL.

- **Ceph** [1] Ověřování uživatelů a šifrování přenosu dat je v systému Ceph založeno na vlastním protokolu cephx podobnému protokolu Kerberos. Jako třetí důvěryhodná strana vystupuje Metadata Server a služba, ke které se přistupuje jsou Object Storage Devices. Pro oprávnění uživatelů používá Ceph "capabilities" (caps).

  Pro ukládání dat používá algoritmus CRUSH (Controlled Replication Under Scalable Hashing), který rozmist'uje data pseudonáhodně.

## 2.4 KIVFS

Současné DFS mají řadu nedostatků: Neúplná implementace funkcí (například neúplná transparentnost, chybějící online synchronizace), omezené možnosti nasazení za serverem / routerem používajícím NAT (Network address translation). Uživatelé jsou nespokojení s malou podporou mobilních zařízení (telefony, tablety), složitou rozšiřitelností (např. nepřehledné zdrojové kódy) a licenčními omezeními (patenty, nekompatibilita s GNU/GPL).

Distribuovaný souborový systém KIVFS vzniká jako odpověď na zmíněné nedostatky. KIVFS je vyvíjen na Katedře informatiky a výpočetní techniky Západočeské univerzity v Plzni. Je programován od naprostých základů s cílem umožnit dostupnost dat na nejrozšířenějších desktopových a mobilních platformách.

KIVFS má v budoucnu nahradit stávající distribuovaný souborový systém OpenASF na katedře informatiky a výpočetní techniky.

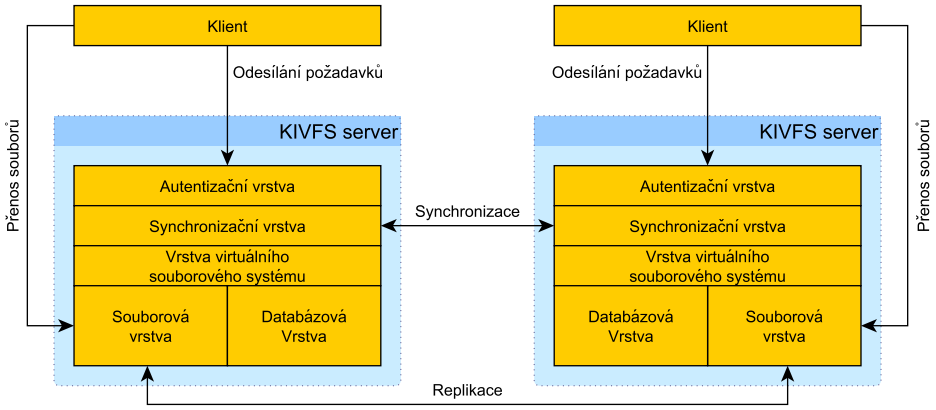
Základní cíle KIVFS jsou:

- Online Multi Master RW replikace (dostupnost všech replik pro čtení i zápis, replikace probíhá okamžitě po zápisu dat).
- Vysoká škálovatelnost, výkonnost, stabilita a bezpečnost.

- Podpora klientské cache a offline operací.
- Podpora mobilních platforem.
- Modulárnost (snadná rozšiřitelnost a modifikovatelnost).

### 2.4.1 Struktura KIVFS

KIVFS se skládá z různých skupin serverů rozdělených do vrstev. Jednotlivé vrstvy mají své specifické funkce. Struktura KIVFS a jednotlivé komponenty jsou znázorněny na obrázku 2.1. Z důvodu snadnějšího zavedení systému bylo přistoupeno k rozdělení funkcí serverů. Každý server je vyvíjen nezávisle na ostatních.




Obrázek 2.1: Schéma vrstev KIVFS

Servery mezi sebou komunikují vlastním protokolem pomocí síťových propojení nad protokolem TCP/IP. Servery jsou naprogramovány v jazyce C. Nezávislost protokolu umožňuje implementovat jednotlivé části KIVFS i v jiných programovacích jazycích. Například klientské aplikace na mobilních platformách jsou realizovány v jazycích: C#, Java a Objective C.

Základní platformou, pro kterou jsou servery naprogramovány je distribuce Debian operačního systému GNU/Linux. S menšími úpravami je možné nasadit servery i na jiné distribuce Unixových systémů.

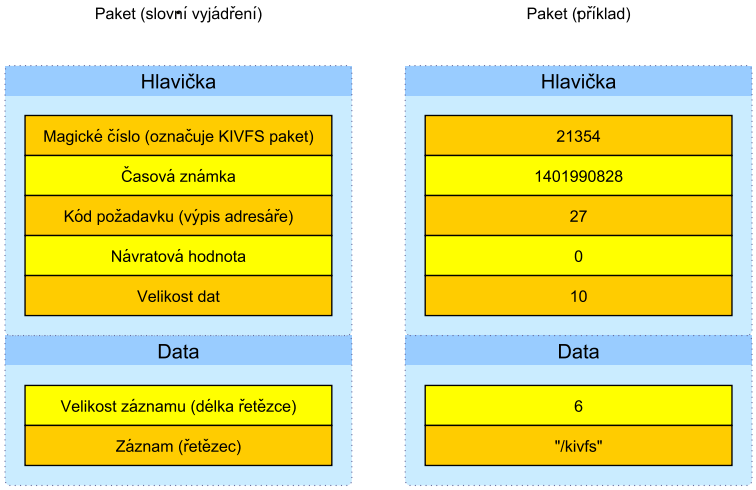
Funkce jednotlivých vrstev:

- **Autentizační vrstva** autentizuje klienta a zabezpečuje komunikace s klientskými aplikacemi. Klienti jsou autentizováni systémem Kerberos a vlastní databází uživatelů. Důvodem je možnost omezit přístup uživatelům majícím účet v systému Kerberos, ale nemajícím přístup do KIVFS. Vrstvou jsou spravována sezení autentizovaných klientů. Spojení s klienty je zabezpečeno šifrováním pomocí protokolu SSL.
- **Synchronizační vrstva** má za úkol synchronizovat požadavky pro přístup k datům. Spravuje požadavky replikačního systému, obnovu uzlů po návratu do sítě. Sbírá informace o dostupnosti jednotlivých serverů a hledá nejkratší cesty podle daných metrik, které předává vrstvě virtuálního souborového systému.
- **Vrstva virtuálního souborového systému** je abstrakcí souborového systému. Jejím úkolem je přerozdělovat požadavky mezi databázovou a souborovou vrstvu.
- **Databázová vrstva** využívá databázi MySQL, ale je možné použít i jiné. V databázi jsou uchovávány následující metadata:
  - Kompletní adresářovou strukturu
  - Atributy souborů
  - Mapování virtuální adresářové struktury na jednotlivé soubory
  - Informace o aktuálnosti a dostupnosti replik
  - Informace o svazcích, ze kterých se souborový systém skládá
  - Tabulky přístupových práv
- **Souborová vrstva** zajišťuje uložení dat na lokální souborový systém. Provádí replikace, deduplikace. Šifruje uložená data. Zprostředkovává přímý přístup pro uživatele a řídí přenos.

### 2.4.2 Komunikační protokol

Veškerá komunikace mezi jednotlivými komponenty probíhá prostřednictvím protokolu KIVFS. Protokol je postaven nad protokolem TCP/IP, který obsahuje mechanizmus pro potvrzování zpráv. Přenášené zprávy jsou binární

a jejich velikost optimalizovaná. Paket protokolu KIVFS je složen z hlavičky, která obsahuje informace o požadavku a těla obsahujícího data. 2.2




Obrázek 2.2: Obsah paketu KIVFS (výpis adresáře)

### 2.4.3 Bezpečnost - aktuální slabá místa

Bezpečnost KIVFS se opírá o stejný model popsaný v části o DFS. Na bezpečnost systému KIVFS nebyl kladen důraz. Bylo zapotřebí vyřešit jiné problémy zejména rychlost a stabilitu. Při implementaci bezpečnostních prvků dle zmíněného modelu nebyly splněny všechny požadavky. Slabá místa byla odhalena v částech šifrování přenosu dat mezi klientem a bezpečnostní vrstvou, autentizace uživatelů a služeb pomocí protokolu Kerberos a autorizování přístupů uživatelů v bezpečnostní vrstvě.

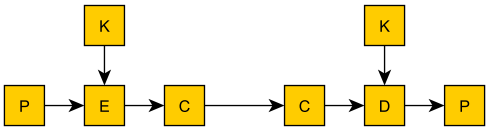
## 2.5 Šifrování komunikace

Spojení mezi serverem a klientem probíhá prostřednictvím obecné sítě, v níž se vyskytují další entity, které mohou komunikaci zachytávat nebo poškozovat. Cílem šifrování komunikace je zamezení odposlechu dat a tím zabránit omezení soukromí uživatele nebo zneužití citlivých informací a také zajištění zprávy proti změnám nebo alespoň ověření, že je zpráva nezměněná.

Šifrování vychází z použití šifrovacího klíče a funkce na otevřená data nebo text. Výsledkem šifrování je pro útočníka nečitelný kód, který je schopen rozluštit jen majitel klíče. Používá se buď jeden klíč pro obě operace (symetrické šifrování) nebo dva klíče, jeden pro šifrování a druhý pro dešifrování (asymetrické šifrování).

### 2.5.1 Symetrické šifrování

Symetrické šifrování používá složitou šifrovací funkci složenou z jednodušších (permutace, substituce, XOR, sčítání a násobení modulo). Nedá se prolomit při znalosti libovolného množství zvoleného textu. Na obrázku 2.3 je znázorněno jak šifrování probíhá.




Obrázek 2.3: Schéma symetrického šifrování

Použité zkratky:

- E - šifrovací funkce (encrypt)
- D - dešifrovací funkce (decrypt)
- K - klíč (key)
- P - prostý text (plain text)
- C - zašifrovaný text (cipher text)

Formální zápis:
Šifrování: $C = E_K(P)$
Dešifrování: $P = D_K(C)$

Nevýhodou symetrického šifrování je transport (utajení) klíče. Všechny komunikující strany musejí znát klíč.

Blokové šifry mapují n-bitový prostý text na n-bitovou šifru za použití k–bitového klíče. Lze je chápat jako jednoduché substituční šifry pracující nad obrovskou abecedou. Každá šifra je soustavou bijekcí definující permutaci nad n-bitovými vektory. Z toho plyne, že je invertibilní. Klíč vybírá konkrétní bijekci.

### S-P síť

Bloková šifra substitučně permutační síť se skládá z několika za sebou zřetězených operací permutace a substituce označovaných jako produkční šifra. Jednotlivé bloky algoritmu jsou nazývány S-Box (substituce) a P-Box (permutace).

### Feinstelova síť

Je základem většiny moderních šifer. Je založena na několikanásobném opakování operace XOR. Blok textu (P) je rozdělen na dvě poloviny($P = (L_0, R_0)$). Klíč ($K$) je rozdělen na několik podklíčů podle počtu iterací algoritmu ($K = k_1, k_2 \dots k_n$). Je nutné, aby všechny klíče byly různé, jinak bude šifra slabá. Každá iterace je pak vypočítána: $L_i = R_{i-1}, R_i = L_{i-1} \oplus f(R_{i-1}, k_i)$. Po poslední iteraci se provede záměna $L_i$ a $R_i$.

Dešifrování je možné provést, i když funkce $f$ není invertovatelná. Klíče se použijí v opačném pořadí.

### DES

Základem data encription standardu je Feinstelova šíť o 16 iteracích a s podklíčem délky 48 bitů. Délka kódovaného bloku je 64 bitů. Klíč má délku 64 bitů (z toho je 56 efektivních a 8 se používá jako kontrolních). Dnes se dá prolomit útokem hrubou silou za méně než 24 hodin. Je považována za nespolehlivou.

Je nahrazena variantou TripleDES, která používá šifrování DES 3 krát za sebou s klíčem délky 112 bitů. První polovina klíče je použita pro první a poslední DES v sérii. Nebo s klíčem délky 168 bitů, které se říká paranoidní varianta (pro každý DES je použita třetina klíče).

### IDEA

je nástupcem DESu a je považována za nejlepší symetrický blokový algoritmus. International Data Encryption Algorithm je implementován v rámci protokolu SSL. Jeho použití je pouze pro nekomerční účely zdarma, neboť je patentován.

Šifruje po blocích velikosti 64 bitů a používá 128 bitový klíč. Je složen ze tří základních funkcí: XOR 16 bitových subbloků, modulární součet $(a + b) mod 2^{16}$ a modulární násobení $(a * b) mod 2^{16} + 1$.

IDEA je 2 krát rychlejší než DES a výrazně bezpečnější. V letech 1994-5 bylo nalezeno několik tříd slabých klíčů. Pravděpodobnost náhodné volby slabého klíče je zanedbatelná $P = 2^{-77}$. Je považována za bezpečnou šifru.

### Blowfish

je symetrická bloková šifra pracující s bloky délky 64 bitů. Klíč může nabývat délky 32 - 448 bitů. Vychází z Feinstelovi sítě s počtem iterací 16.

Algoritmus používá expanze klíče a vytvoření 18 podklíčů, které jsou uloženy v P-polích. Bloky vstupního textu jsou rozloženy do 4 S-Boxů po 8 bitech (levá a pravá část bloku). Návrh počítá s implementací na 32 bitových procesorech. Opět byly nalezeny skupiny slabých klíčů. Pravděpodobnost volby je $P = 2^{-14}$. Je považována za bezpečnou.

### Rijndael

je produkční bloková šifra s proměnnou délkou bloků i klíče (128, 192, 256 i 512 bitů). Počet iterací je odvozen od délky klíče.

Pro klíč (a blok) délky 128 má 10 iterací. Data i klíč se uloží do matice velikosti 4x4.

Algoritmus zjednodušeně:

- Začátek: Provede se XOR vstupu a podklíče.
- 10 iterací:

1. S-Box substituce - nelineární algoritmus
2. Permutace - cyklický posun řádků o 0, 1, 2 a 3 pozice
3. MixColumns - násobení sloupců konstantním polynomem $C(x) = 3x^3 + x^2 + x + 2$ modulo $x^4 + 1$
4. AddRoundKey - šifrovací funkce (XOR matice a podklíče)

- V poslední iteraci je vynechán krok 3.

Algoritmus byl podroben rozsáhlé analýze a zvolen jako nový standard AES (Advanced Encritpion Standard). V současnosti není známa jakákoli podstatná slabina.

### Režimy blokových šifer

- Nejjednodušším je elektronická kódová kniha (ECB (Electrinic Code Book)). Postupně zpracovává otevřený text blok po bloku. ECB s sebou přináší několik problémů. Stejné bloky textu jsou zašifrovány stejně. Nalezení několika stejných bloků v šifře může v některých případech znamenat nalezení otevřeného textu. Není zajištěna integrita otevřeného textu. Útočník může libovolně vkládat, zaměňovat nebo mazat bloky. Používá se pouze na šifrování klíčů.

  Formální zápis: $C = E_K(P)$

- Dalším módem je použití inicializačního vektoru. Ten se spojí funkcí XOR s blokem vstupního textu, pak se použije šifrovací funkce a výsledný zašifrovaný text se stane inicializačním vektorem dalšího bloku. Nazývá se CBC (Cipher Bloch Chaining) a je vhodný pro šifrování zpráv.

  Formální zápis: $C_n = E_K(P_n \oplus C_{n-1})$

- Obdobné použití má i CFB (Cipher FeedBack), kde se vstupní vektor zašifruje klíčem a po té se provede XOR se vstupním textem. Výsledný text je pak použit jako vstupní vektor následujícího bloku.

  Formální zápis: $C_n = E_K(C_{n-1}) \oplus P_n$

- OFB (Output FeedBack) se liší od předchozího tím, že jako vstupní vektor následujícího bloku bere data hned po zašifrování vektoru a funkce ještě před XORem se vstupním textem.

  Formální zápis: $C_n = H_n \oplus P_n$
  $H_n = E_K(H_{n-1})$

### 2.5.2 Asymetrické šifrování

Neboli Public-key systémy (systémy s veřejným klíčem) používají jednosměrné (trapdoor) funkce pro výpočet veřejného klíče, jejichž inverzní funkci lze efektivně spočítat jen s použitím dodatečných informací. Schéma šifrování je naznačeno na obrázku 2.4. Asymetrické šifrování je z důvodu používání složitých matematických operací pomalé. Na rozdíl od symetrického šifrování není třeba sdílet žádné tajemství.

Každý uživatel vlastní pár klíčů:

- Veřejný (public) klíč: Znají ho všichni uživatelé systému a používá se k šifrování zpráv zasílaných tomuto uživateli.
- Tajný, soukromý (private) klíč - má uživatel uchovaný v tajnosti a používá ho k dešifrování přijatých zpráv. Soukromý klíč nelze efektivně odvodit ze znalosti odpovídajícího veřejného klíče.

Příklady současných asymetrických šifer:

#### Merkle Hellman systém

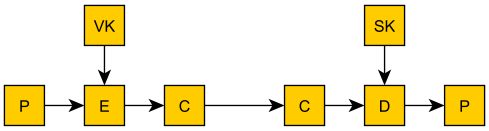
je založen na NP (nedeterministicky polynomiální) složitosti zavazadlového problému. Přenášená zpráva je chápána jako vektor řešení. Zašifrovaná zpráva je výsledná suma - hmotnost batohu. Dá se použít pouze pro šifrování. K prolomení Merkle Hellman šifrování není nutné vyřešit obecný problém batohu. Je nevhodný k ochraně důležitých informací.

#### El Gamal

je založen na obtížnosti výpočtu diskrétních logaritmů v konečném tělese. Používá se pro šifrování i elektronické podepisování. Nevýhodou algoritmu je 2 krát větší délka šifrovaného než vstupního textu.

**Rivest Shamir Adelman**

je založen na problému rozkladu velkého čísla na součin prvočísel (faktorizace). Neexistuje žádný algoritmus, který by pracoval alespoň v polynomiálním čase vůči binárnímu zápisu rozkládaného čísla. Je prakticky nemožné z čísla $n(n = p \times q)$ zjistit v rozumném čase čísla $p$ a $q$. Prvočísla se volí se 100 - 200 číslicemi a řádově stejné. Rychlost realizace je asi 1000 krát pomalejší než algoritmus DES.




Obrázek 2.4: Schéma asymetrického šifrování

Použité zkratky:

- E - šifrovací funkce (encrypt)
- D - dešifrovací funkce (decrypt)
- VK - veřejný klíč (public key)
- SK - soukromý klíč (private key)
- P - prostý text (plain text)
- C - zašifrovaný text (cipher text)

Formální zápis:
Šifrování: $C = E_{VK}(P)$
Dešifrování: $P = D_{SK}(C)$

### 2.5.3 Jednocestné funce - HASH

HASH je jednocestná šifrovací funkce, která převede libovolný text na krátký řetězec konstantní délky (HASH kód).

Vlastnosti:

- Jednoduchý výpočet
- Nemožnost zjištění původního textu ze znalosti HASH kódu
- Pokud je vypočítaný kód ze dvou textů stejný, pak i tyto texty jsou stejné.
- Libovolné množství dat má vždy kód stejné délky.
- Drobná změna vstupních dat má za následek velkou změnu výstupního kódu.

Použití:

- Symetrické systémy - slouží k rozpoznání pravosti dešifrované zprávy.
- Asymetrické systémy - rychlejší autentizace: spočítá se hashovací funkce nad danou zprávou a elektronicky se podepíše až výsledný hashkód.
- Ochrana hesel.

HASH funkce lze rozdělit do dvou kategorií podle použití tajného klíče. Závisí-li výpočet HASH funkce na tajném klíči, označujeme tuto funkci jako MAC (message authentization code). Pokud takový klíč použit není, jde o MDC (manipulation detection code).

#### MD5

pracuje nad vstupními bloky délky 512 bitů a výstup má délku 128 bitů. Algoritmus počítá ve čtyřech cyklech. V současné době byly publikovány útoky na kompresní funkci MD5. Neznamenají aktuální nebezpečí pro všechny implementace, ale svědčí o slabosti algoritmu. Obecně platí, že algoritmy s délkou výstupního řetězce do 160 bitů, jsou považovány za prolomené.

#### SHA-1

Výstupní kód má délku 160 bitů. Pracuje ve třech krocích:

- Inicializace – zarovnání vstupu, příprava interních datových struktur.

- Iterace kompresní funkce – postupná aplikace kompresní funkce na bloky zprávy, akumulace historie výpočtu.
- Dokončení – konstrukce finálního výsledku z akumulovaných vnitřních stavů.

**SHA-256, SHA-512**

jsou principiálně stejné algoritmy jako SHA-1. Liší se od sebe délkou slova (32, nebo 64 bytů) a počtem kol (64 u SHA-256, 80 u SHA-512). Pokrývají slabiny předchozích verzí algoritmů.

### 2.5.4 Existující knihovny a řešení - Protokol SSL

Symetrické šifrování má velmi rychlé algoritmy pro šifrování a dešifrování. Problémem je, jak předat klíč druhé straně? Asymetrické šifrování má přesně opačné vlastnosti. Algoritmus je mnohem pomalejší kvůli ochraně soukromého klíče. Veřejný klíč může být předán druhé straně bez obavy, že útočník rozluští zprávu jím zakódovanou.

Protokol SSL a nyní novější TSL (vycházející ze stejného principu) využívají všech výhod šifrovacích algoritmů a zároveň zamezuje jejich nevýhodám. Protokol SSL/TSL naváže mezi klientem a serverem zabezpečené šifrované spojení a zajistí autenticitu obou komunikujících stran.

Protokol TSL se liší od protokolu SSL zavedením proměnné délky doplnění bloku při použití blokových šifer (Ztíží se tím útok pomocí analýzy délky zpráv). Dále je odlišný algoritmus výpočtu MAC zpráv a změny jsou i ve zprávách `CERTIFICATE_VERIFY`. Lze proto považovat popis SSL za shodný s popisem TSL.

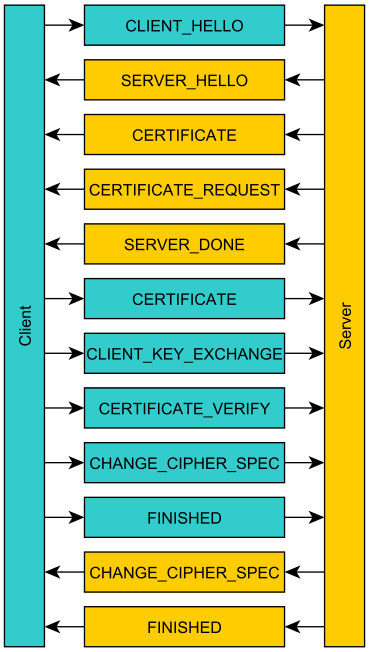
Protokol pracuje ve třech fázích.

1. Z důvodu velkého množství používaných algoritmů šifrování je nutné dohodnout se, jaké algoritmy budou použity vzhledem k možnostem klienta i serveru.
2. Je třeba zajistit, že klient komunikuje skutečně s požadovaným serverem či službou a musejí si předat šifrovací klíč. Tato část (auten-

tifikace) je prováděna pomocí asymetrických šifer. Klíč je generován pomocí HASH funkce.

3. Šifrování přenosu dat symetrickou šifrou pomocí předaného klíče. Ověřování přenášených dat je prováděno pomocí MAC (message authentication code).

Prvním dvěma fázím se říká Handshake (podání si rukou). Na obrázku 2.5 je znázorněn proces Handshake, který probíhá následovně.




Obrázek 2.5: Schéma asymetrického šifrování

- `CLIENT_HELLO` Klient oznamuje nejvyšší verzi SSL/TLS, kterou podporuje, náhodné číslo a seznam doporučených šifrovacích sad a kompresních metod.
- `SERVER_HELLO` Odpověď serveru obsahuje zvolenou verzi protokolu, náhodné číslo, šifrovací a kompresní metodu vybranou z klientem nabídnutého seznamu.
- `CERTIFICATE` Server odešle svůj certifikát, pokud to zvolená šifra umožňuje. Certifikát obsahuje jméno serveru, důvěryhodnou certifikační autoritu (CA) a veřejný klíč serveru.

- `CERTIFICATE_REQUEST` Server může od klienta požadovat autentifikaci, aby spojení bylo navzájem důvěrné.
- `SERVER_DONE` Server oznámí, že je hotov.
- `CERTIFICATE` Klient posílá svůj certifikát, pokud byl serverem vyžádán a pokud certifikát vlastní.
- `CLIENT_KEY_EXCHANGE` Klient v závislosti na zvolené šifře posílá předběžný klíč zašifrovaný veřejným klíčem serveru nebo svůj veřejný klíč a po té proběhne výměna předběžného klíče. Obě strany spočtou z předběžného klíče šifrovací klíč.
- `CERTIFICATE_VERIFY` Pokud server požadoval ověření klienta, klient pošle náhodné číslo ze `SERVER_HELLO` a předběžný klíč zašifrovaný svým soukromým klíčem. Server data dešifruje obdrženým veřejným klíčem klienta.
- `CHANGE_CIPHER_SPEC` Klient oznamuje, že následně už bude komunikovat šifrovaně.
- `FINISHED` Klient konec.
- `CHANGE_CIPHER_SPEC` Server oznamuje, že následná komunikace bude probíhat šifrovaně.
- `FINISHED` Server konec.

## 2.6 Ověření uživatele

Autentizace je proces ověření proklamované identity subjektu (druhé strany). Autentizují se entity (osoby, programy) ale také zprávy. Obecně se snaží dvě komunikující strany zajistit mezi sebou bezpečné spojení pomocí přenosu zpráv nebo ověřením pomocí důvěryhodné třetí strany. Cílem po dokončení přenosu zpráv je, že obě strany mají ověřené identity (opravdu komunikují mezi sebou). Často je současně vytvořen i klíč pro šifrování spojení symetrickým algoritmem (relační klíč (session key)).

Autentizační metody:

- Uživatel zná: Tajemství známé jen komunikujícím stranám například uživatelské jméno a heslo. Používají se kryptografické protokoly typu výzva odpověď.
- Uživatel vlastní: Fyzický objekt - čárové kódy, magnetické a čipové karty.
- Vlastnosti uživatele (biometrické informace): otisk prstu, sítnice, hlas, DNA...

Autentizační protokoly pracují na principu výzva - odpověď a využívají různé metody šifrování. V odpovědi na výzvu se sleduje znalost sdíleného tajemství. Uplatňují se asymetrické a symetrické šifrovací metody a hashovací funkce. Dále se používají časová razítka pro zamezení zopakování zpráv. Heslo se nikdy neposílá přes síť, ale pouze HASH hesla.

- Prosté heslo - Metoda zasílá ID uživatele společně s HASH otiskem hesla na server, který ověří ve svých záznamech, zda dodanému ID odpovídá HASH otisk hesla.
- X.509 je standard pro systémy založené na veřejném klíči. Protokoly využívající veřejné klíče v podobě digitálních certifikátů umožňují identifikaci držitelů certifikátů s využitím jejich soukromých klíčů. Uživatel zašifruje zprávu svým soukromým klíčem a druhá strana tuto zprávu může rozšifrovat s využitím veřejného klíče obsaženého v certifikátu. Certifikát obsahuje identifikační údaje držitele, platnost, údaje o vydavateli certifikátu, sériové číslo a zmíněný veřejný klíč. Certifikát vydává certifikační autorita (CA - Certificate Authority), která ručí za správnost údajů.
- Kerberos je založený na ověřování identity pomocí třetí důvěryhodné strany. Nikdy neposílá heslo uživatele přes síť. K ověřování vůči dalším službám využívá tikety. Uživatel dostane první tiket zašifrovaný hashem jeho hesla, které má Kerberos uložené ve své databázi.

### 2.6.1 Kerberos

Protokol Kerberos splňuje požadavky pro bezpečnou autentizaci uživatelů a služeb. Nikdy nepřenáší hesla po síti. Veškerá důležitá data jsou šifrována.

Díky operačnímu systému Windows a systému Active Directory se stal velmi rozšířeným.

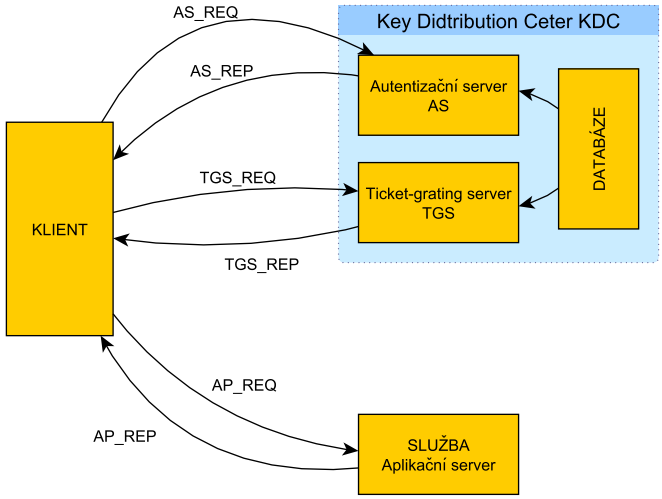
Protokol Kerberos je založený na variantě Needham-Schroedenova protokolu[15] doplněného o časová razítka (doporučení Denningové a Sacca[4]) za předpokladu, že všechny časy jsou synchronizovány s KDC (Key Distribution Center). Protokol využívá třetí důvěryhodné strany KDC. KDC obsahuje autentizační server (AS) a Ticket-Granting server (TGS). Kerberos pracuje na principu tiketů sloužících k ověření identity uživatelů.

KDC udržuje databázi tajných klíčů. Každá entita v síti vlastní svůj tajný klíč známý pouze entitě a KDC. Pomocí znalosti klíče se prokazuje identita entity. Pro komunikaci mezi entitami KDC vygeneruje nový klíč používaný pro šifrování komunikace.

Seznam použitých zkratek a pojmů.

- **AS** - Autentizační server je součástí KDC.
- **KDC** - Key Distribution Center obsahuje AS a TGS. Poskytuje entitám klíče v podobě tiketů.
- **SS** - Servisní středisko je entita poskytující službu, kterou chce jiná entita v síti využívat.
- **TGS** - Ticket Granting Server generuje tikety obsahující klíče pro další komunikaci, které pak KDC poskytuje entitám v síti.
- **TGT** - Ticket Granting Ticket – tiket opravňující uživatele ke komunikaci s řídícím serverem.
- **Pricipál** - Název služby nebo uživatele společně se síťovou adresou.
- **Keytab** - Obsahuje principál a klíč vygenerovaný z hesla. Používá se pro služby (SS).
- **Tiket** - Je unikátní klíč, kterým se uživatel ověřuje vůči dalším službám.
- **Realm** - Neboli říše je oblast spravovaná protokolem Kerberos, ve které se nacházejí uživatelé a služby. Zpravidla odpovídá doméně.

Klient začíná proces ověřením se vůči AS pomocí navzájem známého tajemství. Obdrží tiket (TGT), se kterým kontaktuje TGS a požádá o přístup ke službě na SS. TGT se může používat opakovaně. TGS klientovi pošle další tiket, který obsahuje klíče umožňující ověření vůči SS. Tímto způsobem je možné se autentizovat k dalším službám bez nutnosti znovu zadávat tajemství (heslo). Celý proces je vyznačen na obrázku 2.6 a popsán následujícím seznamem. Zápis ve složených závorkách je šifrovaný klíčem K a zápis v kulatých závorkách není šifrovaný.




Obrázek 2.6: Schéma ověření protokolem Kerberos

1. Ověření uživatele
   Klient zadá své jméno a heslo. Ze jména vznikne principál klienta $P_{Client}$ a z hesla je pomocí HASH funkce vygenerován klíč $K_{User}$.

   - *AS_REQ*: Klient odešle svůj principál a principál služby, ke které se chce připojit, seznam adres, ze kterých se připojuje (může být prázdný kvůli přístupu přes NAT) $IPlist$ a dobu životnosti tiketu $L$.
   - *AS_REP*: Od AS dostane odpověď, pokud jsou oba principálové nalezeni v databázi. Odpověď se skládá z části obsahující klíč sezení s TGS zašifrované klíčem $K_{User}$ vygenerovaného AS a TGT zašifrovaného klíčem TGS.

Uživatel může dešifrovat klíč sezení s TGS jen pokud si klíče $K_{User}$ vygenerované na straně klienta a serveru odpovídají.

$$AS\_REQ = (P_{Client}, P_{Service}, IPlist, L)$$

$$TGT = (P_{Client}, P_{krbtgt}, IPlist, T, L, SK_{TGS})$$

$$AS\_REP = \{P_{Service}, T, L, SK_{TGS}\}K_{User}, \{TGT\}K_{TGS}$$

2. Autorizace přístupu ke službě

   - *TGS_REQ*: Klient zašle TGS požadavek o přístup na službu s použitím TGT a autentifikátoru zašifrovaného klíčem sezení.
   - *TGS_REP*: V odpovědi dostane $TKT_{Service}$ tiket pro přístup na službu zašifrovaný klíčem služby a klíč sezení se službou $SK_{Service}$.

$$Authenticator = \{P_{Client}, T\}SK_{TGS}$$

$$TGS\_REQ = (P_{Service}, L, Authenticator), \{TGT\}K_{TGS}$$

$$TKT_{Service} = (P_{Client}, P_{Service}, IPlist, T, L, SK_{Service})$$

$$TGS\_REP = \{P_{Service}, T, L, SK_{Service}\}SK_{TGS}, \{TKT_{Service}\}K_{Service}$$

3. Vyřízení žádosti o přístup ke službě

   - *AP_REQ*: Klient pošle svůj autentifikátor službě zašifrovaný klíčem sezení a TKT.
   - *AP_REP*: Služba získá z TKT klíč sezení a pomocí něj dešifruje z autentifikátoru časovou značku T, kterou pošle zpět navýšenou o 1. Tím se ověří vůči klientovi. Tento krok je volitelný.

$$Authenticator = \{P_{Client}, T\}SK_{Service}$$

$$AP\_REQ = Authenticator, \{TKT_{Service}\}K_{Service}$$

$$AP\_REP = \{T + 1\}SK_{Service}$$

Veškerá autentizace je řízena centrálně přes KDC, proto je nutné tento server zabezpečit. Pokud by potenciální útočník získal kontrolu nad KDC, mohl by se vydávat za jakéhokoliv uživatele.

### GSSAPI

Rozhraní GSSAPI (Generic Security Service Application Program Interface) nabízí jednotné funkce pro programování bezpečnostních aplikací. Organizace IETF (Internet Engineering Task Force[9]) nastavuje standard používaný v GSSPI, který sjednocuje použití mnoha podobných, ale nekompatibilních bezpečnostních služeb. GSSAPI je implementováno poskytovateli bezpečnostních služeb jako soubor knihoven instalovaných společně s jejich softwarem.

Aplikace využívající GSSAPI spolu komunikují pomocí paketů GSSAPI, které mohou cestovat veřejnou sítí nezabezpečeně. Pakety jsou zabezpečené mechanizmy bezpečnostních služeb. Aplikace na obou stranách jsou po výměně paketů informovány o stavu zabezpečení kontext. Jakmile je bezpečnostní kontext navázán, zprávy mohou být zaobaleny (šifrovány). GSSAPI zajišťuje šifrování a integritu dat a také garantuje identitu komunikujících stran.

GSSAPI bylo standardizováno pro programovací jazyk C dokumentem RFC 2744[10]. Pro Javu se implementace jmenuje JGSS[17].

GSSAPI využívá mechanizmu SPNEGO (Simple and Protected GSSAPI Negotiation Mechanism[11]), který vyjednává mezi aplikacemi bezpečnostní mechanizmy podporované oběma stranami.

## 2.7 Pověření uživatele

Potřeba sdílení dat s dalšími uživateli přináší nutnost zavedení přístupových práv. Tradiční model v unixových systémech funguje tak, že každému souboru je přiřazen právě jeden uživatel jako vlastník, který může se souborem manipulovat dle zvlášť přidělených práv. Dále je souboru přiřazena jedna skupina uživatelů. Zbylí uživatelé mají práva přidělená pomocí skupiny "ostatní". K souboru není možné přidělovat další práva pro další uživatele. Je sice možné vytvářet samostatné skupiny uživatelů pro přidělování práv k souborům, ale to je příliš složité řešení.

Další možností nastavování přístupových práv k souborům jsou přístupové seznamy ACL (Access Control Lists). Tyto seznamy jsou definované v bezpečnostním dodatku standardu POSIX (PSIX 1003.1e/1003.2c). Přístu-

pový seznam ACL umožňuje vytvářet libovolně dlouhé seznamy oprávnění pro daný soubor. Různí uživatelé mohou mít různá práva. Práva k souboru jsou určena jedním seznamem ACL. Každý záznam v seznamu nese informaci pro konkrétní kategorii uživatelů (uživatel, skupina , ostatní). Kategorie vlastník souboru (`USER_OBJ`), vlastnická skupina (`GROUP_OBJ`) a ostatní (`OTHER`) mají stejný význam jako v případě tradičních práv a jsou v ACL povinné. Zajišťují zpětnou kompatibilitu. Další záznamy jsou typu uživatel (`USER`), skupina (`GROUP`) a přístupová maska (`MASK`). Slouží k nastavení přístupových práv dalším uživatelům a skupinám. Záznam `MASK` definuje omezující přístupová práva pro záznamy `USER`, `GROUP` a `GROUP_OBJ`. Výsledná práva jsou logickým součtem práv a masky.

Pro distribuované souborové systémy není možné používat tradiční model přístupových práv. V systému se nachází velké množství uživatelů a definování práv tímto způsobem nemůže obsáhnout všechny potřeby.

# 3 Praktická část

V této části se budeme zabývat realizací bezpečnostního modelu pro distribuovaný souborový systém KIVFS. Je třeba ověřit stávající stav a zajistit funkčnost. Většina bezpečnostních prvků se nachází v bezpečnostní vrstvě KIVFS.

Jako klienta využijeme stávající implementaci testovacího klienta pro operační systém linux (distribuce Debian) bez zabezpečení. Zde se zaměříme na upravení klienta pro využívání zabezpečené komunikace a autentizaci.

Serverová část bezpečnostní vrstvy bude přeložena a ověřena funkčnost šifrování a autentizování.

Dále je třeba ověřit funkčnost tabulek přístupových práv ACL, které jsou implementovány v databázové vrstvě a zavést testování přístupu k souborům a adresářům do bezpečnostní vrstvy.

## 3.1 Ověření

Ověření funkčnosti serverové části bezpečnostní vrstvy nazývané kivfs-proxy-server se nezdařilo. Od doby, kdy byla vytvořena předchozí verze kivfs-proxy-serveru, bylo provedeno mnoho úprav v jádře systému, které vedly k nemožnosti přeložit tuto část. Bylo rozhodnuto, že kivfs-proxy-server bude předělán od základů.

Autorizování uživatelů pomocí ACL bylo ověřeno a je implementováno v databázové části KIVFS. Server se jmenuje kivfs-db-server. Test se skládal z vytvoření adresáře a souboru, který byly postupně nastavovány práva pro testovacího uživatele. Pomocí testovacího uživatele byl ověřen přístup nebo zamítnutí přístupu v závislosti na nastavených právech. Více informací o implementaci viz. kapitola 3.4

## 3.2 Zabezpečení přenosu

Šifrování spojení mezi klientem a proxy vrstvou bude využívat protokolu SSL nebo TSL a bude implementováno s využitím knihoven OpenSSL. Tím bude zaručena bezpečnost přenosu dat proti odposlechu a zároveň ověření autenticity serveru pomocí certifikátu. Ověření klientské aplikace vynecháme z důvodu ověřování uživatele přes protokol Kerberos. Ověření klientů by se dalo aplikovat, ale znamenalo by to pro každého klienta vydat další certifikát.

Volba konkrétního protokolu a šifrovacích algoritmů je založena na softwarových možnostech serveru a klienta. Záleží, jakou verzi knihoven OpenSSL používá klient a server. Za konkrétní volbu u vyšších verzí protokolu SSL odpovídá server. Je možné pomocí funkce `SSL_CTX_set_cipher_list` vybrat, které algoritmy mohou být použity. Pokud klient nemůže použít na serveru vybranou variantu šifrovacích algoritmů, nebude připojen. Nastavení je možné provést v konfiguračním souboru serveru viz. kapitola 3.5.2.

Abychom mohli SSL protokol využít, je nutné nainstalovat balíček OpenSSL (`apt-get install libssl-dev`), jak na server tak na klienta.

Překlad pomocí překladače gcc je nutné zadat s parametrem -lssl, aby se použily knihovny pro SSL.

Pro použití protokolu SSL je nezbytné vygenerovat certifikáty, certifikační autoritu a soukromý klíč. Veřejný klíč je obsažen v certifikátu spolu s identifikačními údaji serveru. Dříve než budeme generovat certifikáty je vhodné upravit konfiguraci OpenSSL. Konfigurace se nachází: `/etc/ssl/openssl.cnf` Zde se podíváme hlavně na položky `dir` a `default_days`. Položka `dir` udává umístění vygenerovaného certifikátu a `default_days` počet dní platnosti a co nejvíce uvedených položek. Při generování se tak dá ušetřit čas. Samotné generování probíhá následovně:

1. Navigujte se do adresáře zadaného v konfigurace OpenSSL a použijte následující příkaz:

```
openssl req -new -x509 -days 3650 -extensions v3_ca
-keyout private/cakey.pem
-out cacert.pem
-config /etc/ssl/openssl.cnf
```

Soubor `cacert.pem` obsahuje certifikát certifikační autority a je možné s jeho pomocí generovat další certifikáty.

2. Privátní klíč a požadavek pro certifikát s veřejným klíčem serveru vytvoříme následovně:

```
openssl req -new -nodes
-out kivfs_server_cert_req.pem
-keyout private/kivfs_server_cert_key.pem
-config /etc/ssl/openssl.cnf
```

Pokud zadáte heslo, budete na něj tázáni pokaždé, když budete spouštět server. V našem případě heslo vynecháme.

3. Teď už zbývá jen veřejný klíč serveru:

```
openssl ca
-infiles kivfs_server_cert_req.pem -days 3650
-out kivfs_server_cert.pem
-config /etc/ssl/openssl.cnf
```

Máme tři soubory, které budeme potřebovat pro správnou funkci serverové části - proxy serveru. Jsou to tyto soubory:

- `cacert.pem` - Certifikát certifikační autority
- `kivfs_server_cert.pem` - Certifikát serveru podepsaný CA
- `kivfs_server_cert_key.pem` - Soukromý klíč serveru

Takto vytvořený certifikát certifikační autority bohužel není důvěryhodný. Pro testovací účely nám stačí. V budoucnu se použije certifikát ZČU s jehož pomocí se připraví i serverový klíč a certifikát. K tomu se dá použít stejný postup jen s vynecháním prvního kroku.

Certifikáty (respektive soukromý a veřejný klíč) klientské aplikace pro dohodnutí šifrovacího klíče spojení budou vytvářeny automaticky při inicializaci.

### 3.2.1 Kód

Pro práci s protokolem SSL a šifrovaným spojením jsem připravil funkce, jejichž jméno začíná `KVFS_SSL_`. Nacházejí se v souboru `kivfs-net.c`, který je součástí knihovny jádra KIVFS. Jsou to funkce pro zahájení šifrovaného spojení a pro šifrovaný přenos dat.

K provedení tzv. handshake, což je vlastně navázání šifrovaného spojení, slouží funkce:

- `kivfs_ssl_client_connect` na straně klienta a
- `kivfs_ssl_server_connect` na straně serveru.

Obě funkce vracejí ukazatel na strukturu `kivfs_connection_t`, kde jsou uloženy ukazatele na proměnnou typu `SSL` a `SSL_CTX`. Proces u obou funkcí probíhá obdobně.

1. Nejprve je nutné inicializovat knihovnu SSL.

```
SSL_library_init();
```

2. Pak se musí nastavit proměnná typu `SSL_CTX` obsahující souvislosti určující klientskou část a serverovou část, požadovanou verzi SSL a certifikáty.

```
ssl_content = SSL_CTX_new(SSLv23_server_method());
SSL_CTX_use_certificate_file(ssl_content, cert,
                             SSL_FILETYPE_PEM);
SSL_CTX_use_PrivateKey_file(ssl_content,
                            cert_key, SSL_FILETYPE_PEM);
SSL_CTX_load_verify_locations(ssl_content,
                              ca_cert, NULL);
```

3. V dalším kroku se vytvoří proměnná typu `SSL` za pomoci předchozího nastavení. Tato proměnná je dále používána jako identifikátor spojení.

```
ssl_handle = SSL_new(ssl_content);
```

4. Následuje vytvoření vazby mezi identifikátorem existujícího nezabezpečeného spojení (Socket) a identifikátorem SSL spojení z předchozího kroku.

```
SSL_set_fd(ssl_handle, socket)
```

5. Po nastavení je zavolána funkce, která provede handshake. Na straně serveru je to:

```
SSL_accept(ssl_handle)
```

   A na straně klienta:

```
SSL_connect(ssl_handle)
```

Při programování funkcí pro odesílání a přijímání dat jsem se držel standardu nastaveného dalšími programátory KIVFS. Funkce mají téměř stejnou strukturu. Rozdíl je ve využití SSL vrstvy. Tím je zajištěna nejvyšší míra kompatibility. Zasílají se stejné části dat, jako v nezašifrované podobě, tedy hlavička a tělo.

```
/**
 * Sends data and receives answer.
 * Uses the same structure like the function without ssl.
 * @param ssl_con
 * @param msg           message to send
 * @param p_msg         message to receive
 * @return              kivfs error code
 */
int kivfs_ssl_send_and_receive(SSL *ssl_con, kivfs_msg_t *msg,
    kivfs_msg_t **p_msg) {
  int error_code = 0;
  if (!(error_code = kivfs_ssl_send(ssl_con, msg)))
    error_code = kivfs_ssl_recv(ssl_con, p_msg);
  return error_code;
}
```

Takto naprogramované funkce umožnily zjednodušení při vývoji klientské aplikace. Kód stačilo upravit jen jednoduchým makrem nahrazujícím původní

funkce za nové se stejnými parametry kromě identifikátoru socketu. Ten byl nahrazen identifikátorem SSL spojení.

```
#define kivfs_send_and_receive(x, y, z) \
kivfs_ssl_send_and_receive(ssl_handle, y, z)
```

## 3.3 Ověření uživatele

Ověření uživatele bude provedeno pomocí protokolu Kerberos. Výhodou použití protokol Kerberos je, že neposílá uživatelské heslo v žádné formě přes síť. Západočeská univerzita používá protokol Kerberos k ověřování uživatelů, to otevírá nám možnost k propojení ověřování s KIVFS.

Pro implementaci byl zvolen Kerberos distribuce Heimdal. Na rozdíl od distribuce MIT (The Massachusetts Institute of Technology), která byla vyvíjena na území USA, se na distribuci Heimdal (vyvíjeného převážně ve Švédsku) nevztahují exportní regulace. Aby volba distribuce neměla výrazný dopad na flexibilitu kódu, bylo využito aplikační rozhraní GSSAPI.

### 3.3.1 Instalace

Kerberos Heimdal a potřebné balíčky nainstalujeme následujícím příkazem: `apt-get install heimdal-kdc heimdal-servers heimdal-servers-x`

### 3.3.2 Nastavení konfiguračních souborů

Po instalaci balíčků je nutné Kerberos nastavit. Nejprve budeme upravovat soubor: `/etc/krb5.conf`, kde je třeba nastavit výchozí říši (`default_realm`), naši požadovanou říši a její vlastnosti a propojení mezi říší a doménovým názvem.

Ukázka konfigurace `/etc/krb5.conf`:

```
[libdefaults]
```

```
  default_realm = DFS.ZCU.CZ
  extra_addresses = 147.228.209.153 147.228.67.127
  147.228.67.103 88.146.153.66

[realms]
  DFS.ZCU.CZ = {
    kdc = fs-1.kiv.zcu.cz
    admin_server = fs-1.kiv.zcu.cz
  }

[domain_realm]
  .kiv.zcu.cz = DFS.ZCU.CZ
  kiv.zcu.cz = DFS.ZCU.CZ
```

Pro Kerberos jsou důležité DNS záznamy. Nastavení provedeme v souboru `/etc/hosts` nebo na DNS serveru. Zapíšeme adresy a jména KDC, serverů se službami a další, které jsme uvedli v `/etc/krb5.conf`. Soubor hosts pak vypadá následovně:

```
147.228.63.46 fs-1.kiv.zcu.cz fs-1 kivfs-1.kiv.zcu.cz kivfs-1
147.228.63.47 fs-2.kiv.zcu.cz fs-2 kivfs-2.kiv.zcu.cz kivfs-2
147.228.63.48 fs-3.kiv.zcu.cz fs-3 kivfs-3.kiv.zcu.cz kivfs-3
147.228.63.63 fs-4.kiv.zcu.cz fs-4 kivfs-4.kiv.zcu.cz kivfs-4
```

Ještě se musí nastavit soubor `/etc/resolv.conf`, aby Kerberos věděl, kde má hledat překlady adres.

```
search kiv.zcu.cz
nameserver 147.228.52.11
```

### 3.3.3 Nastavení databáze KDC

Příkazem `kadmin -l` spustíme správce. Parametr -l značí, že se kadmin spouští lokálně pod místním administrátorským účtem. V programu kadmin inicializujeme říši (realm), nastavíme uživatele a jejich hesla, služby, které vyexportujeme.

- Inicializace realmu: `init DFS.ZCU.CZ`
- Přidání principálu uživatele (zkrácená a kompletní verze jména):

```
fs-1:~# kadmin -l
kadmin> add root
kadmin> add root@DFS.ZCU.CZ
```

  Bez použití části za @ se říše doplní podle standardní hodnoty zadané v konfiguraci Kerbera.

- Jelikož je protokolem Kerberos ověřována i služba je nutné přidat jejich principály:

```
kadmin> add --random-key kivfs/fs-1.kiv.zcu.cz
kadmin> add --random-key kivfs/fs-2.kiv.zcu.cz
kadmin> add --random-key kivfs/fs-3.kiv.zcu.cz
kadmin> add --random-key kivfs/fs-4.kiv.zcu.cz
```

  Část za `@` se opět doplní podle nastavení `default_realm`.

- Export do keytabu (soubor, ve kterém je uchováván principál a tajný klíč služby):

```
kadmin> ext kivfs/fs-1.kiv.zcu.cz
kadmin> ext kivfs/fs-2.kiv.zcu.cz
kadmin> ext kivfs/fs-3.kiv.zcu.cz
kadmin> ext kivfs/fs-4.kiv.zcu.cz
kadmin> exit
```

Výpis vyexportovaných keytabů se provádí příkazem `ktutil list`. Keytaby je třeba umístit na servery, kde běží daná služba.

Správnou funkci Kerbera můžeme ověřit pomocí programu `kauth`, který se vás pokusí autentifikovat vůči Kerberu. Spuštěním programu `klist` se můžeme podívat na vlastnosti přiděleného tiketu.

### 3.3.4 Kód

Pro samotnou integraci Kerbera do serveru jsem využil rozhraní GSSAPI (Generic Security Standard API[8]). GSSAPI vznikla s potřebou sjednotit bezpečnostní mechanizmy a standardizovat jejich použití.

Při vytváření klientské aplikace jsem musel řešit problém. GSSAPI získává credentials (uživatelská pověření, tiket) ze systému od aktuálně přihlášeného uživatele pokud je systém v doméně nebo v říši Kerbera. V našem případě se uživatelé budou snažit přihlašovat převážně ze systémů v domácích skupinách nebo zcela jiných. Je třeba vytvořit nové credentials buď použitím programu `kauth` nebo knihovny krb5. Využití knihovny krb5 v klientské aplikaci je uživatelsky příjemnější. Proces získání credentials zajišťuje funkce:

```
kivfs_krb_client_login(user, NULL /* password */, &user_creds);
```

Pokud zadáte prázdné heslo, aplikace se vás na něj dotáže. V této funkci se inicializuje knihovna Kerberos, nastavují se různé parametry. Nejdůležitější funkcí části kódu je:

```
krb5_verify_user(context, princ, ccache, password, TRUE, NULL);
```

Tato funkce provede vytvoření uživatelových credentials a uloží je do credentials cache (proměnná `ccache`). Dále proběhne import credentials do GSSAPI.

```
major_status = gss_krb5_import_cred(
    &minor_status,
    ccache,
    princ,
    NULL,
    &cred);
```

Proměnné `major_status` a `minor_status` vracejí chybové kódy GSSAPI (major) a použitého mechanizmu (minor). Import proběhne přímo z credentials cache. Credentials jsou identifikována pomocí principála uživatele a jsou uložena do proměnné `cred`.

Následuje funkce, která ověří uživatele vůči Kerberu a předá uživateli tiket pro přístup na službu. Služba vlastní svůj šifrovací klíč, kterým tiket dešifruje. Ověří klienta, pak odešle klientovi časovou známku uloženou na tiketu navýšenou o 1 a tím se ověří vůči klientovi.

```
kivfs_krb_client_init_sec_context(
    session->vfs_connection->socket,
    cred,
    &outGSSContext);
```

V této části je nejdůležitější funkcí `gss_init_sec_context`, která získá pro uživatele tiket pro přístup na požadovanou službu a ověří, že se uživatel skutečně připojuje k této službě. to znamená, že služba je také ověřována vůči KDC. GSSAPI pro komunikaci používá vlastní systém paketů, kterým obaluje data posílaná bezpečnostními mechanizmy. Často je nutné vyměnit více paketů, proto celý proces běží ve smyčce dokud je zapotřebí, což je hlášeno v návratovém kódu `maj_stat`. Funkce má mnoho vstupních parametrů. Některé si dokáže volit sama podle dostupných mechanizmů. Funkce má také mnoho výstupních parametrů, které jsou nastavené aktuálně na hodnotu NULL (využívají se spíše pro testování). Ze vstupních parametrů nás zajímají hlavně `cred`, kde jsou uloženy credentials uživatele, `server` obsahující principál služby a výstupní parametr `context_hdl`, do kterého se uloží bezpečnostní context, který obsahuje důležité informace jako tiket a klíč sezení, využívaný dalšími funkcemi. Pak se zde nacházejí dvě proměnné `input_token` a `output_token`, které slouží k předávání paketů GSSAPI mezi klientem a serverem.

```
maj_stat = gss_init_sec_context(
    &min_stat,
    cred, /* client credentials*/
    &context_hdl, /* gss context */
    server, /* service name */
    GSS_C_NULL_OID, /* mech type */
    GSS_C_MUTUAL_FLAG | GSS_C_SEQUENCE_FLAG, /* requested flags
        */
    0, /* GSS_C_INDEFINITE */
    NULL, /* GSS_C_NO_CHANNEL_BINDINGS */
    input_token,
    NULL, /* actual mech type */
    output_token,
    NULL, /* actual flags */
    NULL /* time_rec */);
```

Následuje série testů, která dokáže, že uživatel i služba jsou ověřeni. Tyto testy zaobaluje funkce `kivfs_krb_client_do_test`. Na serveru je obdobná funkce `kivfs_krb_server_do_test`. Testy jsou prováděny hned po ověření uživatele protokolem Kerberos.

1. test spočítá MIC (message integrity code) ze zadaného textu a odešle obojí na server. Server přijme text, spočítá si svůj MIC, a porovná ho s přijatým. Ten samý test pak provede server a klient ověří MIC. Používají se funkce: `gss_get_mic` a `gss_verify_mic`.
2. test spočívá v zakódování zprávy. Klient pošle serveru zakódovanou zprávu a ten ji dekóduje. To samé provede server a klient zprávu dekóduje. Kód obsahuje také MIC zprávy. Používají se funkce `gss_wrap` a `gss_unwrap`.

Pro výpočet MIC se využívá bezpečnostního kontextu, klíč sezení. Jedná se tedy o MAC. Oba testy ověřují integritu a autenticitu textu a platnost tiketu. Vypršení tiketu je oznámeno návratovým kódem (major status) uloženým v konstantě: `GSS_S_CONTEXT_EXPIRED`.

## 3.4 Oprávnění uživatele

Oprávnění klienta k přístupu ke sdíleným souborům je implementováno pomocí ověřování přístupu v závislosti na seznamu přístupových práv (ACL) vycházejících ze standardu POSIX v databázové části systému KIVFS. Tabulka 3.1 obsahuje přehled dostupných práv.

Soubory a adresáře mají pevně definovaná práva pro:

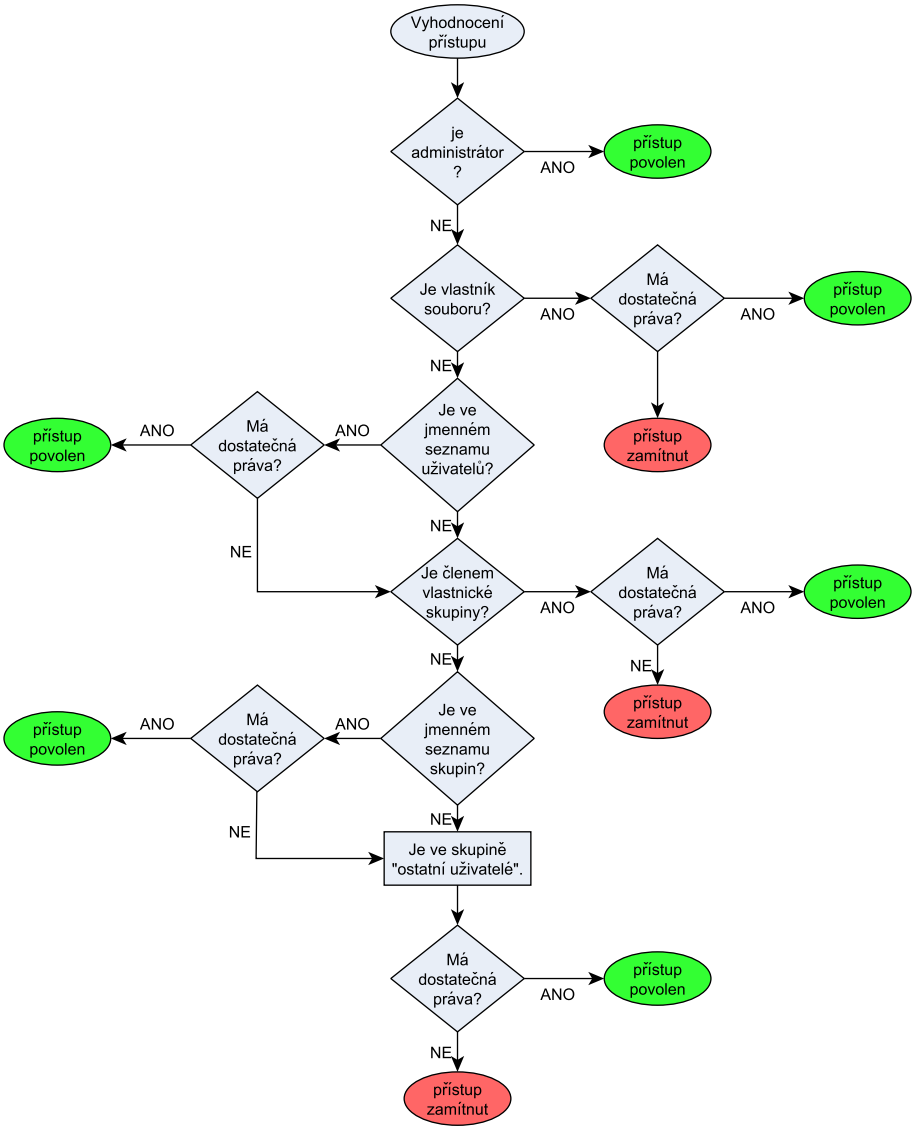
- vlastníka
- skupinu
- ostatní
- masku práv (Maska práv omezuje práva pro vlastnickou skupinu a ostatní uživatele.)

| Osmičkově | Čtení | Zápis | Spuštění |
|---|---|---|---|
| 0 | ne | ne | ne |
| 1 | ne | ne | ano |
| 2 | ne | ano | ne |
| 3 | ne | ano | ano |
| 4 | ano | ne | ne |
| 5 | ano | ne | ano |
| 6 | ano | ano | ne |
| 7 | ano | ano | ano |

Tabulka 3.1: Přístupová práva k souborům

Zvlášť se dají přiřadit definovaná práva pro roli, která může být typu uživatel nebo skupina. Práva u nových souborů nebo adresářů jsou určena dle výchozích práv nadřazené složky. Výchozí práva pro kořenový adresář a v něm vytvořené složky jsou 755 a maska 7. U souborů jsou práva nastavené na 644 a maska 7. Výchozí práva nových složek se dědí od nadřazených složek. U nových souborů se nedědí.

Vyhodnocení přístupu k souboru v závislosti na přiřazených záznamech ALC je řešeno algoritmem znázorněným na obrázku 3.1




Obrázek 3.1: Algoritmus vyhodnocování přístupu k souboru

Tento algoritmus se nachází v souboru `kivfs-db-acl.c` jako funkce:

```
int check_entry_permissions(
        kivfs_client_t *client,
        kivfs_entry_t *entry,
        kivfs_permission_bit_t permission_bit)
```

Návratovou hodnotou funkce je chybový kód kivfs. Konstanta `KIVFS_OK` odpovídá povolení přístupu. Proměnná `client` obsahuje identifikaci klienta, proměnná `entry` identifikaci souboru nebo adresáře a `permission_bit` požadované oprávnění.

Úprava práv se provádí příkazem `setfacl`. Práva se zobrazí příkazem `getfacl`.

- `setfacl <u|g|du|dg> <role> <permission> <path>`
  - u (uživatel), g (skupina), du (defaultní uživatel), dg (defaultní skupina)
  - <role> název role nebo uživatele
  - <permission> číselný zápis práv (viz. tabulka 3.1)
  - <path> cesta k souboru nebo adresáři
- `getfacl <path>`

Natavení práv je uloženo v databázi v tabulkách zvlášť pro uživatele (`acl_users`) a skupiny (`acl_groups`). Pole v tabulkách jsou shodná: `entry_id` - identifikace souboru nebo adresáře, `role_id` - identifikace uživatele nebo skupiny, `type` - typ (uživatel, skupina, defaultní uživatel, defaultní skupina, maska), `permission` - práva.

### 3.4.1 Implementace ACL v bezpečnostní vrstvě

Do serveru kivfs-proxy-server, byla přidána funkce `kivfs_proxy_chkperm`, která v případě přístupu k souboru ověří, zda má uživatel dostatečná práva. Po úspěšném ověření je možné pokračovat ve spojení s ostaními vrstvami KIVFS. Všechny příchozí zprávy (pakety KIVSF) musejí být nejprve rozbaleny, aby bylo možné zjisti, zda se zpráva týká přístupu k souboru.

```
/**
 * Checks permission on the file
 * @param session, client_request, client_response
 * @return kivfs error
 */
int kivfs_proxy_chkperm(kivfs_session_t *session, kivfs_msg_t *
    client_request, kivfs_msg_t **client_response) {
  char              *fullpath     = NULL;
  kivfs_entry_t     *entry        = NULL;
  int               error_code    = 0;

  /* unpacking of kivfs paket */
  if (!(error_code = kivfs_unpack(client_request->data,
      client_request->head->data_size, "%s", &fullpath))) {

      if (strlen(fullpath) == 0) { /* if the command doesn't
          contains file */
        return KIVFS_OK;
      }
    /* read (at least) permission checking */
    error_code = entry_with_path_and_type(session, &entry,
        fullpath, FILE_TYPE_ANY, READ_BIT);
    free(fullpath);
  }

  if (error_code) {

    /* prepare response */
    client_response = kivfs_response(client_request, error_code,
        "%file", entry->file);
    kivfs_free_entry(entry);
    return error_code;
  }
  return KIVFS_OK;
}
```

Funkce `entry_with_path_and_type`, jejíž definice se nachází v balíku databázového serveru `kivfs-db-entry`, přítomná v uvedeném kódu postupně volá výše zmiňovanou funkci `check_entry_permissions` na všechny adresáře obsažené v cestě k zadanému souboru a na soubor samotný.

## 3.5 Překlad a spuštění

Bezpečnostní vrstva KIVFS je napsaná v jazyce C. Je to samostatná serverová aplikace. Komunikuje s ostatními částmi KIVFS pomocí socketů a paketů KIVFS.

Bezpečnostní vrstva využívá funkcí knihovny libkivfscore, OpenSSL a Kerberos (implementace Heimdal). Soubory `Makefile` řídí překlad pomocí nástroje make. Všechny potřebné knihovny pro překlad knihovny libkivfscore a dalších částí systému na distribuci GNU/Linux Debian nainstalujeme příkazem:

```
apt-get install build-essential libssl-dev heimdal-dev
```

Po stažení instalace KIVFS ze svn (Subversion je systém pro správu a verzování zdrojových kódů.) vypadá adresářová struktura jako na obrázku: 3.2

```
.
|-- bin
|-- conf
|-- debian
|-- doc
|-- lib
|-- scripts
|-- sql
‘-- src
    |-- client
    |-- client2
    |-- core
    |-- daemon
    |-- db
    |-- debian
    |-- fs
    |-- proxy
    |-- proxy2
    |-- sync
    ‘-- vfs
```

Obrázek 3.2: Výpis adresářové struktury

### 3.5.1 Překlad

Pro přeložení kódu je možné v adresáři `/src` spustit příkaz `make`, který zajistí překlad všech částí KIVFS a nainstalování sdílené knihovny `libkivfscore.so`.

Překlad jednotlivých částí serveru lze provést spuštěním příkazu `make` v jednotlivých adresářích. Jako první musí být přeložena knihovna sdílená nacházející se v adresáři `/src/core` příkazem: `make install` Na zkrácené ukázce (Obrázek 3.3) je vidět přeložení knihovny `libkivfscore.so`, bezpečnostní vrstvy (proxy2) a testovacího klienta (client2).

```
fs-1:/opt/kivfs_trunk/src# make
cd core && make install
make[1]: Entering directory ‘/opt/kivfs_trunk/src/core’
Compiling all C source files: gcc -Wall -fPIC -c *.c;
Generating KIVFS Core shared library: gcc -Wall -shared *.o
-o libkivfscore.so ‘pkg-config libssl --libs‘; Done!
Shared library libkivfscore.so has been installed to your
system.
make[1]: Leaving directory ‘/opt/kivfs_trunk/src/core’
.
.
.
cd proxy2 && make
make[1]: Entering directory ‘/opt/kivfs_trunk/src/proxy2’
make[1]: Nothing to be done for ‘default’.
make[1]: Leaving directory ‘/opt/kivfs_trunk/src/proxy2’
cd client2 && make
make[1]: Entering directory ‘/opt/kivfs_trunk/src/client2’
make[1]: Nothing to be done for ‘all’.
make[1]: Leaving directory ‘/opt/kivfs_trunk/src/client2’
echo "Done!"
Done!
```

Obrázek 3.3: Ukázka překladu

### 3.5.2 Spuštění

Nejprve je nutné nastavit soubor s konfigurací, který se nachází v adresáři `conf` struktury KIVFS (obrázek 3.2), část proxy. Zde nastavíme ip adresu a port proxy serveru a cesty k certifikátů. Dále je třeba nastavit název služby (musí se shodovat s nastavením KDC a klienta), propojení s další vrstvou KIVFS a seznam šifrovacích metod pro protokol SSL.

```
[proxy]
port=30000
ip=10.0.2.15
#paths to certificate files
cert=/opt/kivfs_trunk/conf/kivfs_server_cert.pem
cert-key=/opt/kivfs_trunk/conf/kivfs_server_cert_key.pem
ca=/opt/kivfs_trunk/conf/cacert.pem
#name of the service [default: kivfs]
service=kivfs
#connection to the next KIVFS layer
to_ip=147.228.63.46
to_port=30002
#You can explicitly say what ciphers should SSL use [default:
    ALL]
cipherlist=!aNULL:!3DES:HIGH:@STRENGTH
```

Na základě zjištění o bezpečnosti šifrovacích metod v teoretické části a testech zátěže procesoru na straně serveru[14] byl zvolen následující řetězec generující seznam šifer pro protokol SSL:
`!aNULL:!3DES:HIGH:@STRENGTH`. Verze OpenSSL 0.9.8o pak podporuje tyto kombinace šifrovacích algoritmů.

```
debian:~$ openssl ciphers -v '!aNULL:!3DES:HIGH:@STRENGTH'
OpenSSL 0.9.8o 01 Jun 2010
SSLv3 Kx=DH       Au=RSA  Enc=AES(256)  Mac=SHA1
SSLv3 Kx=DH       Au=DSS  Enc=AES(256)  Mac=SHA1
SSLv3 Kx=RSA      Au=RSA  Enc=AES(256)  Mac=SHA1
SSLv3 Kx=DH       Au=RSA  Enc=AES(128)  Mac=SHA1
SSLv3 Kx=DH       Au=DSS  Enc=AES(128)  Mac=SHA1
SSLv3 Kx=RSA      Au=RSA  Enc=AES(128)  Mac=SHA1
```

Samotné spuštění proxy serveru se zadává s parametrem odkazujícím na umístění konfiguračního souboru.

```
./kivfs-proxy-server kivfs.conf
```

Server se spustí a začne naslouchat na ip adrese a portu definovaném v konfiguračním souboru.

Klient se spouští se dvěma parametry ip adresa a port serveru.

```
./test-client 10.0.2.15 30000
```

Ihned po spuštění se klient začne připojovat k serveru, ověří certifikát serveru a naváže zabezpečené spojení. Po té vyzve k zadání uživatelského jména a hesla. Dojde k ověření s použitím protokolu Kerberos, které probíhá bez použití SSL. Následná komunikace pokračuje s využitím SSL.

## 3.6 Ověření výkonu

Testování byla prováděna na serveru `fs-1.kiv.zcu.cz`.

Rychlost prováděných operací byla ověřena voláním příkazu nastavení práv zápisem do ACL, který byl proveden 2000 krát.

```
setfacl u root 1 /tmp/test
setfacl u root 7 /tmp/test
```

Tyto příkazy byly zapsány do textového souboru aclTest1000, který byl volán programem `time` (měření doby běhu) a `cat` jako vstup pro testovacího klienta:

```
time cat /tmp/aclTest1000 | ./test-client 147.228.63.46 30000
```

Tabulka 3.2 ukazuje výsledky testů. Šifrované spojení zpomaluje komunikaci. Ztráta rychlosti je daní za vyšší bezpečnost přenosu. Výběr šifry závisí na možnostech klienta a serveru. Funkce knihovny OpenSSL se snaží při navazování spojení zvolit nejvyšší možnou variantu zabezpečení.

| Nešifrovaný přenos | | Šifrovaný přenos | |
|---|---|---|---|
| 2000 příkazů | 1 příkaz | 2000 příkazů | 1 příkaz |
| 25,508s | 0,012754s | 98,279s | 0,0491395 |
| 25,268s | 0,012634 | 98,087s | 0,0490435 |
| 25,146s | 0,012573 | 98,079s | 0,0490395 |
| 25,334s | 0,012667 | 97,918s | 0,048959 |
| 27,941s | 0,0139705 | 98,225s | 0,0491125 |
| 27,736s | 0,013868 | 98,387s | 0,0491935 |
| 27,676s | 0,013838 | 99,203s | 0,0496015 |
| 27,911s | 0,0139555 | 98,634s | 0,049317 |
| 27,601s | 0,0138005 | 98,510s | 0,049255 |
| 27,318s | 0,013659 | 98,695s | 0,0493475 |

Tabulka 3.2: Doba provádění transakcí

Další testování proběhlo na vzorku 1000 dat o velikosti 1MB příkazem:

```
put ./1mb.txt /tmp/1mb.txt
```

Tento příkaz byl 1000 krát zapsán do textového souboru copyTest1000. Opět bylo využito programů `time` pro měření doby běhu a `cat` pro přesměrování vstupu do testovacího klienta.

```
time cat /tmp/copyTest1000 | ./test-client 147.228.63.46 30000
```

Výsledky v tabulka 3.3 ukazují mírné snížení rychlosti při použití šifrování mezi klientem a bezpečnostní vrstvou.

| Nešifrovaný přenos | | Šifrovaný přenos | |
|---|---|---|---|
| 1000 MB | rychlost | 1000 MB | rychlost |
| 18m0.252s | 0,93MB/s | 18m7.893s | 0,92MB/s |
| 17m48.901s | 0,94MB/s | 18m9.766s | 0,92MB/s |
| 17m57.325s | 0,93MB/s | 19m18.303s | 0,86MB/s |
| 17m45.176s | 0,94MB/s | 18m9.827s | 0,92MB/s |

Tabulka 3.3: Rychlost přenosu 1000 x 1MB souboru

Oba testy ukazují zpomalení komunikace při použití šifrování. Použité šifry zpracovávají data po blocích (AES po 16 bytech, 3DES po 8 bytech). Pokud je blok malý je doplněn, což se projevilo hlavně v prvním testu, kde byla doplňována téměř všechna přenášená data. Následek bylo zvětšení množství přenášených dat a tedy zpomalení přenosu (pokud uvažujeme původní velikost dat).

V případě druhého testu je velikost přenášených dat mnohem větší. Zpracovávané bloky dat není nutné doplňovat až na výjimky (konce jednotlivých přenosů). Zpomalení není tak výrazné.

# 4 Závěr

Cílem bakalářské práce bylo nastudovat vlatnosti distribuovaných soborových systémů s ohledem na bezpečnostní otázky, ověřit aktuální síťové zabezpečení KIVFS serveru a komunikace s klienty. Dále bylo cílem navrhnout a realizovat bezpečnostní model pro současnou verzi KIVFS (rok 2013) a ověřit jeho funkčnost a výkonnostní parametry.

V bakalářské práci byly popsány principy a algoritmy distribuovaných systémů, algoritmy pro symetrické a asymetrické šifrování a jednocestné hashovací funkce. Byly vysvětleny funkce protokolu SSL a protokolu Kerberos. Byly navrženy a implementovány následující změny:

- Šifrování komunikace mezi klientem a bezpečnostní vrstvou pomocí protokolu SSL.
- Ověřování identity uživatele pomocí protokolu Kerberos.
- Ověřování přístupových práv uživatele v bezpečnostní vrstvě.

Zvýšení zabezpečení systému KIVFS se podařilo. Klientská aplikace předává data bezpečnostní vrstvě po šifrovaném spojení s využitím protokolu SSL. Uživatel i služba, k níž se přistupuje, je ověřována vůči třetí důvěryhodné straně protokolem Kerberos vždy při navázání spojení s bezpečnostní vrstvou. Přístupová práva uživatele jsou ověřována v bezpečnostní vrstvě. Do knihovny libkivfs jsou přidány funkce umožňující programátorům zabezpečit přenos dat pomocí protokolu SSL a ověřování uživatelů s využitím protokolu Kerberos. Protokol Kerberos byl implementován rozhraním GSSAPI jehož výhodou je možnost použít stejné funkce pro různé implementace protokolu Kerberos a další bezpečnostní protokoly.

Test šifrovaného spojení byl proveden pomocí programu `tcpdump`, který zachycuje komunikaci na síťových rozhraních, a pomocí programu `strace`, který zachycuje veškeré vstupy a výstupy procesu identifikovaného jeho pid (process ID). Cílem testu bylo prokázání, že komunikace je šifrovaná. Výsledkem testu je ověření, že komunikace mezi klientem a bezpečnostní vrstvou je šifrovaná.

Měření rychlosti provádění instrukcí a přenosu dat pomocí programů `time` a `cat` ukazují zpomalení komunikace při šifrovaném přenosu dat vůči přenosu

nešifrovanému. Cílem testu bylo zjištění zda šifrování ovlivňuje rychlost přenosu dat. Šifrování dat přenos zpomaluje v závislosti na velikosti přenášených dat. Pokud je velikost jednotlivých přenášených dat malá, zpomalení je větší, protože se ve skutečnosti přenáší data větší velikosti. Zabezpečení přenosu dat je v tomto případě důležitější a snížení rychlosti přenosu je nutná daň.

# Literatura


[1] ANDREW LEUNG, E. L. M. Scalable Security for Large, High Performance Storage Systems. *Proceedings of the 2nd ACM Workshop on Storage Security and Survivability (StorageSS 2006)*. 2006. Dostupné z: `http://ceph.com/resources/publications/`.

[2] *Systems Designer's Introduction to the Architecture*. ANSA. Dostupné z: `http://www.ansa.co.uk/ANSATech/91/RC25300.pdf`.

[3] BRAAM, P. J. The Coda Distributed File System. *Linux Journal, 50*. 1998. Dostupné z: `http://www.cs.cmu.edu/afs/cs/project/coda-www/ResearchWebPages/docdir/lj98.pdf`.

[4] DENNING, D. – SACCO, G. Timestamps in key distributed protocols. *Communication of the ACM*. 1981, 24, 8, s. 533–535.

[5] *Moore's law*. Encyclopædia Britannica Online, s. v., 2007. Dostupné z: `http://www.britannica.com/EBchecked/topic/705881/Moores-law`.

[6] FRANCO MILICCHIO, W. A. G. *Distributed Services with OpenAFS: for Enterprise and Education*. Springer, 2007. ISBN 978-3540366331.

[7] GARMAN, J. *Kerberos: The Definitive Guide*. O'Reilly Media, 2003. Dostupné z: `http://ebooksmio.com/security-related/18211-kerberos-the-definitive-guide-by-jason-garman.html`. ISBN 0596004036.

[8] *Heimdal GSS-API functions*. Heimdal. Dostupné z: `http://www.h5l.org/manual/HEAD/gssapi/group__gssapi.html`.

[9] IETF. The Internet Engineering Task Force. Dostupné z: `http://www.ietf.org/`.

[10] *Generic Security Service API Version 2 : C-bindings*. The Internet Society, 2000. Dostupné z: `http://tools.ietf.org/html/rfc2744`.

[11] *The Simple and Protected Generic Security Service Application Program Interface (GSS-API) Negotiation Mechanism*. The Internet Society, 2005. Dostupné z: `http://tools.ietf.org/html/rfc4178`.

[12] Jindřich Skupa. *KIVFS - Synchronizace a trasování požadavků*. PhD thesis, Západočeská univerzita v Plzni, Fakulta aplikovaných věd, 2012.

[13] KSHEMKALYANI, A. D. *Distributed computing: principles, algorithms, and systems*. Cambridge : Cambridge University Press, 2008. ISBN 978-0-521-87634-6.

[14] Marek Pivnička. *KIVFS: Zabezpečení, šifrování a ověření identity*. PhD thesis, Západočeská univerzita v Plzni, Fakulta aplikovaných věd, 2009.

[15] NEEDHAM, R. – SCHROEDER, M. Using encryption for authentification in large networks of computers. *Communication of the ACM*. December 1978, 12, 21.

[16] Radek Strejc. *KIVFS - Datové úložiště*. PhD thesis, Západočeská univerzita v Plzni, Fakulta aplikovaných věd, 2012.

[17] Schönefeld, M. *Refactoring of Security Antipatterns in Distributed Java Components*. University of Bamberg Press, 2010. Dostupné z: `http://books.google.cz/books?id=cUWFz3oZLyAC`. ISBN 9783923507689.

[18] *GSSAPI Programming Guide*. SUN Microsystems, 2002. Dostupné z: `http://www.shrubbery.net/solaris9ab/SUNWdev/GSSAPIPG/toc.html`.