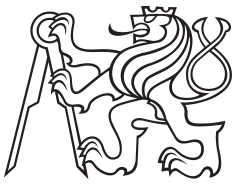
Diplomová práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačů

# Vyhledávání nad šifrovanými daty v aplikacích s architekturou klient-server

Bc. Lukáš Toman

Vedoucí: Ing. Martin Mudra
Obor: Softwarové inženýřství
Studijní program: Otevřená informatika
Květen 2020

# Poděkování

Děkuji svému vedoucímu práce panu Ing. Martinu Mudrovi za všechny jeho cenné rady a připomínky při tvorbě práce. Dále děkuji své rodině a přítelkyni za podporu během studia.


# Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu.

V Praze, 10. května 2020

# Abstrakt

Tato práce se zabývá metodami vyhledávání nad šifrovanými daty a jejich použití v klient-server architekurách. Popisuje základní principy technik pro šifrování privátním a veřejným klíčem, které umožňují vyhledávání v zašifrovaných datech. Hlavním cílem této práce je uplatnění těchto metod při návrhu a implementaci systému pro ukládání auditních logů a následném vyhledávání nad nimi.



**Klíčová slova:** šifra, vyhledávání, klient-server, auditní logy

**Vedoucí:** Ing. Martin Mudra



# Abstract

This thesis presents methods for searching on encrypted data and their usage in client-server architectures. It describes basic principles for encrypting with private and public key, which allow searching in encrypted data. Main goal of this thesis is appliaction of presented methods in design and implementation of system for storing audit logs and following searches on stored logs.



**Keywords:** cipher, search, client-server, audit logs

**Title translation:** Search on encrypted data in systems

# Obsah

# Obrázky



# Tabulky

# Část I

# Úvod

# Kapitola 1

# Úvod

## 1.1 Motivace

V dnešní době je čím dál tím více využíváno webových aplikací a cloudových služeb. S tím je spojené přesunutí dat uživatelů od samotných uživatelů na vzdálené servery. To pale představuje velké bezpečností riziko, protože data mohou obsahovat citlivé údaje a uživatelé nemají záruku, že server nezobrazí tyto data nepovolaným subjektům. Příkladem dat s citlivými údaji jsou například auditní logy, které obsahují informace o jednotlivých akcích uživatelů nějakého systému. Jednou z možností jak tento problém řešit je zašifrování dat předtím, než jsou na server odeslána. S tím dochází ke ztrátě určité funkcionality, kterou by server mohl poskytovat. Jednou z těchto funkcionalit je vyhledávání. Při použití klasických metod šifrování, by si musel uživatel všechna data stáhnout, rozšifrovat je a až poté nad nimi provádět operace vyhledávání. To znamená, že celý výpočet se přesouvá k uživateli, i přesto, že by mohl probíhat na serveru. To nemusí být vždy možné z hlediska nároků na výpočetní kapacitu uživatelova zařízení. Dalším problémem je v tomto případě zatížení spojení mezi serverem a uživatelem, protože si uživatel všechna svá data musí nejprve stáhnout. Proto existují metody šifrování, které umožnují do určité míry vyhledávat nad šifrovanými daty bez nutnosti dešifrování se zachováním disktrénosti dat.

## 1.2 Cíl práce

Hlavním cílem práce je návrh a implementace platformy, která bude sloužit pro zaznamenávání auditních logů v šifrované podobě a následné bude umožňovat vyhledávání nad uloženým logy. Platforma bude založena na vybrané technice pro vyhledávání nad šifrovanými daty. Celá platforma se bude skládat ze serverové a klientské části. Serverová část bude poskytovat rozhraní pro ukládání zašifrovaných auditních logů a pro vyhledávání v těchto záznamech pomocí klíčových slov. Klientská část bude rozdělena na klientskou knihovnu a klientskou aplikaci. Knihovna bude auditní logy šifrovat a odesílat zašifrované logy na příslušný server. Aplikace bude umožňovat zobrazovat uložené auditní logy a bude umožňovat specifikovat vyhledávací dotazy, podle kterých se zobrazí odpovídající auditní logy.

# Část II

# Analýza

# Kapitola 2

# Analýza technologií

## 2.1 Šifrování

Obecně lze říct, že šifrování je metoda, jak dvě strany mohou tajně komunikovat za přítomnosti tajného posluchače.[3] Pro ilustraci je na obrázku 2.1 takováto komunikace za přítomnosti útočníka znázorněna.

Šifra je konkrétně dvojice funkcí $E$ a $D$, kde $E$ je šifrovací funkce a $D$ dešifrovací funkce. Šifrovací a dešifrovací funkce jsou definovány nad množinami $M$, $K$ a $C$, kde $K$ je množina klíčů, $M$ je množina zpráv a $C$ je množina šifrovaného textu. Pro použití v systému auditních logů budou množiny $M$, $K$ a $C$ definovány jako $M = \{0,1\}^*$, $K = \{0,1\}^*$ a $C = \{0,1\}^*$. Šifrovací funkce $E$ je definována jako $E : K \times M \to C$ a dešifrovací funkce $D$ je dána jako $D : K \times C \to M$. Tyto funkce zároveň musí splňovat $\forall m \in M, \forall k \in K : D(k, E(k, m)) = m$.[3] V následujícím textu budou použity fiktivní uživatelé Alice a Bob, kteří se obecně používají v popisech kryptografických systémů.[11]

### 2.1.1 Asymetrické šifrování

Asymetrické šifrování nebo také šifrování s veřejným klíčem je metoda šifrování, při které se používají dva klíče - jeden veřejný a druhý privátní. Veřejný klíč je obecně známý a slouží pro šifrování zprávy. Pro dešifrování zprávy slouží


**Obrázek 2.1:** Komunikace mezi Alicí a Bobem, která je odposlouchávána

privátní klíč, který musí být držen v tajnosti.

### 2.1.2 Symetrické šifrování

Symetrické šifrování nebo také šifrování s privátním klíčem je metoda šifrování, při které se používá jeden klíč, který umožňuje jak šifrování, tak i dešifrování zprávy. Klíč musí být držen v tajnosti.

## 2.2 Auditní logy

Auditní logy jsou záznamy, které dokumentují akce jednotlivých uživatelů daného systému.[12] Uživatelem se rozumí jakýkoliv subjekt, který s daným systémem interaguje, například osoba nebo i jiný systém. Pod akcí uživatele si lze představit transakci, které se sestává z různých dílčích úkonů, které musejí být zaznamenány.

Auditní logy typicky obsahují unikátní identifikátor uživatele, unikátní identifikátor transakce, akci, kterou daný uživatel provedl, časovou stopu a další relevatní informace. Auditní logy se používají v systémech, kde je nutné zaznamenávat chování uživatelů, jako jsou například finanční nebo zdravotní systémy. [12]

V systémech, ke kterým přistupují tisíce uživatelů denně a ve kterých je

| UserId | TransactionId | Message | DateTime |
|---|---|---|---|
| 12406 | 1e93f44d-df58-4236-bcb4-a8626f72defd | Attempt to delete user 84521. | 01.01.2020 13:00 |
| 12406 | 1e93f44d-df58-4236-bcb4-a8626f72defd | User 84521 deleted. | 01.01.2020 13:00 |
| 15485 | 4d50845e-6f5e-4c6d-8b4e-a3e93ee01502 | User signed in. | 02.01.2020 14:31 |
| 15485 | 889f0375-7dd4-437d-8c73-0259736392d6 | Attempt to displayed profile of user 54123. | 02.01.2020 14:40 |
| 15485 | 889f0375-7dd4-437d-8c73-0259736392d6 | Profile of user 54123 displayed. | 02.01.2020 14:40 |
| 15485 | b2a591c5-7f28-400f-b779-cd92647fce7b | User signed out. | 02.01.2020 14:50 |

**Obrázek 2.2:** Názorná ukázka auditních logů

zároveň nutné zaznamenávat všechny akce těchto uživatelů, může dojít k produkci velkého množství auditních logů. Proto je nezbytné, aby systém auditních logů vykonával operace nad uloženými logy efektivně.

## 2.3 Klient-server architektura

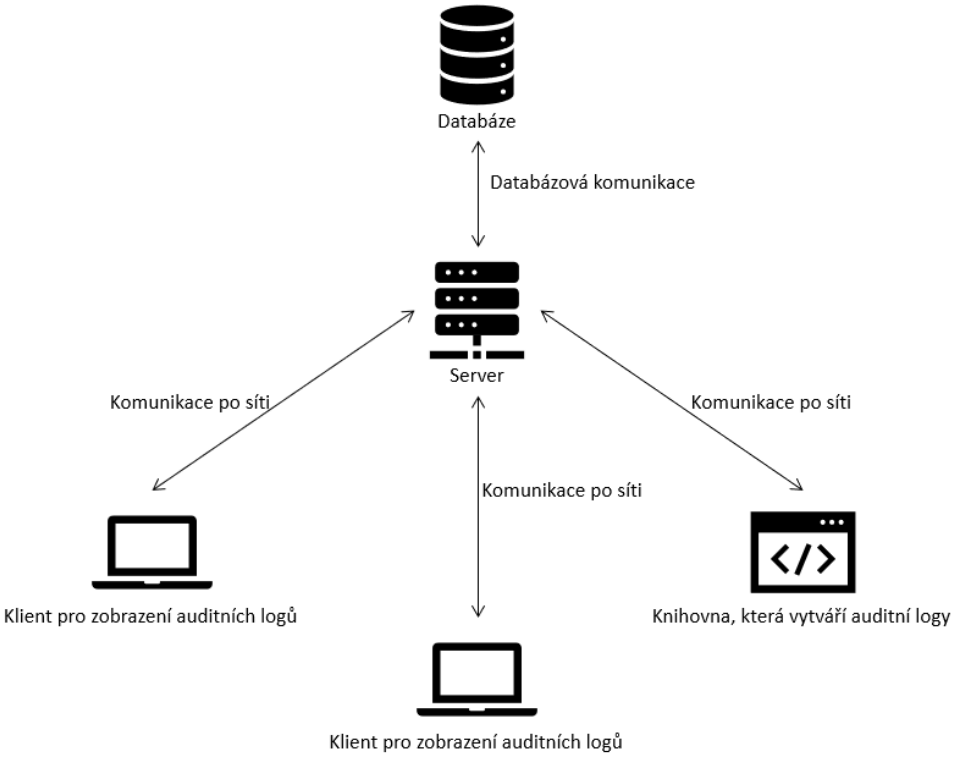
Klient-server architektura je síťová architektura aplikace, která odděluje klienta, často aplikace s grafickým rozhraním, a server, kteří spolu komunikují po počítačové síti.[20] V této architektuře slouží server jako poskytovatel služeb a klienti jako žadatelé o tyto služby.[8] Službou se v této definici rozumí množina funkcí, které server poskytuje klientům. Tato architektura má několik výhod:

- Rozdělení zpracování dat mezi několik počítačů. [7]
- Lze vertikálně i horizontálně škálovat. [7]
- Není nutná replikace dat mezi jednotlivými počítači. [7]

Hlavní nevýhodou této architektury je její centralizovanost, neboli závislost klientů na serveru. Pokud tento server selže, pak všichni klienti přestanou fungovat. V rámci systému auditních logů server poskytuje služby čítající nahrání auditního logu a vyhledávání v záznamech auditních logů. Klienti jsou dvojího typu:

- Aplikace pro zobrazení auditních logů a specifikaci vyhledávacích dotazů, pomocí kterých se bude vyhledávat.
- Knihovna, která bude vytvářet auditní logy a odesílat je na server.

Na obrázku 2.3 je zobrazena názorná klient-server architektura pro systém auditních logů.




**Obrázek 2.3:** Klient-server architektura systému auditních logů.

# Kapitola 3

# Analýza požadavků

Na základě analýzy auditních logů a klient-server architektury byly sestaveny funkční a nefunknční požadavky jak na konkrétní techniku pro vyhledávání nad šifrovanými daty, tak na celý systém.

- Funkční požadavky určují jaké chování daný systém nabízí.
- Nefunkční požadavky určují vlastnosti a omezující podmínky daného systému.

## 3.1 Funkční požadavky na techniku pro vyhledávání nad šifrovanými daty

- Technika bude umožňovat vyhledávání na serveru bez nutnosti rozšifrování daného textu.
- Technika bude umožňovat efektivní vyhledávání ve velkém počtu záznamů.
- Technika bude podporovat pokročilé metody vyhledávání, primárně vyhledávání více klíčových slov v jednom záznamu.

## 3.2 Nefunkční požadavky na techniku pro vyhledávání nad šifrovanými daty

- Technika bude bezpečná, tzn. server se nedozví nic o obsahu zašifrovaného textu.

## 3.3 Funkční požadavky na systém

- Systém bude umožnovat odesílání auditních logů na server v šifrované podobě a v této podobě je ukládat.
- Systém bude poskytovat rozhraní pro zobrazení uložených auditních logů.
- Systém bude podporovat vyhledávání auditních logů podle specifických slov v šifrované podobě.

## 3.4 Nefunkční požadavky na systém

- Systém bude muset efektivně vyhledávat ve velkém počtu auditních logů.
- Vyhledávání bude probíhat v serverové části.
- Serverové část bude snadno škálovatelná.
- Klientská část pro odesílání auditních logů na server bude implementována v podobě knihovny.
- Klientská část pro zobrazování auditních logů bude implementována v podobě desktopové aplikace.
- Server nesmí zjistit žádné informace z uložených auditních logů a z vyhledávacích dotazů, jen jestli uložený auditní log odpovídá dotazu nebo ne.

# Kapitola 4

# Analýza metod pro vyhledávání nad šifrovanými daty

V této části budou představeny možnosti vyhledávání v šifrovaných datech se zaměřením na použití v klient-server aplikaci pro správu auditních logů. Budou popsány techniky pro symetrické a asymetrické šifrování a to konkrétně:

- Sekvenční vyhledávání s použitím symetrické šifry
- Sekvenční vyhledávání s použitím asymetrické šifry
- Vyhledávání pomocí indexu vytvořeného s použitím symetrické šifry

Zároveň budou popsány jejich výhody, nevýhody a vhodnost použití těchto metod pro systém auditních logů.

## 4.1 Sekvenční vyhledávání s použitím symetrické šifry

Tato metoda byla popsána v práci "Practical Techniques for Search on Encrypted Data".[9] Je založena na sekvenčním vyhledávání slov v zašifrovaném textu. Pomocí této metody lze řešit problém zašifrování dokumentů, které chce uživatelka Alice nahrát na server a chce v nich vyhledávat za použití

jejího privátního klíče. Proto jsou tyto dokumenty zašifrované za použití symetrické šifry.

### 4.1.1 Definice problému

Alice má sadu dokumentů $D = \{D_1 \dots D_k\}$, kde každý dokument lze rozdělit na slova $W = \{W_1 \dots W_l\}$. Každé slovo může být jakýkoliv token: 64-bitový blok, slovo daného jazyka, věta apod.[9] Pro jednoduchost budeme uvažovat, že slova mají stejnou velikost. V případě, že slova mají rozdílnou velikost, pak je nutné určit nějakou velikost, na kterou se všechny slova upraví - ke kratším slovům se přidají konstantní znaky a delší slova se rozdělí na více.[9]

Alice chce nahrát tyto dokumenty na server Bob a poté zobrazit jen dokumenty, které obsahují slovo $W_i$. Bob následně může s nějakou pravděpodobností určit, zda dokument $D_j$ obsahuje slovo $W_i$. Zároveň se Bob o dokumentu $D_j$ a o slově $W_i$ dozví jen zda $D_j$ obsahuje $W_i$ a popřípadě jeho pozici a počet výskytů v dokumentu $D_j$.[9] Celý algoritmus funguje pomocí následujícího principu.

### 4.1.2 Krok 1 algoritmu: Před-šifrování slov

Nejprve Alice před-šifruje všechna slova $W = \{W_1 \dots W_l\}$ pomocí nějaké pseudonáhodné permutace $E_{k''}$, kde $k''$ je privátní klíč Alice. Tímto dostane množinu slov $X = \{X_1 \dots X_n\}$, kde $X_i = E_{k''}(W_i)$.

#### Pseudonáhodná permutace (PRP)

**PRP** je funkce $E : K_E \times Z \to Y$, kde $K_E = \{0,1\}^n$ je množina klíčů, $Z = \{0,1\}^m$ je množina vstupů a $Y = \{0,1\}^m$ je množina výstupů.[3] Řekneme, že $E$ je bezpečná PRP, jestliže pro každý algoritmus $A$ s konečným časem výpočtu je funkce $Adv(A)$, zanedbatelnou funkcí, definovanou v kapitole 4.1.2.[3] Funkce $Adv(A)$, **výhoda útočníka** $A$, proti PRP je definována jako:

$$Adv(A) : |Pr[A^{E_k, E_k^{-1}} = 1] - Pr[A^{\pi, \pi^{-1}} = 1]|$$

Kde $\pi$ reprezentuje náhodnou permutaci vybranou z množiny všech bijekcí na $Z = \{0,1\}^m$ a kde pravděpodobnosti jsou převzaty z pravděpodobností výběru $k$ a $\pi$.[9]

#### Zanedbatelná funkce

Funkce $f : \mathbb{Z}_{\leq 1} \to \mathbb{R}$, kde $\mathbb{Z}_{\leq 1}$ je množina celých čísel menších nebo rovno než 1 a $\mathbb{R}$ je množina reálných čísel, je **zanedbatelná funkce**, pokud pro všechny $c \in \mathbb{R} | c > 0$ platí:[3]

$$\lim_{n \to \infty} f(n) n^c = 0$$

### 4.1.3 Krok 2 algoritmu: Generování náhodných slov

Alice vygeneruje sekvenci hodnot $S_1 \dots S_l$ pomocí nějakého bezpečného generátoru pseudonáhodných čísel. Každá hodnota $S_i$ má velikost $n - m$, kde $n$ je velikost slova $W_i$ a $m$ je parametr, který určuje počet falešně pozitivních nálezů ve výsledném schématu.[9]

#### Generátor pseudonáhodných čísel (PRG)

**PRG** je funkce $G : K_G \to S$, kde $K_G = \{0,1\}^n$ je množina klíčů a kde $S = \{0,1\}^m$ je množina výstupů.[3] Řekneme, že $G$ je bezpečný PRG, jestliže pro každý algoritmus s konečným časem výpočtu, je funkce $Adv(A)$ zanedbatelnou funkcí, definovanou v kapitole 4.1.2.[3] Funkce $Adv(A)$, **výhoda útočníka** $A$, proti PRG je definována jako:

$$Adv(A) : |Pr[A(G(U_{K_G})) = 1] - Pr[A(U_S) = 1]|$$

Kde $U_{K_G}$, $U_S$ jsou náhodné veličiny, které mají rovnoměrné rozdělení na $U$ a $S$.[9]

### 4.1.4 Krok 3 algoritmu: Výpočet PRF

Po vygenerování hodnot $S_1 \dots S_l$ pro každou hodnotu $S_i$ Alice vypočítá hodnotu bezpečné PRF $F_{k_i}(S_i)$, která je dlouhá $m$.[9] Hlavní rozdíl mezi PRP a PRF je existence inverzní funkce pro PRP. Neboli existuje funkce $PRP^{-1}$, která pro každý vstup $j$ splňuje $j = PRP^{-1}(PRP(j))$

### Pseudonáhodná funkce (PRF)

**PRF** je funkce $F : K_F \times X \rightarrow Y$, kde $K_F = \{0,1\}^n$ je množina klíčů, $X = \{0,1\}^m$ je množina vstupů a $Y = \{0,1\}^m$ je množina výstupů.[3] Řekneme, že PRF je bezpečná, jestliže pro každý algoritmus A s konečným časem výpočtu, je funkce $Adv(A)$, definovaná v 4.1.4, zanedbatelnou funkcí, definovanou v 4.1.2.[3]

### Výhoda útočníka proti PRF

Funkce $Adv(A)$ **výhoda útočníka** $A$ proti PRF je definována jako:

$$Adv(A) : |Pr[A^{F_k} = 1] - Pr[A^R = 1]|$$

Kde $R$ reprezentuje náhodnou funkci vybranou z množiny všech zobrazení $X \rightarrow Y$ a kde pravděpodobnosti jsou převzaty z pravděpodobností výběru $k$ a $R$.[9]

### Volba klíče

Klíč $k_i$ si může Alice zvolit několika způsoby - může použít jeden klíč pro celý dokument nebo použít pro každé slovo jiný klíč, tzn. pro každé slovo $W_i$ se může rozhodnout, jestli použije ještě nepoužitý klíč $k_i$, nebo klíč, který byl již použit. Klíče také může generovat následujícím způsobem: Alice si zvolí nějakou PRF $F$ a hlavní klíč $k'$, které použije pro vygenerování klíče jako $k_i = F_{k'}(X_i)$.[9]

## 4.1.5 Krok 4 algoritmu: Šifrování

Pro kompletní zašifrování slova $W_i$ o velikost $n$ Alice použije předšifrované slovo $X_i$ a vypočítá tzv. trapdoor. Trapdoor určí jako: $T_i =< S_i, F_{k_i}(S_i) >$, kde $S_i$ a $F_{k_i}(S_i)$ jsou sekvence představené dříve. Následně pomocí $T_i$ vytvoří zašifrované slovo $C_i = X_i \bigoplus C_i$. Takto zašifrovaná slova $C_1 \dots C_l$ uloží na server.

**Algorithm 1** : *BuildIndex*

Input: $D, k_p, k_c$;
Output: $I_D$ /* The index for the document*/

$I_D = \phi$;
**for** all $w_i \in D$ **do**
  Generate a pseudo-random string $s_i$ using $G$;
  Compute trapdoor $T(w_i) = E_{k_p}(w_i)$;
  Compute ciphertext $c_i = T(w_i) \oplus \langle s_i, F_{k_c}(s_i) \rangle$;
  $I_D = I_D \cup c_i$;
**end for**
Return $I_D$;

**Obrázek 4.1:** Ukázka pseudokódu pro šifrování.[6]

**Algorithm 2** : *SearchIndex*

Input: $I_D, T(w)$;
Output: $D$ or $\phi$

**for** all $c_i \in I_D$ **do**
  **if** $c_i \oplus T(w)$ is of the form $\langle s, F_{k_c}(s) \rangle$ **then**
    Return $D$;
  **end if**
**end for**
Return $\phi$;

**Obrázek 4.2:** Ukázka pseudokódu pro vyhledávání.[6]

#### Trapdoor

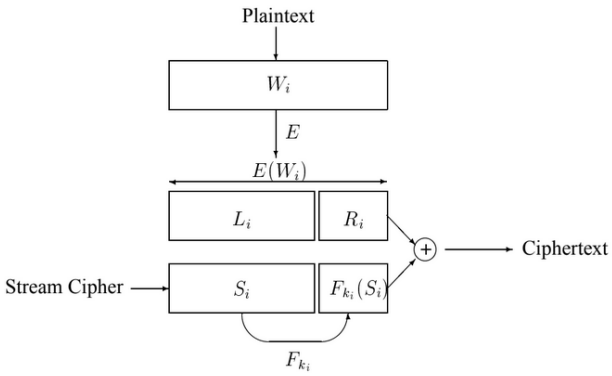
**Trapdoor** pro slovo $W$ je struktura, která umožňuje vyhledávat v šifrovaném textu slovo $W$.[9]

### 4.1.6 Vyhledávání

Pokud bude chtít Alice nad takto zašifrovaným dokumentem vyhledávat slovo $W$, pak vypočítá před-šifrované slovo $X = E_{k''}(W)$ a $k = F_{k'}(W)$ a serveru odešle slovo $X$ a klíč $k$. Server poté ověří, že daný dokument obsahuje slovo $W$ tak, že alespoň pro jedno zašifrované slovo $C_i \in \{C_1 \dots C_l\}$ platí, že $C_i \bigoplus X$ je ve formátu $< s, F_k(s) >$ pro nějaké $s \in \{0,1\}^{n-m}$.

### 4.1.7 Dešifrování

Pokud by Alice generovala klíče $k_i$ pomocí $k_i = F_{k'}(E_{k''}(W_i))$, pak by Alice nemohla dešifrovat text, protože by musela znát $E_{k''}(W_i)$.[9] Tento problém může Alice vyřešit tím, že popisovanou techniku použije jen pro vytvoření indexu, sloužícímu k sekvenčnímu vyhledávání, který uloží na server a zároveň




**Obrázek 4.3:** Schéma pro zašifrování slov.[9]

na server uloží celý text, který bude zašifrovaný nějakou běžnou technikou šifrování. To ale znamená neefektivní využití diskové paměti serveru. Dalším řešením je, že před-šifrované slovo $X_i = E_{k''}(W_i)$ rozdělí Alice na dvě $X_i =< L_i, R_i >$, kde $L_i$ má velikost $n - m$ a $R_i$ má velikost $m$. Poté místo klíče $k_i = F_{k'}(X_i)$ Alice vygeneruje klíč $k_i$ jako $k_i = F_{k'}(L_i)$. Pro dešifrování Alice vygeneruje $S_i$ pomocí pseudonáhodného generátoru a spočítá $L_i$ pomocí začátku šifrovaného slova $C_i$ o velikosti $n - m$ jako $L_i = C_{i_{n-m}} \bigoplus S_i$. Díky tomu může Alice spočítat klíč $k_i = F_{k'}(L_i)$ a tímto dešifrovat celé slovo.[9]

### 4.1.8 Zhodnocení použití pro systém auditních logů

Z hlediska funkčních požadavků splňuje technika požadavek na podporu pokročilých technik vyhledávání. Podporuje vyhledávání auditního logu obsahující slovo $W$ a zároveň slovo $W'$. Také lze vyhledat auditní logy, které obsahují slovo $W$ nebo $W'$. Je také možné vyhledat auditní logy, které obsahují slovo $W$ v daném okolí slova $W'$ a vyhledat auditní logy, které obsahují určitý počet výskytu slova $W$.

Tato metoda nesplňuje požadavek na efektivní vyhledávání ve velkém počtu záznamů. Pro vyhledávání slova $W$ v auditním logu o velikosti $n$ slov je potřeba $\mathcal{O}(n)$ šifrovacích operací.[9] To představuje vysoké zatížení serveru při vyhledávání ve veklém počtu auditních logů.

Pro systém auditních logů jsou důležité i další vlastnosti této techniky. Jednou z těchto vlastností je možnost obnovení původního textu ze zašifrovaných slov, jak bylo definováno v kapitole 4.1.7. Díky tomu odpadá nutnost uložení vlastní zprávy auditního logu, kterou by bylo jinak nutné zašifrovat

pomocí klasických metod šifrování. Velkou nevýhodou této metody je možnost provedení statistického útoku na takto zašifrované auditní logy. Server při hledání slova $W$ totiž zjistí všechny pozice slova v textu a zároveň četnost těchto slov. Tímto způsobem lze data zkompromitovat. Jednou z možností, jak tomuto zabránit, je zmenšení parametru $m$ a tímto způsobem zvýšit výskyt falešných nálezů.[9] S tím se pojí další problém a tím je samotná existence falešných nálezů. To v systému auditních logů může způsobit přetížení klienta, který auditní logy zobrazuje. Klient musí tyto falešné nálezy identifikovat a odstranit z výsledku vyhledávání.

## 4.2 Sekvenční vyhledávání s použitím asymetrické šifry

Tato metoda byla definována v práci "Public Key Encryption with Keyword Search".[2] Je založena na sekvenčním vyhledávání slov v zašifrovaném textu. Umožňuje uživatelce Alice vyhledávat zprávy, které zašifroval pomocí Alicina veřejného klíče Bob. Proto je tato technika založena na šifrování pomocí asymetrické šifry.

### 4.2.1 Definice problému

Uživatel Bob chce poslat uživatelce Alici zprávu $M$, která obsahuje citlivé údaje, a proto nechce, aby si ji mohl nepovolaný subjekt zobrazit. Bob zašifruje zprávu $M$ pro Alici jejím veřejným klíčem a přidá k ní speciální strukturu nazývanou $PEKS$ pro každé slovo, podle které může Alice danou zprávu na serveru vyhledat. Bob tedy odešle zprávu:

$$< E_{A_{pub}}(M), PEKS(A_{pub}, W_1), \ldots, PEKS(A_{pub}, W_n) >$$

Pro vyhledávání slova $W_i$ vytvoří Alice trapdoor, 4.1.5, slova $W_i$ a odešle ho na server a server pomocí trapdooru nalezne zprávy obsahující slovo $W_i$. Tato metoda je založena na použití Bilineárního zobrazení.

### Bilineární zobrazení

**Bilineární zobrazení** je zobrazení $e : G_1 \times G_2 \rightarrow G_t$, kde $G_1$, $G_2$ a $G_t$ jsou cyklické grupy stejného řádu, pro které platí[1]:

$$\forall u \in G_1, \forall v \in G_2, \forall a, b \in \mathbb{Z} : e(u^a, v^b) = e(u, v)^{ab}$$

Kde $\mathbb{Z}$ je množina všech celých čísel.

### Bilineární zobrazení pro PEKS

Bilineární zobrazení, které je použito pro tuto metodu šifrování je definováno jako $e : G_1 \times G_1 \rightarrow G_2$, kde $G_1$ a $G_2$ jsou grupy prvočíselného řádu $p$. Zároveň $e$ musí mít následující vlastnosti:

- Vypočitatelnost: pro $g, h \in G_1$ existuje polynomiální algoritmus, který vypočítá $e(g, h)$[1]
- Nedegerativnost: pokud $g$ je generátor grupy $G_1$, pak $e(g, g)$ je generátor grupy $G_2$[1]

### 4.2.2 Krok 1 algortimu: Definování grup

Alice nejprve definuje dvě grupy $G_1$ a $G_2$ s řádem $p$. Poté vygeneruje číslo $\alpha \in \mathbb{Z}_p^*$, kde $\mathbb{Z}_p^*$ je množina celých nezáporných čísel menších než $p$, a nalezne generátor $g$ grupy $G_1$.[2]

### 4.2.3 Krok 2 algortimu: Generování klíčů

Alice vygeneruje svůj veřejný klíč $A_{pub}$ jako $A_{pub} = [g, h]$, kde $h = g^\alpha$ a $A_{priv}$ jako $A_{priv} = \alpha$.[2]

### 4.2.4 Krok 3 algortimu: Generování PEKS

Bob pro vygenerování $PEKS$ klíčového slova $W$ načte veřejný klíč Alice $A_{pub}$ a nejprve vypočte $t$ jako $t = e(H_1(W), h^r)$ pro nějaké náhodné $r \in \mathbb{Z}_p^*$, kde $\mathbb{Z}_p^*$ je množina celých nezáporných čísel menších než $p$ a kde $H_1$ je hašovací funkce, 4.2.4, definována jako $H_1 : 0,1^* \rightarrow G_1$ a poté vypočte $PEKS$ jako $PEKS = [g^r, H_2(t)]$, kde $H_2$ je hašovací funkce, 4.2.4, definována jako $H_2 : G_2 \rightarrow 0,1^{log(p)}$[2]. Tímto způsobem Bob vygeneruje $PEKS$ pro všechna klíčová slova $W_1 \dots W_n$, nad kterými poté může Alice vyhledávat.

#### Hašovací funkce

**Hašovací funkce** je funkce $H : \{0,1\}^* \rightarrow \{0,1\}^n$, která slouží k mapování dat jakékoliv délky do řetězců pevně dané délky.[15]

### 4.2.5 Vyhledávání

Alice pro vyhledávání zprávy obsahující slovo $W$ odešle na server trapdoor $T_W$ slova $W$, definované jako $T_W = H_1(W)^\alpha$, kde $\alpha = A_{priv}$. Server poté pomocí trapdooru $T_W$ zjistí zda zpráva obsahuje slovo $W$ tak, že každý $PEKS$ rozdělí na dvě části $C$ a $D$, kde $C = g^r$ a $D = H_2(t)$ a pokud $H_2(e(T_w, C)) = D$, pak tato zpráva obsahuje slovo $W$.[2]

### 4.2.6 Zhodnocení použití pro systém auditních logů

Tato technika umožňuje používání pokročilejších dotazovacích technik. Umožňuje vyhledávání auditních logů obsahující slova $W$ a $W'$ a také umožňuje vyhledávat auditní logy obsahující slovo $W$ nebo slovo $W'$. Pokud jsou $PEKS$ vytvořeny ze všech slov v auditním logu a poslány ve stejném pořadí, pak lze použít i další dotazovací techink. Například nalezení auditního logu, ve kterém se slovo $W$ vyskytuje v okolí $W'$. Lze také nalézt auditní logy obsahující určitý počet výskytů daného slova. Díky tomu splňuje tato technika funkční požadavek na podporu pokročilých dotazovacíh technik.

Technika nesplňuje funčkní požadavek na efektivní vyhledávání. Její časová

náročnost pro ověření, zda auditní log obsahuje dané slovo, je $\mathcal{O}(n)$[2], která je dána počtem $n$ klíčových slov. V případě vyhledávání ve velkém počtu auditních logů by tímto způsobem byl velmi vytížen server.

Následující vlastnosti je také nezbytné zvážit při použití v systému auditních logů. Pokud jsou vytvořeny $PEKS$ vytvořeny ze všech slov v auditním logu, pak je i tato technika náchylná k statistickému útoku stejně jako v metodě sekvenčního vyhledávání s použitím privátního klíče. Technika neumožňuje obnovení původního textu z PEKS, proto je nutné původní text auditního logu zašifrovat klasickou metodou a uložit společně s PEKS. S tím se pojí i větší nároky na úložiště.

## 4.3 Bezpečný index

Tato metoda byla popsána v práci "Secure Indexes".[5] Je založena na vytvoření indexu složeného z klíčových slov, podle kterých lze poté Alice pomocí svého privátního klíče daný dokument nalézt. Tento index je proto vytvořen pomocí privátního klíče.

### 4.3.1 Definice problému

Alice má dokument $D$ a klíčová slova $W = \{W_1 \dots W_l\}$, pomocí kterých vytvoří bezpečný index $ind$, který nahraje s dokumentem $D$ na server Bob. Bob poté může v čase $\mathcal{O}(1)$[5] s určitou pravděpodobností ověřit, zda dokument $D$ obsahuje slovo $W_i$. Nic jiného se Bob o dokumentu $D$ a o slově $W_i$ nedozví.[5] Metoda je založena na následujícím schématu.

### 4.3.2 Krok 1 algortimu: Generování klíče

Alice si zvolí bezpečnostní parametr $s$. Poté pomocí $s$ vytvoří PRF,4.1.4, $F : \{0,1\}^n \times \{0,1\}^s \to \{0,1\}^s$ a primární klíč $K = \{k_1 \dots k_r\}$, kde $k_i \in \{0,1\}^s$.[5]

### 4.3.3 Krok 2 algortimu: Výpočet trapdoor

Pro každé klíčové slovo $W_i$, které chce Alice do indexu vložit, vypočítá trapdoor,4.1.5, $x = \{x_1 \dots x_r\}$, kde $x_1 = F(W_i, k_1) \dots x_r = F(W_i, k_r)$.[5]. Pokud by Alice následně vložila $x$ do indexu, pak by bylo možné provést statistický útok nad množinou indexů.[5] Proto je nutné dále tyto trapdoory upravit tak, aby byly pro daný index unikátní a neobjevovaly se v indexu jiném.

### 4.3.4 Krok 3 algortimu: Zabezpečení proti statistickému útoku

Pro zabezpečení indexu proti statistickému útoku Alice ke každému trapdooru $x$ přidá unikátní $id$ dokumentu a vygeneruje sekvenci $y = \{y_1 \dots y_r\}$, kde $y_1 = F(id, x_1) \dots y_r = F(id, x_r)$.[5] Takto vygenerované sekvence se poté vloží do Bloom filtru.

#### Bloom filtr

**Bloom filtr** je struktura reprezentována polem bitů o velikosti $m$. Na začátku jsou všechny prvky pole nastaveny na 0. Každý Bloom filtr ma zároveň $n$ hašovacích funkcí, definované v kapitole 4.2.4, $h_1 \dots h_n$, kde $h_i = \{0,1\}^* \rightarrow [0, m-1]$. Pro vložení prvku $p$ do Bloom filtru se nastaví na pozicích $h_1(p) \dots h_n(p)$, bity na 1. Pro ověření, že prvek $l$ je obsažen v Bloom filteru, se zkontroluje, že na všech pozicích $h_1(l) \dots h_n(l)$, je nastavena 1. Pokud ano, prvek je obsažen v Bloom filtru. Falešné nálezy mohou nastat, protože každá pozice může být nastavena na 1 několika různými vkládanými prvky.[5]

### 4.3.5 Krok 4 algortimu: Úplné zabezpečení

Alice vytvoří horní hranici $u$ počtu slov v dokumentu $D$. Poté spočítá počet unikátních slov $v$ v dokumentu $D$ a vloží $(u - v) * r$ jedniček náhodně a rovnoměrně do Bloom filtru pro jeho lepší zabezpečení proti útokům.[5]

**Algorithm 3** : $BuildIndex_{BF}$

Input: $D, k_p, k_c, h_1, \ldots, h_r$
Output: $BF_D$ /* The index for the document*/

$BF_D = \phi$;
**for** all $w_i \in D$ **do**
  Compute trapdoor $T(w_i) = E_{k_p}(w_i)$;
  Compute string $x_i = E_{k_c}(\langle id(D), T(w_i)\rangle)$
  **for** $j = 1$ to $r$ **do**
    compute bit-position $b_j = h_j(x_i)$;
    set $BF_D[b_j] = 1$;
  **end for**
**end for**
Return $BF_D$;

**Obrázek 4.4:** Ukázka pseudokódu pro vytvoření indexu.[6]

**Algorithm 4** : $SearchIndex_{BF}$

Input: $BF_D, T(w), k_c, h_1, \ldots, h_r$
Output: $D$ or $\phi$

Compute $x = E_{k_c}(\langle id(D), T(w)\rangle)$;
**for** j=1 to r **do**
  **if** $BF_D[h_j(x)] \neq 1$ **then**
    Return $\phi$;
  **end if**
**end for**
Return $D$;

**Obrázek 4.5:** Ukázka pseudokódu pro vyhledávání.[6]

### 4.3.6 Vyhledávání

Pro vyhledávání slova $W$ v zabezpečeném indexu Alice vytvoří trapdoor $x = \{x_1 \ldots x_r\}$, kde $x_1 = F(W_i, k_1) \ldots x_r = F(W_i, k_r)$. Tento trapdoor odešle na server, který poté pomocí $id$ dokumentu vygeneruje $y = \{y_1 \ldots y_r\}$, kde $y_1 = F(id, x_1) \ldots y_r = F(id, x_r)$. Pokud jsou v Bloom filtru obsažena všechna slova $y_1 \ldots y_r$, pak je slovo nalezeno.[5]

### 4.3.7 Zhodnocení použití pro systém auditních logů

Tato technika z části splňuje funkční požadavek na efektivitu vyhledávání. Pro ověření zda auditní log obsahuje sekvenci $y$ z kapitoly 4.3.4 je potřeba $\mathcal{O}(1)$.[5] Problém je v samotném generování sekvence $y$, která je vygenerována pomocí trapdooru $x$ a identifikátoru daného auditního logu. Kvůli tomuto by byla metoda neefektivní pro použití v systému auditních logů, protože generování sekvence $y$ pro všechny $id$ auditních logů by bylo pro server náročné.

Technika splňuje funkční požadavek na podporu pokročilejších dotazovacích technik vyhledávání. Lze vyhledat auditní logy obsahující slova $W$ a $W'$ a také auditní logy, které obsahují slovo $W$ nebo $W'$. Funkcionalitu vyhledávání

auditních logů, které obsahují slovo $W$ v nějakém počtu lze docílit úpravou algoritmu šifrování - ke každému slovu $W$, předtím než se zašifruje, přidá počet slov $W$, které se objevily v textu před ním.

Nevýhodou je, že ze zabezpečeného indexu nelze obnovit původní text, ten proto musí být uložen v zašifrované podobě společně s indexem. S tím se pojí větší nároky na velikost úložiště v systému auditních logů. Další nevýhodou z hlediska nároků na celý systém auditních logů je výskyt falešných nálezů, které jsou dány podstatou Bloom filtrů. To znamená větší nároky na klientskou aplikaci, které bude muset identifikovat falešné nálezy a odstranit je z nalezených auditních logů. Další nevýhodou, která přímo neodporuju požadavkům na techniku, je nemožnost vyhledávání auditních logů, které obsahují slovo $W$ v okolí $W'$.

# Část III

# Návrh

# Kapitola 5

# Výběr techniky

Techniky pro sekvenční vyhledávání definované v kapitolách 4.1 a 4.2 jsou obtížně použitelné pro systém auditních logů, kvůli časové složitosti daných algoritmů. Pro nalezení auditního logu, který obsahuje slovo $w$ v množině všech auditních logů $L = l_1 \dots l_n$, kde každý auditní log $l$ obsahuje slova $W = w_1 \dots w_m$, kde $m_i$ je počet slov v auditním logu $l_i$, je nutné $\sum_0^n m_i$ operací. To nesplňuje funkční požadavek na efektivní vyhledávání ve velkém počtů záznamů definovaný v kapitole 3.1.

Použití techniky bezpečného indexu definované v kapitole 4.3 je z hlediska efektivity vhodnější. Je potřeba jen $\mathcal{O}(1)$ operací pro ověření, zda daný auditní log obsahuje slovo $W$. Celková časová náročnost pro vyhledávání je tedy dána počtem auditních logů $n$.

Technika bezpečného indexu vytváří unikátní index pro každý auditní log, protože ke každému slovu přidává unikátní identifikátor vytvářeného auditního logu. Podle schématu proto musí server pro vyhledání daného slova nejprve načíst identifikátory všech auditních logů a díky nim vytvořit sekvence pro vyhledávání. Tímto způsobem by byla zahlcena výpočetní kapacita serveru a neefiktivní vyhledávání ve velkém množství auditních logů.

Z důvodů uvedených výše byla pro systém auditních logů vybrána technika bezpečného indexu, kterou ale není možné použít v její čisté formě. Proto byla technika upravená pomocí následujícího způsobu.

## 5.1 Úprava algoritmu

Při vytváření indexu pro daný auditní log bude vynecháno šifrování s použitím identifikátoru dokumentu, jak bylo definováno v kapitole 4.3.4. Proto indexy obsahující stejná slova budou obsahovat 1 na stejných pozicích. Na základě této změny bude upraven algoritmus pro vyhledávání následujícím způsobem:

- Klient vloží všechna vyhledáváná slova do nového indexu.
- Tento index odešle na server.
- Server porovná uložené indexy s obdrženým indexem a pokud uložený index má nastaveno 1 na všech místech jako obdržený index, pak vrátí příslušný log .

Tímto způsobem server nebude zahlcen výpočtem sekvencí s identifikátorem auditního logu a může použít pro porovnání indexů například regulární výrazy. Nevýhodou této úpravy je možnost použití útoku zvaného frekvenční analýza.

### 5.1.1 Útok frekvenční analýzou na nové schéma

Všechny auditní logy obsahující stejné slovo budou mít nastaveny na stejných pozicích 1. Díky tomu lze provést frekvenční analýzu následujícím způsobem. Pokud útočník bude znát procentuální zastoupení jednotlivých informací obsažených v záznamech auditních logů, může seřadit počty výskytů 1 na jednotlivých pozicích sestupně a poté díky tomu zjistit, co jednotlivé auditní logy představují.

Pro příklad mějme tři auditní logy $A$ = "*User XXX logged in.*", $B$ = "*User YYY logged in.*" a $C$ = "*User ZZZ deleted.*. Systém je nastaven tak, že pro log $A$ a $B$ vytvoří index se slovem *login* a pro log $C$ index se slovem *deletion*. Pro $A$ a $B$ vypadá index například takto 100010001 a pro $C$ takto 01010010. Útočník ví, podle Kerckhoffsova principu, že nejčastěji auditovanou akcí je přihlášení. Proto může díky této informaci odhadnout, že auditní logy $A$ a $B$ obsahují informace o přihlášení. V reálné implementaci to není takto přímočaré, protože z podstaty Bloom filtru může být určité místo nastaveno na 1 několika různými slovy. Proto může například nastat, že index pro auditní log $D$ bude obsahovat 1 na stejných místech jako $A$ a $B$, ale slovo *login* obsahovat nebude. Proto je i vhodné zvolit parametry Bloom filtru tak,

aby docházelo k častějším falešným nálezům, a tím i lepšímu zabezpečení. Další zmatení útočníka je docíleno přidáním náhodných slov do Bloom filtru při úplném zabezpečení definovanám v kapitole 4.3.5.

### Kerckhoffsův princip

**Kerckhoffsův princip** je pravidlo v kryptografii, které říká, že kryptografický systém má být bezpečný i když je všechno, až na použitý kryptografický klíč, známo.[18]

## 5.2 Výběr parametrů pro Bloom filtr

Důležitým parametrem Bloom filtru je poměr falšených nálezů $fp$, který Bloom filtr vrací. Tento parametr je závislý na velikosti $m$ Bloom filtru a počtu $r$ hašovacích funkcí, které jsou použity při vkládání slova do filtru.

Pro odvození parametrů je nejprve důležité definovat pravděpodobnost, že po vložení $n$ slov do Bloom filtru je bit na určitě pozici roven 0. Tato pravděpodobnost je dána jako $(1 - \frac{1}{m})^{rn}$. Tento výraz lze přepsat na $e^{-rn/m}$ za použití $\lim_{k\to\infty}(1 - \frac{1}{k})^k = e^{-1}$ [13]. Proto pravděpodobnost výskytu falešného nálezu lze vyjádřit $(1 - e^{-rn/m})^r$. V případě bezpečného indexu lze vzít proměné $m$ a $n$ jako konstaty [5] a tedy po zderivování a úpravě je minimum pro $r$ dáno jako $r = (\ln 2)(m/n)$,[5] Pro takto zvolené $r$ je pravděpodobnost falešného výskytu dána rovnicí $fp = (1/2)^r$. Po úpravě a vyjádření $r$ vychází $r = -\log_2(fp)$. Následně je nutné zvolit hodnotu parametru $n$, které představuje kapacitu Bloom filtru. Kapacita Bloom filtru je předpokládaný počet slov, které se do Bloom filtru vloží. Díky $r$ a $n$ lze určit velikost $m$ Bloom filtru pomocí rovnice $m = nr/\ln 2$.

V tabulce 5.1 jsou ukázky poměrů falešných nálezů a jejich počty hašovacích funkcí. V tabulce 5.2 jsou znázorněny velikosti Bloom filtru v závislosti na poměru falešných nálezů a kapacitě.

| Poměr falešných nálezů | Počet hašovací funkcí |
|---|---|
| 0.4 | 1 |
| 0.3 | 2 |
| 0.1 | 3 |
| 0.01 | 7 |
| 0.001 | 10 |

**Tabulka 5.1:** Ukázka poměr falešných nálezů a počtu hašovacích funkcí

| Poměr falešných nálezů/Kapacita | 10 | 20 | 30 | 50 | 100 |
|---|---|---|---|---|---|
| **0.4** | 14 | 29 | 43 | 72 | 144 |
| **0.3** | 29 | 58 | 87 | 144 | 289 |
| **0.1** | 43 | 87 | 130 | 216 | 433 |
| **0.01** | 101 | 202 | 303 | 505 | 1010 |
| **0.001** | 144 | 289 | 433 | 721 | 1443 |

**Tabulka 5.2:** Ukázka velikost Bloom filtru pro poměry falešných nálezů a počtu vkládaných slov

## 5.3 Výběr parametrů pro algortimus bezpečného indexu

V předchozí části byly definované vzorce pro výpočet počtu potřebných klíčů $r$ a velikosti Bloom filtru $m$ jako $r = -\log_2(fp)$ a $m = nr/\ln 2$. Proto před vytvořením samotného indexu je nezbytné, aby byla definována požadovaná pravděpodobnost výskutu falšených nálezů $fp$ a počet slov $n$, které budou do indexu vloženy. Pro platformu auditních logů je důležité, aby každý log měl nastavené stejné parametry nebo měl parametry z dané množiny možných parametrů. Kdyby každý auditní log měl nastaveno vlastní $fp$ a $n$, pak by bylo vyhledávání náročné na výpočet - klient by musel podle upraveného schématu, definovaném v kapitole 5.1, pro každý $fp$ a $n$ vytvořit jiný index. Proto je potřeba další úpravy techniky pro vytvoření bezpečného indexu.

### 5.3.1 Konfigurace

V systému bude definována množina konfigurací, konkrétně dvojic $fp$ a $n$, podle kterých se budou indexy vytvářet. Tyto konfigurace budou uloženy na serveru, který bude klientům poskytovat rozhraní pro jejich stažení.

Klient vytvářející index nejprve stáhne ze serveru všechny možné konfi-

gurace. Ze stažených konfigurací vybere jednu, která bude odpovídat počtu klíčových slov vytvářeného auditního logu a klientovu nastavení preferované míry falešných nálezů. Pomocí konfigurace vytvoří index a spolu s konfigurací ho odešle na server. Ten tento index uloží a naváže na vybranou konfiguraci. Díky tomuto propojení pak může klient při vyhledávání vytvořit index, který odpovídá parametrům indexu uloženého.

Systém bude umožňovat přidávání nových konfigurací. Uložené konfigurace nebude možné měnit nebo je mazat z důvodu navázání na indexy jednotlivých logů. Pokud by se konfigurace smazala, nešlo by auditní logy navázané na danou konfiguraci nalézt, nebo by muselo dojít u auditních logů svázaných s danou konfigurací k navázání na jinou konfiguraci a přegenerování indexu.

## 5.4 Výpočet pozice při vkládání do Bloom filtru

V klasickém použití Bloom filtru je pro vložení slova zapotřebí $r = -\log_2(fp)$ hašovacích funkcí. Pro použití Bloom filtru v bezpečném indexu jsou tyto hašovací funkce nahrazeny za jednu hašovací funkcí, která pro hašování používá různé klíče. Proto je zapotřebí $r = -\log_2(fp)$ klíčů, které se použijí při hašování. Z toho důvodu nelze použít obyčejnou hašovací funkci, která vytváří haš jen na základě vstupního slova. Na základě těchto požadavků bylo rozhodnuto o použití algoritmu z množiny HMAC-SHA.

### 5.4.1 HMAC

HMAC je typ autentizačního kódů zprávy, při kterém se používá kryptografická hašovací funkce v kombinaci s klíčem.[16]

#### Kryptografická hašovací funkce

Kryptografická hašovací funkce je hašovací funkce vhodná pro použití v kryptografii.[14] Ideální kryptografická funkce má následující vlastnosi:[14]

- Je deterministická, neboli pro zprávu vytvoří vždy stejný haš

- Není možné nalézt zprávu, bez použití útoku hrubou silou, která má daný haš
- Není možná nalézt dvě rozdílné zprávy, bez použití útoku hrubou silou, které mají stejný haš
- Malá změna ve zprávě vygeneruje zcela nový haš, který nemá nic společného s hašem původní zprávy
- Všechny hodnot haše jsou stejně pravděpodobné

### 5.4.2 SHA

SHA je množina kryptografických hašovacích funkcí.[22] SHA je tvořena následujícími algoritmy:[22]

- SHA-1 s hašem dlouhým 160 bitů
- SHA-224 s hašem dlouhým 224 bitů
- SHA-256 s hašem dlouhým 256 bitů
- SHA-384 s hašem dlouhým 384 bitů
- SHA-512 s hašem dlouhým 512 bitů

### 5.4.3 Výpočet pozice z haše

Pro vytvoření pozice pro Bloom filtru z haše vygenerovaném pomocí HMAC-SHA je nutné z tohoto haše vytvořit číslo, které bude reprezentovat pozici v Bloom flitru. Z definice 5.4.1 je každý bit haše nezávislý na ostatních. Díky tomu lze z hodnoty haše určit jakoukoliv sekvenci bitů, jejíž délka je dána počtem bitů, které reprezentují číslo na dané platformě, například 32. Tato sekvence bitů je poté převedena na číslo a z takto vytvořeného čísla se pomocí operace modulo vytvoří odpovídající pozice v Bloom filtru.

### 5.4.4 Vytvoření klíčů pro HMAC-SHA

Pro vložení slova do Bloom filtru je zapotřebí $r = -\log_2(fp)$ klíčů. Tyto klíče nelze předem definovat, protože daný počet klíču je znám až po načtení

konfigurace ze serveru. Proto je z praktického hlediska nutné použití funkce pro odvození klíče. Díky této funkci bude stačit znát jen jeden hlavní klíč, ze kterého se vytvoří potřebný počet klíčů.

### Funkce pro odvození klíče

Funkce pro odvození klíče je kryptografická hašovací funkce, která ze zadaného hesla, hlavního klíče, vytvoří jeden nebo více klíčů.[19]

## 5.5 Výběr konkrétní hašovací funkce

Bylo rozhodnuto, že v systému bude implementováno několik hašovacích funkcí z množiny HMAC-SHA a to SHA-1, SHA-256, SHA-384, SHA-512. Konkrétní typ hašovací funkce bude specifikován při nasazení a bude uložen na serveru. Server bude poskytovat typ hašovací funkce společně s jednotlivými konfiguracemi Bloom filtru.

# Kapitola 6

# Návrh architektury

## 6.1 Knihovna

Knihovna bude obsahovat funkcionalitu potřebnou k vytvoření záznamu auditního logu. Bude poskytovat rozhraní pro specifikaci zprávy auditního logu a klíčových slov. V případě, že žádná klíčová slova nebudou zadána, knihovna rozdělí zadanou auditní zprávu na jednotlivá slova, která použije jako klíčová.

Pro vytvoření samotného auditního logu knihovna nahraje ze serveru platné konfigurace Bloom filtru a typ hašovací funkce. Poté vybere konkrétní konfiguraci podle počtu klíčových slov a preferované míry falešných nálezů. Preferovaná míra falešných nálezů bude specifikována v nastavení knihovny. Výběr konkrétní konfigurace je popsán v kapitole 6.1.2. Z konfigurace, typu hašovací funkce a klíčových slov vytvoří knihovna samotný index pomocí Bloom filtru.

Následně knihovna zašifruje pomocí algoritmu AES zprávu auditního logu a všechna klíčová slova. Klíčová slova je nutné uložit pro pozdější odstranění falešných nálezů při vyhledávání. Pro hašování a šifrování použije knihovna hlavní klíč, který bude určený v nastavení knihovny. Ze zašifrované zprávy, indexu, konfigurace a zašifrovaných slov vytvoří samotný auditní log, který odešle na server.

### 6.1.1 AES

AES je standardizovaný algoritmus používaný k šifrování v informatice. Jedná se o symetrickou blokovou šifru.[10]

### 6.1.2 Výběr konkrétní konfigurace

Knihovna vybere konkrétní konfiguraci následujícím způsobem:

1. Pokusí se nalézt ty konfigurace, které odpovídají preferované pravděpodobnosti výskytu falešných nálezů.
2. Pokud žádná konfigurace neodpovídá preferované hodnotě, pak určí konfigurace s mírou výskytu falšených nálezu, které jsou nejbližší nižší.
3. Pokud stále nejsou žádné konfigurace vybrány, pak vybere konfigurace s mírou výskytu falšených nálezu, které jsou nejbližší vyšší.
4. Z těchto konfigurací vybere tu, která má kapacitu rovnou počtu klíčových slov.
5. Pokud taková konfigurace neexistuje, pak vybere konfiguraci s kapacitou nejlbižší vyšší.
6. Pokud taková konfigurace stále neexistuje, pak vybere konfiguraci s kapacitou nejbližší nižší.

## 6.2 Tenký klient

Tenký klient bude implementován jako desktopová aplikace, která bude uživatelům umožňovat přístup k systému auditních logů. Aplikace bude poskytovat jednoduché GUI, skrz které budou uživatelé interagovat se systémem.

Aplikace bude uživateli zobrazovat jednotlivé záznamy auditních logů. Protože se očekává že auditních logů bude vysoký počet, bude muset aplikace umožňovat zobrazení po stránkách. Aplikace proto nejdříve načte počet všech auditní logů ze serveru a poté zobrazí uživateli ovládací prvek, pomocí kterého budou mít uživatelé možnost zobrazit jen určitý počet logů a danou stránku.

Dále bude aplikace uživateli poskytovat rozhraní pro specifikaci klíčových slov, podle kterých se bude vyhledávat v záznamech auditních logů. Při vyhledávání nejprve aplikace načte všechny možné kofigurace Bloom filtru a typ používané hašovací funkce. Poté pro každou konfiguraci vytvoří index. Tyto indexy odešle na server, který vrátí nalezené auditní logy. Tyto auditní logy následně aplikace rozšifruje. Poté aplikace porovná klíčová slova jednotlivých auditních logů s vyhledávanými slovy a odstraní falešné nálezy. Poté zobrazí nalezené auditní logy uživateli.

Aplikace bude umožňovat zobrazení všech konfigurací Bloom filtru uložených na serveru a typ používané hašovací funkce. Uživatel také bude mít možnost skrz aplikaci specifikovat nové konfigurace, které aplikace odešle na server.

## 6.3 Server

Server bude poskytovat rozhraní pro komunikaci s jednotlivými klienty. Bude umožňovat nahrání nového auditního logu, stažení seznamu auditních logů a vyhledávání v záznamech auditních logů. Také bude umožňovat klientům načítat a vytvářet konfigurace Bloom filtru. Společně s konfiguracemi Bloom filtru bude poskytovat typ hašovací funkce, které se v systému používá.

Nové auditní logy a konfigurace bude validovat a poté je ukládat do databáze. Vyhledávání auditních logů bude probíhat pomoc regulárních výrazů, které server vytvoří z přijatých indexů.

## 6.4 Databáze

Databáze bude sloužit k ukládání samotných auditních logů, konfigurací a typu hašovací funkce. Bude zprostředkovávat vlastní vyhledávání v záznamech auditních logů.

## 6.5 Komunikace

Klient a knihovna bude se serverovou částí komunikovat pomocí protokolu HTTPS, který používá protokol HTTP společně s protokolem SSL nebo TLS pro bezpečný, šifrovaný přenos.[17] Pro jednotlivé akce budou použity následující HTTP metody:

- Pro uložení auditního logu bude použita metoda POST. V těle dotazu bude serializovaný auditní log ve formátu JSON.
- Pro vyhledávání v záznamech auditních logů bude použita metoda POST. V URL parametrech bude specifikováno stránkování, začátek a počet auditních logů. V těle dotazu budou serializovány indexy s odpovídajícími konfiguracemi ve formátu JSON. Dotaz na server bude obsahovat několik indexů, podle kterých se bude vyhledávat. Tyto indexy musejí být serializovány do těla, protože v případě serializace přímo do URL jako parametry, pak by bylo možné přesáhnout maximální délku URL danou pro webový server. POST metoda pro vyhledávání byla zvolena, protože metoda GET nemá specifikováno chování pro tělo [4] a některé implementace serverů ignorují tělo GET dotazu nebo ignorují takovéto dotazy.
- Pro listování v záznamech auditních logů bude použita metoda GET, kde se v rámci URL parametrů specifikuje stránkování.
- Pro stáhnutí konfigurací bude použita metoda GET.
- Pro vytvoření konfigurace bude použita metoda POST. V těle dotazu bude serializovaná konfigurace, konkrétně kapacita a poměr falešných nálezů, ve formátu JSON.

### 6.5.1 Stavové kódy

Server bude vracet následující stavové kódy:

- 200 OK - pro validní a zpracované dotazy
- 400 Bad Request - pro dotazy, které obsahují chybu a nelze je zpracovat
- 404 Not Found - pro neexistující adresu
- 500 Internal Server Error - pokud dotaz nemůže být dokončen z důvodu neočekávané chyby

# Část IV

# Implementace

# Kapitola 7

# Implementace

## 7.1 Výběr platformy

Na základě zhodnocení požadavků bylo rozhodnuto, že serverová část bude implementována v jazyce C# ve verzi 8.0 na platformě ASP.NET Core ve verzi 3.0. Klientská část pro zobrazování a specifikaci vyhledávání bude implementována v jazyce C# ve verzi 8.0 na platformě .NET Core pomocí technologie WPF. Část pro odesílání auditních logů bude v podobě knihovny implementována v jazyce C# ve verzi 8.0 na platformě .NET Core. Jako databáze pro ukládání auditních logů byla zvolena Elasticsearch databáze ve verzi 7.7.

## 7.2 Nástroje pro vývoj

Celý systém byl vyvíjen na platformě Windows 10 64-bit za použití nástroje Microsoft Visual Studio 2019.

## 7.3 Server

Serverová část byla rozdělena do několik komponent kvůli deduplikaci kódu a umožnění výměny jednotlivých částí. Používá nativní IoC kontejner pro ASP.Net Core pro vkládání závislostí do jednotlivých komponent. Dále používá knihovnu pro logování fungování systému, zejména logování neočekávancých chyb, knihovnu Serilog[1]. Jednotlivé komponenty serveru:

- AuditLogs.Server.Core - knihovna, které obsahuje hlavní logiku serveru
- AuditLogs.Server - ASP.NET Core API aplikace představující rozhraní serveru
- AuditLogs.Structures - knihovna, které obsahuje společné modely primárně určené pro komunikaci

## 7.4 Knihovna

Knihovna byla implementována jako dll pro její snadnou přenositelnost a vložení do aplikací, které potřebují funkcionalitu auditního logování. Knihovna byla také rozdělena do několika komponent. Komponenty knihovny:

- AuditLogger - vlastní knihovna, která obsahuje funkcionalitu pro vytváření auditních logů a jejich posílání na server
- SecureIndex - knihovna, která obsahuje funkcionalitu samotného bezpečného indexu
- AuditLogs.Structures - knihovna, které obsahuje společné modely primárně určené pro komunikaci
- AuditLogs.Shared - knihovna, která obsahuje funkcionalitu společnou pro několik komponent
- Commons - knihovna, které obsahuje obecnou funkcionalitu

[1] https://serilog.net/

## 7.5 Klientská aplikace

Klientská aplikace je implementována jako desktopová aplikace pro Windows. Je také rozdělena do komponent. Závislosti jednotlivých komponent jsou řešeny pomocí IoC kontejneru SimpleInjector[2]. Pro logování fungování systému je použita knihovna[3]. Komponenty aplikace:

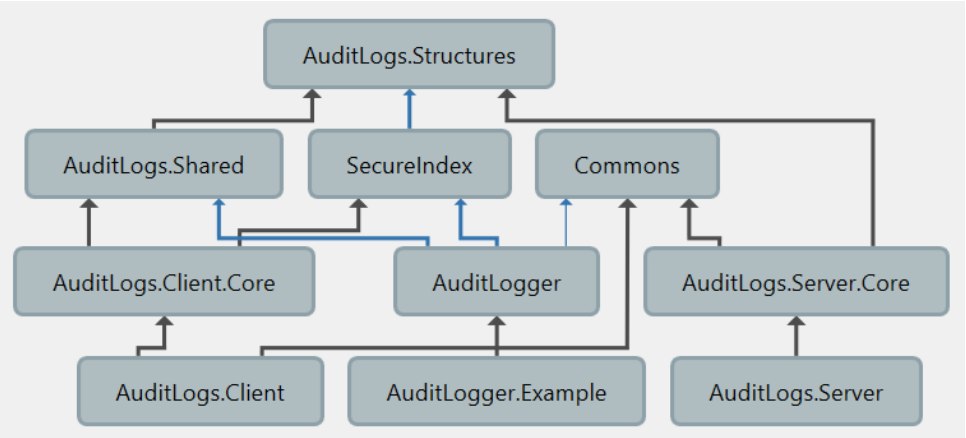
- AuditLogs.Client.Core - knihovna obsahující hlavní funkcionalitu desktopového klienta
- AuditLogs.Client - WPF aplikace, které slouží pro zobrazení auditních logů
- AuditLogs.Shared - knihovna, která obsahuje funkcionalitu společnou pro několik komponent
- SecureIndex - knihovna, která obsahuje funkcionalitu samotného bezpečného indexu
- Commons - knihovna, které obsahuje obecnou funkcionalitu

## 7.6 Testovací aplikace

Dále byly vytvořeny dvě testovací aplikace:

- AuditLogger.Example - ASP.Net Core MVC aplikace, která slouží pro názorné předvedení funkcionality AuditLogger knihovny
- SecureIndex.Test - Konzolová aplikace pro testování bezpečného indexu
- AuditLogs.Test - Konzolová aplikace pro testování celého systému

[2]https://simpleinjector.org/
[3]https://serilog.net/




**Obrázek 7.1:** graf závislostí jednotlivých komponent

# Část V

# Testování

# Kapitola 8

# Testování

Pro testovaní byly vytvořeny konzolové aplikace SecureIndex.Test a AuditLogs.Test. Testy v aplikaci SecureIndex.Test jsou zaměřené na otestování vlastního bezpečného indexu a použitých hašovacích funkcí. Testy v aplikaci AuditLogs.Test jsou zaměřené na otestování celého systému, konkrétně zda systém vrací korektní výsledky při vyhledávání. Testy byly provedeny na počítači s těmito parametry:

- Procesor: Intel Core i5-6200U 2.3GHz
- Paměť: 8 GB DDR4
- Operační systém Windows 10 64-bit

## 8.1 Testování distribuce hašovacích funkcí

Pro použití bezpečného indexu bylo zapotřebí otestovat zda implementace jednotlivých hašovacích funkcí se chovají podle předpokladu, tzn. že vypočítávájí pozice pro Bloom filtr čistě náhodně. Vypočtené pozice by měly mít rovnoměrné rozdělení. Pro otestování zda hašovací funkce splňují tyto požadavky byla použita metoda chí-kvadrát test dobré shody.

Pro test bylo vygenerováno 100000 náhodných slov, ze kterých poté byla vygenerována pozice v Bloom filtru, který měl velikost 1000. Z předpokladu,

že hašovací funkce má rovnoměrné rozdělení vychází, že každý index se měl objevit ve výsledku 100krát. Pro test byly určeny hodnota hladiny významnosti jako 0,05 a hodnota stupně volnosti na 999. Chí-kvadrát test dobré shody byl použit pro otestování nulové hypotézy: "Hašovací funkce má rovnoměrné rozdělení". Pro vypočtení chí-kvadrát testu dobré shody byla použita knihovna Accord framework.[1]

Pro všechny implementace haš funkcí nebyla nulová hypotéza zamítnuta. Jinými slovy se daná hašovací funkce pravděpodobně chová podle předpokladu. Je ale možné, že se při provedení testu pro více slov projeví odchylka od předpokladu.

### 8.1.1 Chí-kvadrát test dobré shody

Chí-kvadrát test dobré shody je statistický test, který je používán pro ověření hypotéz. Test umožňuje ověřit, zda má náhodná veličina dané rozdělení pravděpodobnosti.[21]

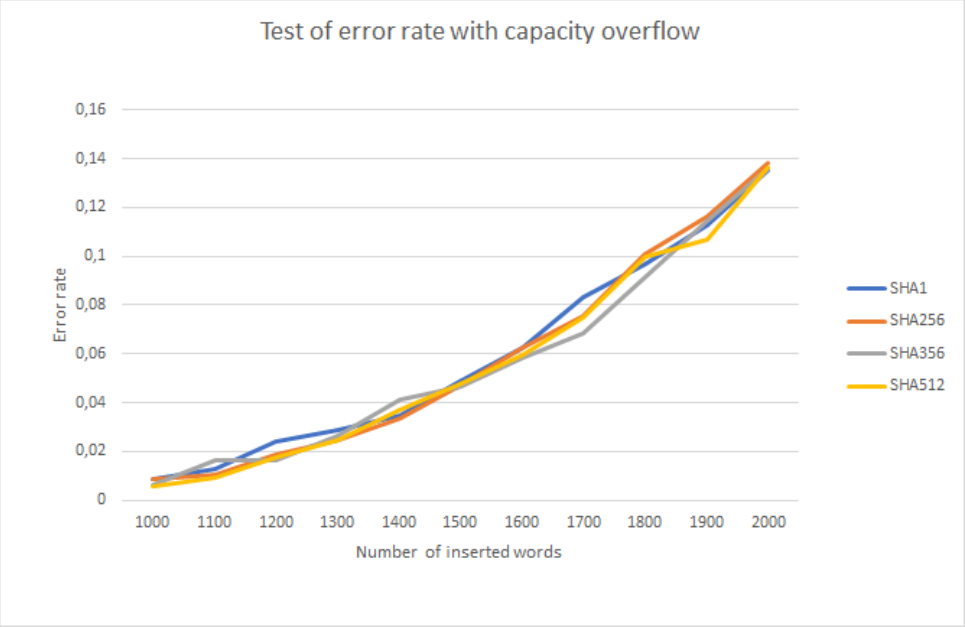
## 8.2 Testování poměru falešných nálezů při vložením více slov než je kapacita Bloom filtru

Pomocí tohoto testu bylo otestováno zvýšení míry falešných nálezů bezpečného indexu při vložení více slov než je jeho kapacita. Míra falešných nálezů je dána pomocí vzorce $(1 - e^{-rn/m})^r$ defionvaném v kapitole 5.2.

Pro účely testování byla vytvořena konfigurace, které udávala kapacitu 1000 a míru falešných nálezů 0.01. Poté byly s touto konfigurací vytvořeny bezpečné indexy a vloženo do nich 1000, 1100,..., 1900, 2000 slov. Poté byl pro každý index otestován počet falešných nálezů pro 1000 náhodných slov, které nebyly do indexu vloženy.

Na grafu 8.1 lze vidět rychlost růstu míry výskytu falešncýh nálezů při přesáhnutí kapacity daného Bloom filtru. Pro otestované parametry Bloom filtru při vložení 2000 slov vychází míra falešných nálezů 0.13. To zanemená, že se míra výskytu falešných nálezu pro 2000 zvýšila zhruba 13krát a otestované hodnoty opdovídají předpokládaným hodnotám růstu.

[1] http://accord-framework.net/




**Obrázek 8.1:** Graf míry falešných nálezů v závislosti na počtu vložených slov nad kapacitu

| Hašovací funkce | Počet falešných nálezů | Míra falešných nálezů |
|---|---|---|
| SHA-1 | 126 | 0.0126 |
| SHA-256 | 117 | 0.0117 |
| SHA-384 | 121 | 0.0121 |
| SHA-512 | 106 | 0.0106 |

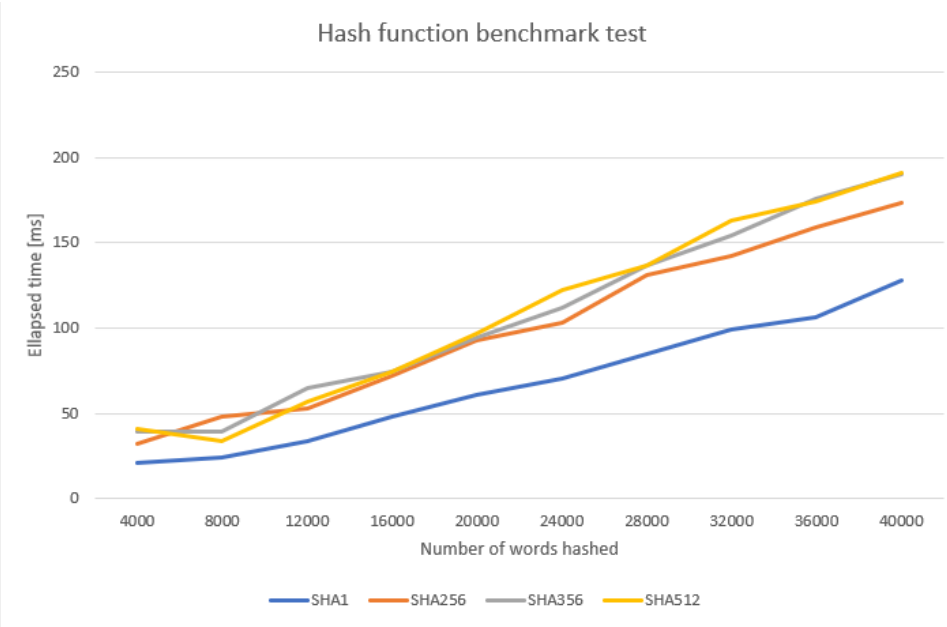
**Tabulka 8.1:** Výsledné počty falešných nálezů

## 8.3 Testování míry falešných nálezů při vložením stejného počtu slov jako je kapacita Bloom filtru

Pomocí tohoto testu byl otestován poměr falešných nálezů pro jednotlivé implementace hašovacích funkcí.

Pro tento test byl vytvořen bezpečný index s kapacitou 10000 slov a mírou falešných nálezů 0.01. Poté do něj bylo vloženo 10000 náhodných slov a následně proběhlo 10000 dotazů, zda index obsahuje náhodné slovo, které do indexu vloženo nebylo.

Z testu vyplynula tabulka 8.1, ve které lze vidět, že výsledné hodnoty odpovídají nastavené hodnotě míry falešných nálezů.




**Obrázek 8.2:** Graf rychlosti výpočtu jednotlivých hašovacích funkcí

## 8.4 Benchmark test hašovacích funkcí

Tento test měl za úkol otestovat rychlost hašování jednotlivvých hašovacích funkcí. Slova, pro které byl vypočítán haš, měly velikosti 10, 20, 30, 50 znaků. Byly provedeny testy postupně pro 1000, 2000, . . . , 9000, 10000 náhodných slov pro každou velikost. Z testování vyplynul graf 8.2. Z grafu je patrné, že implementace SHA-1 je nejrychlejší.

## 8.5 Testování validity vrácených záznamů při vyhledávání

Tento test měl za úkol otestovat systém jako celek. Protože nebyly k dispozici reálné logy, na kterých by šlo systém otestovat, byla stažena kniha Romeo and Juliet. Tato kniha je dostupná online v knihovně Project Gutenberg[2]. Každý řádek knihy sloužil jako zpráva auditního logu a všechny slova na řádku byla použita pro vytvoření indexu. Takto vytvořené auditní logy byly uloženy do databáze a poté bylo pro každé slovo otestováno, zda vyhledávání vrací validní výsledky. Tento test proběhl v pořádku a lze tedy říct, že systém se chová tak jak má.

[2]http://www.gutenberg.org/ebooks/1112

# Část VI

# Závěr

# Kapitola 9

# Závěr

## 9.1 Zhodnocení splnění cílů

Cílem této práce bylo analyzovat možnosti vyhledávání nad šifrovanými daty, navrhnout a implementovat platformu, která bude umožňovat zaznamenávání auditních logů v šifrované podobě a poté nad nimi vyhledávat. V části II byly analyzovány techniky pro vyhledávání nad šifrovanými daty a jejich vhodnost pro použití pro systém auditních logů. V části III byla konkrétní technika vybrána a upravena pro systém auditních logů. Poté byl celý systém v části IV implementován a v části V otestován. Cíl této práce lze tedy považovat za splněný.

## 9.2 Návrhy na zlepšení

### 9.2.1 Způsob ukládání indexů

V této chvíli se indexy ukládají jako celé textové řetězce tvořené 1 a 0. To není z hlediska efektivity využití paměti optimální. Proto by bylo vhodné upravit ukládání a ukládat je v zkomprimované podobě. Například nahradit dlouhé sekvence řetězce, které obsahují jen stejný znak, za dvojici znak a délka sekvence. Pro příklad následující index "100001111110001"by byl nahrazen

za 1{5}0{4}1{6}0{3}1. Díky tomu lze ušetřit u velkých indexů značný počet paměti. Pokud by se použilo komprimování indexů, musel by se upravit i vyhledávácí mechanismu.

### 9.2.2 Vyhledávání pomocí regulárních výrazů

Ve stávajícím řešení funguje vyhledávání auditních logů na úrovni databáze za pomocí regulárních výrazů. To z hlediska složitosti, jak časové tak paměťové, není úplně optimální. Proto by v budoucnu bylo vhodné udělat analýzu možností vyhledávání bez použití regulárních výrazů.

# Přílohy

# Příloha A

## Slovník

**AES** Advanced encryption standard. vi, 37, 38

**GUI** Graphical User Interface. 38

**HMAC** keyed-Hash Message Authentication Code. vi, 33–35

**HTTP** Hypertext Transfer Protocol. 40

**HTTPS** Hypertext Transfer Protocol Secure. 40

**IoC** Inversion of Control. 44, 45, 64

**JSON** JavaScript Object Notation. 40

**PEKS** Public-key encryption with keyword search. vi, 19–22

**PRF** Pseudonáhodná funkce. v, 15, 16, 22

**PRG** Generátor pseudonáhodných čísel. 15

**PRP** Pseudonáhodná permutace. 14, 15

**SHA** Secure Hash Algorithm. vi, 33–35, 51

**SSL** Secure Sockets Layer. 40

**TLS** Transport Layer Security. 40

**URL** Uniform Resource Locator. 40

# Příloha B

# Literatura


[1] J. Bethencourt. Intro to bilinear maps. *https://people.csail.mit.edu/alinush/6.857-spring-2015/papers/bilinear-maps.pdf*, 2015. Online; accessed 1.5.2020.

[2] D. Boneh, G. Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. *Advances in Cryptology - Eurocrypt*, 2004.

[3] D. Boneh and V.Soup. *A graduate course in applied cryptography.* 2017.

[4] R. Fielding and J. Reschke. Hypertext transfer protocol (http/1.1): Semantics and content. *https://tools.ietf.org/html/rfc7231*, 2014. Online; accessed 1.5.2020.

[5] E.-J. Goh. Secure indexes. *Proc. of IEEE SRSP*, 2004.

[6] H. Hacıgumus, B. Hore, B. Iyer, and S. Mehrotra. Search on encrypted data. *Secure data management in computer decentralized systems: 383-425*, 2007.

[7] C. Kambalyal. 3-tier architecture. 2010.

[8] S. Shakirat. Client-server model. *IOSR Journal of Computer Engineering*, 16, 2014.

[9] D. Song, D. Wagner, and A. Perrig. Practical techniques for search on encrypted data. *Proc. of IEEE SRSP*, 2000.

[10] Wikipedia contributors. Advanced encryption standard. *https://cs.wikipedia.org/wiki/Advanced_Encryption_Standard*, 2020. Online; accessed 1.5.2020.

[11] Wikipedia contributors. Alice and bob. *https://en.wikipedia.org/wiki/Alice_and_Bob*, 2020. Online; accessed 1.5.2020.

[12] Wikipedia contributors. Audit trail. *https://en.wikipedia.org/wiki/Audit_trail*, 2020. Online; accessed 1.5.2020.

[13] Wikipedia contributors. Bloom filter. *https://en.wikipedia.org/wiki/Bloom_filter*, 2020. Online; accessed 1.5.2020.

[14] Wikipedia contributors. Cryptographic hash function. *https://en.wikipedia.org/wiki/Cryptographic_hash_function*, 2020. Online; accessed 1.5.2020.

[15] Wikipedia contributors. Hash function. *https://en.wikipedia.org/wiki/Hash_function*, 2020. Online; accessed 1.5.2020.

[16] Wikipedia contributors. Hmac. *https://en.wikipedia.org/wiki/HMAC*, 2020. Online; accessed 1.5.2020.

[17] Wikipedia contributors. Https. *https://en.wikipedia.org/wiki/HTTPS*, 2020. Online; accessed 1.5.2020.

[18] Wikipedia contributors. Kerckhoffs's principle. *https://en.wikipedia.org/wiki/Kerckhoffs's_principle*, 2020. Online; accessed 1.5.2020.

[19] Wikipedia contributors. Key derivation function. *https://en.wikipedia.org/wiki/Key_derivation_function*, 2020. Online; accessed 1.5.2020.

[20] Wikipedia contributors. Klient-server. *https://cs.wikipedia.org/wiki/Klient-server*, 2020. Online; accessed 1.5.2020.

[21] Wikipedia contributors. Pearson's chi-squared test. *https://en.wikipedia.org/wiki/Pearson-s_chi-squared_test*, 2020. Online; accessed 1.5.2020.

[22] Wikipedia contributors. Secure hash algorithm. *https://cs.wikipedia.org/wiki/Secure_Hash_Algorithm*, 2020. Online; accessed 1.5.2020.

# Příloha C

# Uživatelská příručka

Pro spuštění celého systému je vhodné použít nástroj Microsoft Visual Studio 2019. Popsaný postup spuštění jednotlivých komponent bude popsán pro tento nástroj.

## C.1 Nasazení serveru

Pro nasazení serveru je zapotřebí mít zprovozněnou ElasticSearch[1] databázi. Po spuštění ElasticSearch databáze je potřeba definovat indexy v databázi a nahrát potřebné konfigurace. To lze například pomocí nástroje Kibana[2]. Skrz tento nástroj lze provést příkazy uvedené v souboru ESInit.txt. Po inicializaci databáze je nutné upravit konfiguraci server v souboru AuditLogs.Server.appsettings.json, konkrétně hodnotu "DatabaseConfiguration.Url"a nahradit ji za adresu na které běží ElasticSearch. Poté stačí spustit projekt AuditLogs.Server.

[1] https://www.elastic.co/downloads/
[2] https://www.elastic.co/downloads/

## C.2 Spuštění desktopového klienta

Pro spuštění desktopového klienta je nejdříve nutné upravit konfigurační soubor AuditLogs.Client.App.config a konkrátně změnit hodnotu "AuditLogServer"na adresu serveru. Poté již lze spustit projekt AuditLogs.Client.

## C.3 Ukázka knihovny

Pro vytváření nových auditních logů byl vytvořen projekt AuditLogger.Example.
Pro spuštění je nutné nastavit v konfiguračním souboru AuditLogger.Example.appsettings.json hodnotu "AuditLogConfiguration.AuditLogServer"na adresu serveru. Poté stačí spustit projekt AuditLogger.Example a vytvářet nové logy.

## C.4 Použití knihovny

Pro použití knihovny v externím sytému je nutné naimprotovat knihovnu do systému a zaregistrovat všechny její komponenty v IoC kontejneru. Poté stačí vytvořit instanci třídy AuditLogger, která zpřístupňuje rozhraní knihovny.

# Příloha D

## Obsah přiloženého CD

- \DP - složka obsahující zdrojové soubory systému
- \ESInit.txt - textový soubor, který obsahuje inicializační skripty pro databázy
- \Thesis - složka obsahující zdrojové kódy tohot dokumentu
- \Vyhledávání_nad_šifrovanými_daty.pdf - tento dokument ve formátu pdf.

# ZADÁNÍ DIPLOMOVÉ PRÁCE

## I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Toman** Jméno: **Lukáš** Osobní číslo: **406314**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Studijní obor: **Softwarové inženýrství**

## II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

**Vyhledávání nad šifrovanými daty v aplikacích s architekturou klient-server**

Název diplomové práce anglicky:

**Search on encrypted data in systems**

Pokyny pro vypracování:

Nastudujte doporučenou literaturu.
Analyzujte možnosti vyhledávání nad šifrovanými daty v aplikacích využívající architekturu client-server, kde server slouží k ukládání šifrovaných dat formou úložiště. Diskutujte technologická omezení vyplývající z použitých postupů.
Implementujte „proof-of-concept“ systém využívající architektury client-server prokazující možnosti vyhledávání nad šifrovanými daty. Zejména se zaměřte na možnosti vyhledávání v auditních informacích na serverové straně. Implementaci důsledně zdokumentujte a otestujte.

Seznam doporučené literatury:

[1] D. Song and D. Wagner and A. Perrig, Practical Techniques for Search on Encrypted Data. In Proc. of IEEE SRSP, 2000
[2] E-J. Goh, Secure Indexes. Technical report 2003/216, In IACR ePrint Cryptography Archive, (2003).
[3] D. Boneh, G. di Crescenzo, R. Ostrovsky, and G. Persiano, Public Key Encryption with Keyword Search. In Advances in Cryptology - Eurocrypt 2004
[4] Hakan Hacigümüs, Bijit Hore, Balakrishna R. Iyer, Sharad Mehrotra, Search on Encrypted Data. Secure Data Management in Decentralized Systems 2007
[5] B. Waters, D. Balfanz, G. Durfee, and D. Smetters. Building an encrypted and searchable audit log. In Proceedings of the 11th Network and Distributed System Security (NDSS) Symposium, 2004

Jméno a pracoviště vedoucí(ho) diplomové práce:

**Ing. Martin Mudra, Katedra počítačové grafiky a interakce**

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **09.07.2019** Termín odevzdání diplomové práce: **22.05.2020**

Platnost zadání diplomové práce: **19.02.2021**

| Ing. Martin Mudra<br>podpis vedoucí(ho) práce | podpis vedoucí(ho) ústavu/katedry | prof. Mgr. Petr Páta, Ph.D.<br>podpis děkana(ky) |
|---|---|---|

## III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. _____
Datum převzetí zadání

_____
Podpis studenta