VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

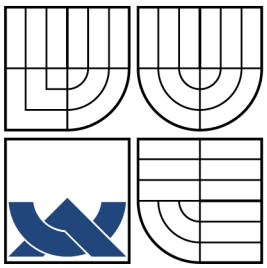
SYSTÉM KLIENT-SERVER ZALOŽENÝ NA OPENSSL

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

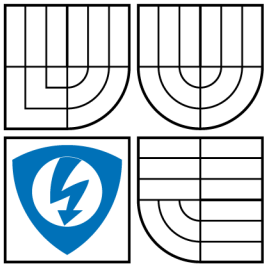
AUTOR PRÁCE
AUTHOR

BC. JAROSLAV KOHOUT

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND
COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

SYSTÉM KLIENT-SERVER ZALOŽENÝ NA OPENSSL

CLIENT-SERVER SYSTEM BASED ON OPENSSL

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — BC. JAROSLAV KOHOUT
AUTHOR

VEDOUCÍ PRÁCE — ING. JAN MALÝ
SUPERVISOR

BRNO 2008

Zde vložíte list zadání

Z důvodu správného číslování stránek

Zde vložíte první list licenční smlouvy

Z důvodu správného číslování stránek

Zde vložíte druhý list licenční smlouvy

Z důvodu správného číslování stránek

ABSTRAKT

Cílem práce je nastudovat možnosti systému OpenSSL v prostředí PHP a implementovat zabezpečení pomocí tohoto rozšíření na příkladu systému klient-server, který je určen k ukládání citlivých dat. Je bezpodmínečně nutné aby celý systém byl příkladně zabezpečený proti celé škále útoků, vedoucích ke získání informací.

KLÍČOVÁ SLOVA

Zabezpečení PHP, OpenSSL.

ABSTRACT

Aim of this diploma thesis is study of possibilities of OpenSSL extension in PHP environment and its implementation in securing client-server system example. This system will be use to store confidential data. Whole system will be exemplar of securing against scale of attacks leads to gain private data.

KEYWORDS

Securing PHP, OpenSSL.

KOHOUT. J *Systém klient-server založený na OpenSSL. Brno: VUT Brno. Fakulta elektrotechniky a komunikačních technologií. Ústav telekomunikací, 2008. 60 s., 3 s. příloh. Diplomová práce. Vedoucí práce byl Ing. Jan Malý.*

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Systém klient-server založený na OpenSSL“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne ....

(podpis autora)

Poděkování

Tímto bych chtěl poděkovat vedoucímu práce Ing. Janu Malému, který mi při řešení problémů během psaní této práce vždy vyšel vstříc a pomáhal mi ji zdokonalovat konstruktivní kritikou.

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

PHP Hypertextový Preprocesor – Hypertext Preprocessor

DES Standard Kódování Dat – Data Encrypting Standard

3DES Trojnásobné Standardní Kódování Dat – Triple Data Encrypting Standard

AES Pokročilé Kódování Dat – Advanced Encrypting Standard

RSA Iniciály autorů – Rivest, Shamir, Adleman

NIST Národní Institut pro Standardy a Technologii – National Institute of Standards and Technology

Rijndael Zkratka příjmení autorů – Rijmen and Daemon

MD5 Message-Digest algorithm 5

SHA Secure Hash Algorithm

NSA Národní Bezpečnostní Agentura – National Security Agency

SPN Substitučně-Permutační Síť – Substitution-Permutation Network

PHP Hypertextový Preprocesor – Hypertext Preprocessor

HTML HyperTextový značkovací jazyk – HyperText Markup Language

SSL Vrstva Bezpečných Socketů – Secure Socket Layer

TLS Bezpečná Transportní Vrstva – Transport Layer Security

PKI Infrastruktura veřejného klíče – Public Key Infrastructure

RFC Žádost o komentáře – Request For Comments

ASP Active Server Pages

JSP Java Server Pages

XSS Cross Site Scripting

HTTP Hypertectový Přenosový Protokol - Hypertext Transfer Protocol

HTTPS Hypertectový Přenosový Protokol přes SSL - Hypertext Transfer Protocol over Secure Socket Layer

ÚVOD

V dnešní době, kdy přes internet využíváme nejrůznějších služeb, přes email až po elektronické bankovnictví je čím dál tím více důležité chránit údaje, které jsou na serverech uloženy. Na nich jsou údaje o uživateli, jako například login a heslo. Pokud by se někdo nepovolaný dostal k těmto údajům, mohl by například přes internetové bankovnictví operovat s penězi na našem účtu. To je samozřejmě nepřípustné a správné zabezpečení těchto dat by mělo být na prvním místě.

Cílem diplomové práce je nastudovat zabezpečení PHP (Hypertextový Preprocesor – Hypertext Preprocessor) pomocí rozšíření OpenSSL k ukládání citlivých dat. Využijeme přitom kryptologii. Kryptologie je věda, zabývající se utajováním informací, stojící na pomezí matematiky a informatiky. Kryptologii můžeme rozdělit do dvou hlavních částí, a to kryptografie, což je disciplína o utajení informace, a kryptoanalýza, coby disciplína zabývající se postupy, jak informaci utajit. Vzhledem k tomu, že dnešní počítače mají vysoký výpočetní výkon, nemusí nás omezovat složitost některých matematických operací pro šifrování či dešifrování. Má to také tu výhodu, že případný pokus o rozšifrování dat může být pro útočníka mnohem složitější.

OpenSSL je kryptografická knihovna, která poskytuje implementaci algoritmů, které jsou v oboru nejvíce doporučované. Obsahuje algoritmy pro zašifrování jako DES (Standard Kódování Dat – Data Encrypting Standard), AES (Pokročilé Kódování Dat – Advanced Encrypting Standard) a RSA (Iniciály autorů – Rivest, Shamir, Adleman). Zaměřením kryptografie je zabezpečit důležitá data před útokem či zneužitím. Nejčastějším médiem, přes které dochází k útoku, je počítačová síť.

Knihovna OpenSSL disponuje prostředky k symetrickému, tak i asymetrickému šifrování. V případě symetrického to znamená použití stejného klíče jak pro šifrování, tak i pro dešifrování. V asymetrickém používáme veřejný a soukromý klíč. Pomocí veřejného klíče se informace zašifrují a soukromým se rozšifrují. Obě mají své výhody i nevýhody, které popíši v této práci.

Vytvořím víceuživatelský systém, do kterého se budou moci uživatelé registrovat. Každému z uživatelů vygeneruji vlastní klíče potřebné k šifrování a dešifrování. Do systému budu ukládat některá citlivá uživatelská data, která budou šifrována veřejným klíčem a uložena do databáze. Po přihlášení uživatele do systému uvidí svá data rozšifrovaná svým privátním klíčem. Administrátor tohoto systému si bude moci zobrazit všechny údaje uživatelů.

1 TEORETICKÝ ÚVOD

V této kapitole se zmíním o oblastech, které budou v tomto projektu stěžejní.

1.1 Šifrování (Kryptografie)

Šifrování neboli kryptografie je podoborem vědního oboru kryptologie. Ta je rozdělena do dvou hlavních vědních disciplín, a to kryptoanalýzy a již zmíněné kryptografie.

Kryptografie ([5]) vznikla z všeobecně známé nutnosti chránit informace, studuje šifrovací algoritmy, kryptografické nástroje apod. Slovo kryptografie pochází z řečtiny – kryptós znamená skrytý a gráphein znamená psát. Někdy je tento pojem obecněji používán pro vědu o čemkoli spojeném se šiframi jako alternativa k pojmu kryptologie. Nejvýznamějšími pojmy v kryptografii ([7]) jsou:

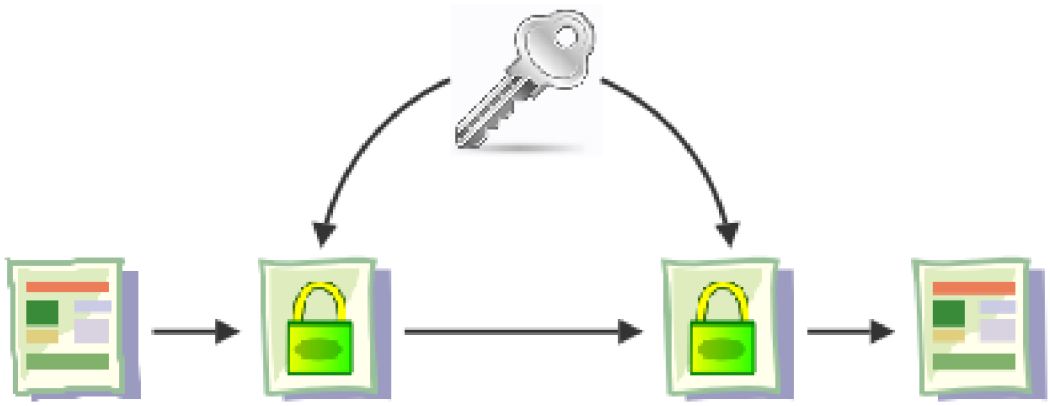
- **šifra** - tímto označujeme kryptografický algoritmus, který převádí čitelnou zprávu neboli *prostý text* do nečitelné podoby *šifrovaný text*
- **klíč** - jedná se o tajnou informaci pomocí které informace zašifrujeme a bez které nelze text přečíst
- **hashovací funkce** - jedná se o předpis pro výpočet kontrolního součtu
- **symetrická šifra** - používá pro šifrování i dešifrování tentýž klíč
- **asymetrická šifra** - používá tzv. veřejný klíč pro šifrování a tzv. soukromý klíč pro dešifrování

Kryptoanalýza je vlastně opakem kryptografie. Zabývá se luštěním šifer, což v praxi znamená zjištění tajného klíče, kterým je informace zašifrovaná. Její význam je čím dál tím větší, vzhledem k teoretickým slabinám běžně používaných šifer. Tento termín se obecně používá pro prolamování kódu.

Kryptografie se po staletí vyvíjela k větší složitosti zároveň s lidskou civilizací a mnohokrát ovlivnila běh dějin. Zejména utajení či vyzrazení strategických vojenských informací mělo a stále má zásadní vliv (vzpomeňme například na německý šifrovací stroj Enigma). Složitost šifry úzce souvicí s bezpečným přenosem informací a se schopností protivníka šifru rozbít.

1.1.1 Symetrické šifrování

Symetrické šifrování je založeno na principu jednoho klíče, viz 1.1. Příkladem symetrického šifrování je DES (Standard Kódování Dat – Data Encrypting Standard). Ten byl vyvinutý v roce 1974 v USA americkou vládou ve firmě IBM a v roce 1977 byl přijat jako standard. Je často používaný. Výhodou symetrických šifrování je rychlost algoritmu. Problémem je však distribuce klíče. Musíme zajistit, aby se klíče nezmocnil někdo nepovolaný a musí se předem domluvit tajný klíč. Často se používají společně s asymetrickými šiframi. Symetrické šifry dělíme na blokové a proudové podle toho, jak pracují s prostým textem určeným k šifrování. Blokové pracují s pevně stanoveným počtem bitů, který se nazývá blok. Proudové šifry pracují postupně s jednotlivými znaky a jejich transformace se s časem mění.



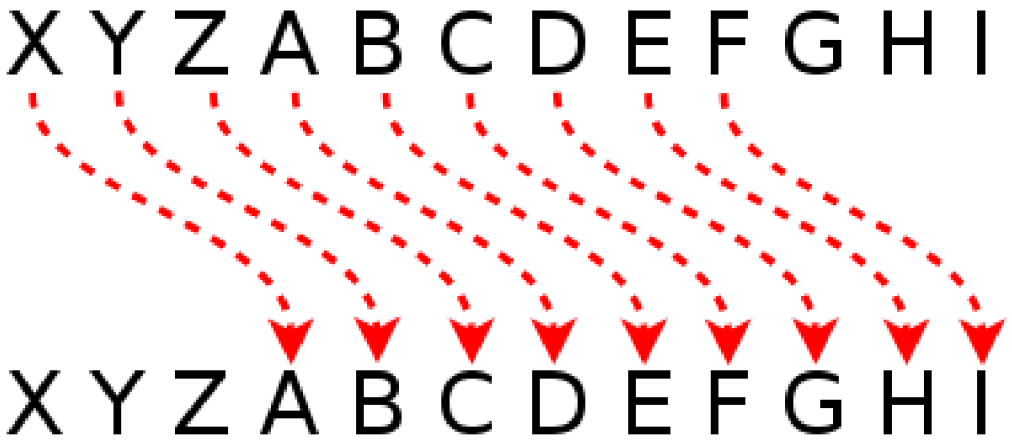
Obr. 1.1: *Symetrické šifrování*

Pro příklad symetrické šifry ukáži tzv. Caesarovu Šifru ([7]) jejíž princip můžeme vidět na obr. 1.2 a je velice jednoduchý: je provedeno abecední posunutí písmen a klíčem je číslo o kolik písmeno posuneme.

Podle tohoto klíče (posunutí písmen o 3 pozice) by například prostý text: „CAESAR“ byl zašifrován na šifrový text: „FDHVDU“.

DES

DES ([5, 7, 10]) se řadí mezi algoritmy Feistelova typu a je založen na postupné aplikaci relativně jednoduchých transformací, které umožňují vytvořit složitý kryp-



Obr. 1.2: *Caesarova šifra - posunutí o 3 písmena*

tografický algoritmus. Počátky šifrovací normy DES se datují na konec 60.let. Byl vyvinut společností IBM, původně pro společnost Loyd. Vzhledem ke svému zdokonalování se stal kandidátem na standard šifrování dat v USA a v roce 1977 byl také jako standard přijat. Přestože byl DES nejprve určen pro ochranu „citlivých“ vládních dat, rozšířil se do komerční sféry a používá se dodnes na celém světě.

Šifra kryptuje text po 64 bitových blocích. Z toho 56 bitů je použito pro vlastní kódování a 8 bitů je kontrolní součet. Algoritmus DES používá dvě základní kryptografické techniky: konfúze(též zmatení, konfúze; angl. confusion = výsledek šifrování, u něhož nelze predikovat, jakou změnu šifrovaného textu vyvolá byť jen malá změna otevřeného textu. Existuje tedy složitá funkční závislost mezi zašifrovaným textem a párem klíč - otevřený text, což zvyšuje bezpečnost proti kryptoanalýze.

DES má 16 rund, což znamená, že aplikuje stejnou kombinaci zmíněných technik na blok otevřeného textu 16krát. Algoritmus pracuje s osmi S-boxy (substituční boxy). Každý S-box je reprezentován tabulkou o 4 řádcích a 16-ti sloupcích. Každý má 6-ti bitový vstup a 4 bitový výstup (tedy 48 bitový vstup do S-boxů a 32 bitový výstup). Vstupem každého S-boxu je 4 bitové číslo. 6 vstupních bitů S-boxu určuje, ve kterém řádku a sloupci tabulky je třeba hledat výstup.

V současné době je již DES prolomen. Pro zvýšení jeho bezpečnosti byl navržen model 3DES (Trojnásobné Standardní Kódování Dat – Triple Data Encrypting Standard). Ten funguje tak, že provede třikrát algoritmus DES pokaždé s jiným klíčem. Toto řešení je bezpečné, ale nevýhodou tohoto řešení je, že se hodí jen pro některé aplikace a je příliš pomalý.

AES

Vznik standardu AES ([5, 7, 11]) je spojen s prolomením DES (1997). Je znám také pod názvem Rijndael (Zkratka příjmení autorů – Rijmen and Daemon). Tím, že byl DES prolomen vznikl požadavek na vytvoření nového (lepšího a bezpečnějšího) standardu. Vzhledem k tomu, že tímto (DES) standardem se řídila vláda USA a její dokumenty byly zašifrovány tímto algoritmem, vypsal NIST (Národní Institut pro Standardy a Technologii – National Institute of Standards and Technology) výběrové řízení do kterého se přihlásilo 15 kandidátů. Nakonec vyhrál návrh belgických, výše zmíněných, vědců Joana Daemona a Vincenta Rijmena. Vláda USA začala AES používat.

Šifra využívá symetrického klíče. Metoda šifruje data postupně v blocích s pevnou délkou 128 bitů. Šifra se vyznačuje vysokou rychlostí šifrování. Rijndael podporuje volitelně i větší datové bloky a klíče (ve 32bitových přírůstcích), než NIST požaduje, nicméně pro AES byla stanovena délka datového bloku na 128 bitů. Délka klíče je pak 128, 192 nebo 256 bitů. Rijndael je velmi flexibilní a čistý návrh, který lze popsat jak bajtově, tak pomocí 32bitových slov. V současné době nebyla tato metoda ochrany dat zatím prolomena.

1.1.2 Asymetrické šifrování

Vzniklo v roce 1975, vyvinuli ho Whitfield Diff a Martin Hellman. Dá se říci, že asymetrické šifrování je řešením nedostatků symetrické kryptografie. Je založena na existenci dvou klíčů – veřejném klíči a klíči soukromém. Výhodou symetrického šifrování je jeho rychlost, avšak je problém s distribucí klíče.

RSA

RSA ([12])byl vypracován na Massachussetském Institutetu Technologie v roce 1977 za spolupráce vědců Ronalda L. Rivest, Adi Shamira a Lena Adlemana. Algoritmus je obecně založen na „neschopnosti“ člověka vymyslet rychlý algoritmus pro rozklad velkých čísel na jeho prvočinitele. Používá se s drobnými úpravami dodnes v mnoha odvětvích – u mobilních telefonů, bankomatů, elektronických podpisů atd. Používá Fermatovu větu a modulární aritmetiku.

Pro jednoduchost vysvětlení principu asymetrického šifrování RSA byla zavedena jména Alice a Bob, jakožto účastníky fiktivní šifrované komunikace. Tímto vysvětlení urychlíme a zjednodušíme.

Představme si tedy komunikaci mezi Alicí a Bobem, která je šifrovaná asymetricky (tedy veřejným a soukromým klíčem). Alice má svůj soukromý klíč (AS) a veřejný klíč (AV). Bob má také svůj soukromý klíč (BS) a veřejný (BV). Komunikace může probíhat například takto:

1. Alice chce zaslat utajený text, proto zprávu šifruje.
2. Šifrování probíhá za pomoci soukromého klíče Alice (AS).
3. Zašifrovaná zpráva pomocí klíče AS se zašifruje ještě veřejným klíčem Boba (BV).
4. Zašifrovaná zpráva klíči (AS) a (BV) je zaslána příjemci zprávy – Bobovi – který ji bude dešifrovat.
5. Pro dešifrování nejdříve použije svůj soukromý klíč (BS) aby tím odšifroval svůj veřejný klíč (BV).
6. Dále pro dešifrování zprávy (nyní je zašifrovaná již pouze klíčem AS) použije klíče veřejný klíč Alice (AV).
7. Zpráva je nyní dešifrovaná a Bob si může přečíst text zprávy

1.2 Hashovací funkce

Hashovací funkce ([7]) je formálně funkce h, která převádí vstupní posloupnost bitů (či bytů) na posloupnost pevné délky n bitů. Je to předpis pro výpočet kontrolního součtu (hashe) ze zprávy či většího množství dat. Může sloužit ke kontrole integrity dat, k rychlému porovnání dvojice zpráv, indexování, vyhledávání apod. Je důležitou součástí kryptografických systémů pro digitální podpisy. Nejznámější hashovací funkce jsou MD5 (Message-Digest algorithm 5) a SHA (Secure Hash Algorithm).

Z definice hashovací funkce je patrné, že musíme ošetřit, aby hash byl unikátní a nemohl být pro různé zpracovávané řetězce stejný. Toto nazýváme „kolize“. Jinými slovy musíme zajistit, aby pro různé řetězce byl také různý hash. Teoreticky se však kolizím nelze vyhnout.

- **MD5** - ([13])byl vytvořen v roce 1991 (Ronaldem Rivestem), aby nahradil dřívější hašovací funkci MD4. V roce 1996 byla objevena vada v návrhu MD5, která mohla způsobit kolize. Ikdyž nebyla zásadní, kryptologové začali raději

doporučovat jiné algoritmy, jako je například SHA. Příklad MD5 hashe je v tabulce 1.1.

- **SHA** - V současné době je tento standard ([14]) mnohem perspektivnější. Byl navržen pro NSA (Národní Bezpečnostní Agentura – National Security Agency). Dnes se používá 5 algoritmů, jsou to SHA-1, SHA-224, SHA-256, SHA-384, a SHA-512, přičemž čtyři poslední jsou nazývány SHA-2. Příklad hashe SHA je v tabulce 1.2.

řetězec	**MD5 hash**
prosty text	442adf4e284b0d680e8aa90a52403aca
prosty	5678cbe02222a6743d99f884e9421169
text	1cb251ec0d568de6a929b520c4aed8d1

Tab. 1.1: Příklady MD5 hashe.

řetězec	**SHA-1 hash**
prosty text	44c8f62666dc09425e3b07ab78c771ee245c95a1
prosty	652a2e255bbaaf4e70b7491b81da80394ca090ee
text	372ea08cab33e71c02c651dbc83a474d32c676ea

Tab. 1.2: Příklady SHA-1 hashe.

1.3 Kryptoanalýza

Kryptoanalýza [5], [7] (z řeckého kryptós – „skrytý“ a analýein – „uvolnit“ nebo také „rozvázat“ je opakem kryptografie a zabývá se metodami získávání obsahu šifrovaných informací bez přístupu k tajným informacím, které jsou za normálních okolností potřeba. Jde především o získávání tajného klíče. „Krytoanalýza“ je také když mluvíme o pokusu obejít bezpečnost nějaký kryptografický systém nebo protokol obecně, ikdyž nemusí jít přímo o šifrování.

Z historického hlediska se kryptoanalýza velice změnila. První známý výklad kryptoanalýzy je znám z 9. století. Napsal jej arab Abu Yusuf Yaqub ibn Ishaq al-Sabbah Al-Kindi. Tato definice obsahuje popis metody zvané frekvenční analýza.

Zatímco dříve se používalo při šifrování například záměna písmene za jiné, dnešní kryptografické systémy jsou mnohem složitější a využívají různých matematických

operací. To je především dáno technologickým pokrokem a vývojem v oblastí počítačů. Při kryptoanalýze se tudíž také používají výpočetní možnosti počítačů, nebo speciálních strojů pro dešifrování. Dnes už existují také firmy, které se zabývají čistě jen kryptoanalýzou.

Zde uvedu některé známé kryptografické metody:

- **Frekvenční analýza** - jedná se o velmi starý princip. Používá se u substitučních šifer, v níž je šifrová abeceda tvořena písmeny. Princip je takový, že se vychází z písmena, které se v textu nejčastěji vyskytuje. Nalezne se k němu ekvivalent v šifrovém textu a podle toho se určí další písmena.

- **Lineární** - jedná se o základní formu kryptoanalýzy, kdy hledáme aproximaci k akci šifrování. Používá se u blokových a proudových šifer.

- **Differenciální** - na blokové i proudové šifry. Jedná se o studii jak se rozdíly na vstupu (prostý text) projeví na výstupu (šifrový text).

- **Integrální** - částečně aplikovatelná na blokové šifry. Založena na aplikaci SPN (Substitučně-Permutační Síť – Substitution-Permutation Network), což jse sled matematických operací používaných v blokových šifrách.

- **Brute Force attack** - tento typ útoku na šifru k jejímu dešifrování používá co největšího množství možností. Například použije postupně všechny typy klíčů k dešifrování.

- **Mod n** - používá se na kryptoanalýzu blokových nebo proudových šifer.

- **Related-keys** - jedná se o jakoukoli formu kryptoanalýzy, kde útočník může sledovat operace šifrování různými klíči jejichž hodnoty jsou původně neznámé, ale jsou známé některé matematické vazby spojené s klíči.

- **Slide** - používá se pro šifry, jejichž algoritmus je založen na jednoduchosti, ale pro větší odolnost šifry je algoritmus proveden vícekrát.

- **XSL** - používá se na blokové šifry. Tento typ kryptoanalýzy vzbudil rozruch, neboť má potenciál prolomit AES.

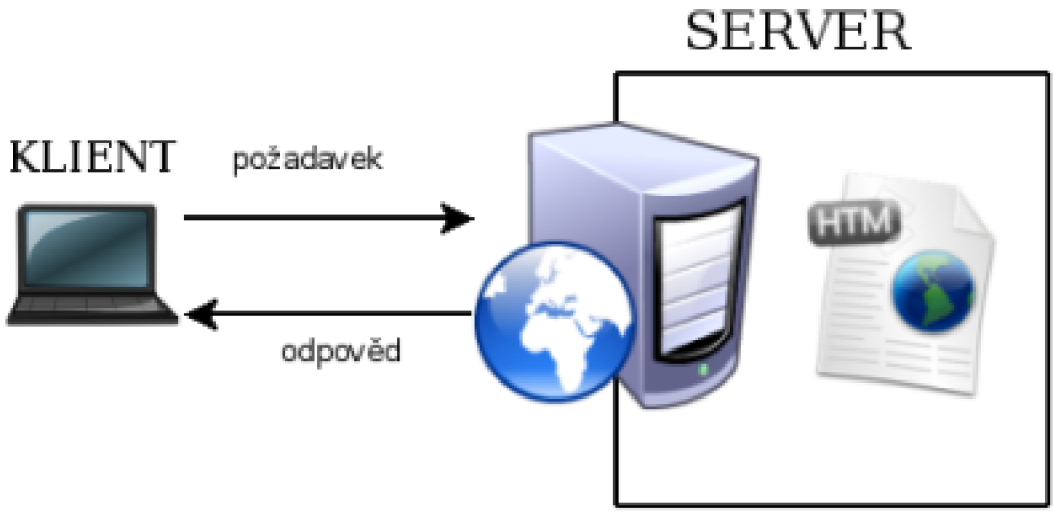
1.4 Webové aplikace

1.4.1 Jazyk HTML

HTML (HyperTextový značkovací jazyk – HyperText Markup Language) ([15]) je jazyk určený pro tvorbu webových stránek, především pro zobrazení obsahu stránek. HTML soubor je čistý textový dokument, který lze vytvářet jakýmkoliv textovým editorem na prakticky jakékoliv platformě a má koncovku .htm nebo .html.

Způsobu komunikace, pokud si chceme zobrazit nějakou webovou stránku, je klient-server. Na straně klienta je internetový prohlížeč a na straně serveru je aplikace (nejčastěji Apache), která „poslouchá“ většinou na portu 80 požadavky klienta. Příklad takovéto komunikace můžeme vidět na obrázku 1.3.

Nevýhodou MTML je, že je statický. To znamená, že pokud si klient zobrazí stránku napsanou v tomto jazyce, zobrazí se mu přesně to, co jsem do zdrojového kódu nadefinoval bez možnosti interakce ze strany klienta. Tuto nevýhodu řeší skriptovací jazyky, mezi něž patří například i PHP, kterým se budu zabývat v kapitole 1.4.2.



Obr. 1.3: *Komunikace klient-server bez PHP na serveru*

Na obrázku vidíme, jak zjednodušeně funguje komunikace klient-server bez použití technologie PHP na straně serveru.

1. klient vyšle požadavek (zadáme do prohlížeče adresu stránky, kterou chceme

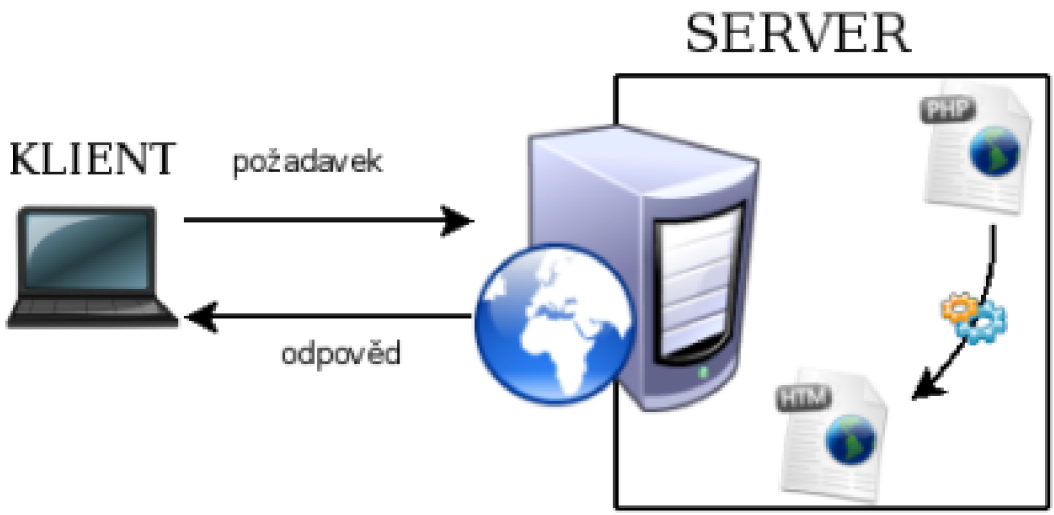
navštívit)

2. aplikace na straně serveru požadavek přijme

3. požadavek je dále zpracován tak, že vrátí klientovi požadovanou stránku (např index.html)

1.4.2 PHP

PHP (Hypertextový Preprocesor – Hypertext Preprocessor) ([2]) je univerzální skriptovací jazyk s volně dostupným zdrojovým kódem (Open Source), který je obzvláště vhodný pro vývoj webových aplikací a lze jej zapouzdřit do HTML. Jedná se o technologii, která je vykonávána na straně serveru (narozdíl od např. JavaScriptu, který je vykonáván na straně klienta). Rozdíl mezi použitím čistě jen HTML a mezi použitím PHP ke generování dynamických stránek je vidět na obrázcích 1.3 a 1.4. Znázorňují, že při zpracovávání na straně serveru se v případě PHP HTML stránka generuje. Jaká bude tato stránka je záležitostí skriptu napsaného v jazyce PHP. PHP má velké množství funkcí, jako je například možnost práce s databází. Jednou z možností je i práce s knihovnou OpenSSL, kterou popíši v kapitole 2.



Obr. 1.4: *Komunikace klient-server s PHP na serveru*

1. klient vyšle požadavek (zadáme do prohlížeče adresu stránky, kterou chceme navštívit)

2. aplikace na straně serveru požadavek přijme

3. jelikož se jedná o požadavek na dokument s příponou php (např. index.php), provede se skript v ní napsaný a vygeneruje se stránka v HTML kódu

4. server vrátí vygenerovanou stránku klientovi (index.php)

PHP je zaměřeno hlavně na skripty na straně serveru, takže dokáže například sběr dat z formulářů, generování dynamického obsahu, nebo přijímat a odesílat cookies. Existují tři hlavní oblasti, v nichž jsou použity PHP skripty:

- Skriptování na straně serveru. Toto je tradiční a hlavní oblast nasazení PHP. Pro správnou funkci potřebujeme tři věci. Parser PHP (CGI nebo modul serveru), webový server a webový prohlížeč. Potřebujeme spustit webserver s připojenou instalací PHP. Výstup z PHP programu můžete pak zobrazovat webovým prohlížečem, v němž otevřeme stránku v PHP umístěnou na daném serveru. Pokud pouze experimentujete nebo testujeme s programováním v PHP, může toto všechno běžet na jednom počítači.

- Skriptování v příkazovém řádku. Můžete přinutit PHP skript, aby běžel bez serveru a browseru. Pro toto použití potřebujete pouze PHP parser. Tento typ použití je ideální pro skripty spouštěné pravidelně v cronu (na *nixu nebo Linuxu) nebo v Naplánovaných úlohách (ve Windows). Tyto skripty mohou být rovněž použity v úlohách pro zpracování textu.

- Psaní desktopových aplikací. PHP pravděpodobně není ten nejlepší jazyk pro tvorbu desktopových aplikací s grafickým rozhraním, ale pokud známe PHP velmi dobře a chceme využít jeho pokročilých vlastností v aplikacích na straně klienta, můžeme takový program napsat pomocí PHP-GTK. Tímto způsobem lze rovněž napsat aplikaci pro více platforem. PHP-GTK je rozšíření jazyka PHP, které však není k dispozici v hlavní distribuci.

Zdrojový kód PHP může vypadat například takto:

```
<?php
echo <html><head>;
echo "<title>Název stránky</title> \n";
echo "</head> \n";
echo "<body> \n";
$retezec = "Toto www je stránka napsaná v PHP. \n";
echo "$retezec \n";
```

```
echo "</body> \n";
echo "</html> \n";
?>
```

Tento zdrojový kód se přeloží na straně serveru na HTML. V prohlížeči se zobrazí takto:

Toto je www stránka napsaná v PHP.

Díky těmto možnostem jazyka můžeme vytvářet dynamicky sránky na základě požadavku klienta. Například pokud klient zašle v požadavku na server dotaz na vyhledání nějakého řetězce v databázi, PHP skript nám může vygenerovat příslušný výpis z databáze.

Ukládání dat do databáze a práce s nimi je dnes v moderním webovém serveru nutností. S tím však souvisí i nutnost data v databázi chránit. V PHP je jednou z možností použití knihovny OpenSSL 2, díky které můžeme využívat funkce určené k šifrování.

1.4.3 Zabezpečení skriptovacích jazyků

Kromě jazyka PHP existuje mnoho jiných skriptovacích jazyků jako je například ASP (Active Server Pages) nebo JSP (Java Server Pages) skripty. V něčem se od sebe liší, ale mají jedno společné: musíme zabezpečit [3] jejich „nedostatky“ abychom předešli zneužití těchto technologií k případnému útoku, který může vést například k získání dat z databáze.

Častými útoky jsou typu XSS neboli Cross Site Scripting spočívají ve vložení nebezpečného kódu (napsaném v klientském skriptovacím jazyce jako je např. JavaScript) do obsahu webové stránky. Zabezpečení proti těmto útokům tkví v zamezení uživatelům vkládat skriptovací konstrukce do webových stránek.

- **Okamžitý útok** - zvaný non-persistent je založen na okamžitém zobrazení nebezpečného skriptu. Data se zobrazí, ale nikam se neukládají. Tento útok je nejčastější u vyhledávacích služeb. Útočník „donutí“ uživatele aby kliknul na odkaz, pod kterým se skrýva skript, který může přenést údaje uložené v prohlížeči uživatele.

- **Perzistentní útok** - je jeden z nejnebezpečnějších. Využívá možnosti uživatele ukládat svá data na web (např. diskusní fóra, komentáře apod.) Využívá

přitom nezabezpečené aplikace, která neošetří možnost vkládání běžný HTML kód, který může obsahovat skript.

- **Lokální útok** - někdy zvaný jako DOM-based nebo local je velmi podobný okamžitému útoku s tím, že ke zpracování nebezpečného skriptu se zneužije existující klientský skript. Největší nebezpeší je v lokálních webových aplikacích, protože nejrozšířenější z prohlížečů, Internet Explorer, považuje javascripty spouštěné v lokální zóně za bezpečné. Lokální skripty mohou přistupovat k souborům na lokálních discích.

- **Zneužití oprávněného požadavku** - je útok podobající se typově XSS. Cílem útočníka je donutit registrovaného uživatele, který je přihlášen do určitého systému vykonat požadavek, který je v danou chvíli z pohledu uživatele oprávněný. Útočník pro útok musí získat důvěru (to může být realizováno útokem XSS). Pomocí klientského skriptu se odešle skrytý formulář. Nepřenášejí se žádné důvěrné informace, neboť uživatel je v danou chvíli přihlášen. Ochranou proti tomuto útoku může být například tzv. *tokenizace*, který je založen na generování jednorázových přístupových hesel. Další možností je např. tzv. dodatečná autentizace, například autorizační sms.

Samotný skriptovací jazyk lze zabezpečit nastavením správných direktiv v konfiguračních souborech. Některé direktivy by měly být z bezpečnostního hlediska vypnuty. Jedná se o direktivy *safe_mode* – ta nastavuje bezpečný režim, je však nespolehlivá a do PHP verze 6 se s ní nepočítá, *display_errors* – zobrazování chyb je nebezpečné, protože může útočníkovi značně pomoci, *magic_quotes_sybase* – mění zásadním způsobem ošetřování speciálních znaků v SQL dotazu, *register_globals* – otevírá možnost útoku pomocí vložených proměnných.

Dále existují direktivy, které jsou z hlediska bezpečnosti vhodné: *open_basedir* – zamezuje ze skriptu přístup mimo zadaný adresář, *disable_functions* – můžeme specifikovat zakázání určitých funkcí (exec), *session.use_only_cookies* – nebudou se brát v potaz session ID přicházející z GET nebo POST dat a *session.save_path a upload_tmp_dir* – nastavují cestu k ukládání sessions a s dočasných dat. Vhodné nastavit na bezpečné místo.

1.4.4 Zabezpečení dat v databázi

Dalším aspektem v zabezpečení webových aplikací je bezpečnost dat v databázi [4]. Pomocí skriptovacích jazyků můžeme klást dotazy na databázi. Těmito dotazy mů-

žeme například data vkládat, mazat, nebo číst. V PHP může takový dotaz vypadat například takto:

```
mysql_query("SELECT * FROM user WHERE id='$user' ");
```

Snahou útočníka je nalezení místa, kde klademe dotaz a tento dotaz zmanipulovat tak, aby dotaz změnil ve svůj prospěch. Příkladem budiž dotaz

```
mysql_query("SELECT * FROM user WHERE id='$user' OR 'x'='x' ");
```

kde můžeme vidět část kódu `OR 'x'='x'`. Tato část je z hlediska bezpečnosti dotazu velice nebezpečná, neboť podmínka bude platit vždy a může vypsat všechny položky z tabulky *user*.

Zabezpečení tohoto problému spočívá v převedení všech kontrolních značek v jejich datovou reprezentaci a doplněním o značky ' , " a NULL o \ . V PHP tuto úlohu provádějí funkce mysql_real_escape_string() nebo addslashes().

Vedle řešení zneužití dotazu na datazázi se musíme také zaměřit na vlastní data uložené v databázi. Pokud do databáze ukládáme citlivá data jako jsou například hesla, není vhodné ukládat je přímo jako řetězec hesla. Je vhodné používat například hashovací funkce. Toto nazýváme *jednocestným zabezpečením*, což je v principu zakódování dat, kdy nepotřebujeme získat data zpět do původní formy. Při potřebě pak vytvoříme znovu hash a porovnáme s hashem v databázi. Druhým typem zabezpečení jsou tzv. *vratné metody*, kde potřebujeme data ze zakódované formy dostat zpět na původní. K tomuto se využívá principů symetrického nebo asymetrického šifrování.

1.4.5 Zabezpečení komunikace pomocí HTTPS

K zabezpečení proti útokům vedoucím k odcizení dat se také používá šifrovaná komunikace. Jedná se o protokol HTTPS (Hypertectový Přenosový Protokol přes SSL - Hypertext Transfer Protocol over Secure Socket Layer) ([16]), který je nadstavbou protokolu HTTP (Hypertectový Přenosový Protokol - Hypertext Transfer Protocol) ([17]). Ten poskytuje zvýšenou bezpečnost před odposloucháváním či podvržením dat. HTTPS není přímo zvláštní protokol, data jsou přenášena pomocí HTTP, ale namísto aby byla přenášena v běžném textu, jsou šifrována pomocí SSL nebo TLS viz kapitola 2.1.

Stejně jako protokol HTTP, tak i HTTPS je implementován webovým serverem Apache. První zmíněný je dostupný na portu 80 a druhý na portu 443. Pokud se na tomto serveru rozhodneme používat HTTPS, máme dvě možnosti. Dá se použít mod_ssl a Apache-SSL. Celá nadstavba funguje na principu asymetrického šifrování, která je implementována pomocí knihovny OpenSSL.

2 OPENSSL

V dnešním světě internetu je mnoho aplikací, které potřebují zabezpečení a kryptografie je jedna z možností. Primární zaměření kryprografie je diskrétnost dat, jejich integrita a autentizace. Toto může být použito proti velkému množství útoků ze sítě jako jsou například „odposlouchávání“ (eavesdropping), „klamání cíle“ (IP spoofing), „únos spojení“ (connection hijacking) či „ovlivňování“ (tampering).

OpenSSL [1, 8] je kryptografická knihovna, která poskytuje implementaci průmyslově nejvíce doporučovaných algoritmů. Obsahuje algoritmy jako 3DES, AES (viz. kapitola 1.1.1), RSA (viz. kapitola 1.1.2) a také hashovací a autentizační algoritmy. Jádro knihovny je napsané v programovacím jazyce C. Je distribuována jako open source (otevřený zdrojový kód) a je dostupná pro většinu operačních systémů Unixovského tipu (Solaris, Mac OS X, BSD), OpenVMS a Microsoft Windows.

Následkem útoků na systémy, které obsahují citlivá data vznikla potřeba tyto data chránit zabezpečováním těchto systémů. Útočník se snaží využívat slabin obecných principů síťové komunikace, nebo slabin konkrétních aplikací či protokolů. Časté je přesměrování komunikace na sebe a odposlouchání. Další z možností je přístup bez oprávnění, či omezení fungování služeb.

Některé typy útoků

- **IP spoofing** - znamená falšování zdrojové IP adresy, čímž se útočník snaží předstírat, že je někdo jiný, nebo se snaží zamaskovat svoji pravou IP adresu.
- **Algoritmy hrubé síly** - crackovací program zkouší jako možné heslo všechny možné existující kombinace. Časově náročné, ale s jistotou, že na heslo dříve či později přijde. Příkladem může být slovníkový útok.
- **Rainbow tables** - z řetězců je vytvořena tabulka hashů (jde o urychlení prolomení hesla) - nemusí se znovu vypočítávat hash.
- **Analyzátor paketů (sniffer)** - pouze naslouchá a analyzuje síťový provoz.
- **Buffer overflows** - využívá chybně napsaných programů.
- **Man-in-the-middle** - tento typ útoku lze popsat tak, že útočník může číst a měnit data která si posílají dvě vzájemně komunikující strany podle jeho libosti a to aniž by věděly, že jsou terčem útoku.

Rozdíl mezi OpenSSL a SSL 2.1 je v tom, že SSL je protokol, kterým lze zabezpečit síťovou komunikaci protokolů aplikační vrstvy. Pomocí OpenSSL můžeme generovat klíče a certifikáty pro SSL, nebo například zabezpečit data či citlivé údaje prostřednictvím algoritmů, které OpenSSL podporuje.

2.1 SSL protokol

SSL (Vrstva Bezpečných Socketů – Secure Socket Layer) protokol ([7, 18]) byl vyvinut a publikován firmou Netscape. Verze 3.0 vytvořila základ pro TLS (Bezpečná Transportní Vrstva – Transport Layer Security). Vrstva SSL (resp. TLS) je v modelu protokolů TCP/IP vložena mezi aplikační a transportní vrstvu. Je schopna u každého paketu zajistit jeho privátnost (šifrování), integraci a autorizaci. Autorizace se provádí na základě jednocestné hashovací funkce (hashe) počítaného z dat a sdíleného tajemství. Při navazování spojení vrstva SSL provádí autentizaci serveru (na základě certifikátu). Autentizace klienta je volitelná. Při generování klíčů pro komunikaci můžeme využít knihovny OpenSSL. Pomocí SSL lze zabezpečit protokoly aplikační vrstvy, jako například HTTP, FTP, POP3, SMTP apod.

Vytvoření SSL spojení:

1. Klient pošle serveru požadavek na SSL spojení (Client_Hello), spolu s různými doplňujícími informacemi (verze SSL, nastavení šifrování atd.).

2. Server pošle klientovi odpověď na jeho požadavek (Server_Hello), která obsahuje stejný typ informací, ale především certifikát serveru.

3. Na základě přijatého certifikátu si klient ověří autentičnost serveru. Certifikát také obsahuje veřejný klíč serveru.

4. Podle obdržených informací vygeneruje klient základ šifrovacího klíče, kterým se bude šifrovat následná komunikace. Ten zašifruje veřejným klíčem serveru a pošle mu ho.

5. Server použije svůj soukromý klíč k rozšifrování základu šifrovacího klíče. Z tohoto základu vygenerují jak server, tak klient hlavní šifrovací klíč.

6. Klient a server si navzájem potvrdí, že od této chvíle bude jejich komunikace šifrovaná tímto klíčem. Fáze handshake tímto končí.

7. Je ustaveno zabezpečené spojení šifrované vygenerovaným šifrovacím klíčem.

8. Probíhá komunikace přes šifrované spojení.

2.2 PKI

PKI (Infrastruktura veřejného klíče – Public Key Infrastructure) je soustavou technických a organizačních opatření spojených s vydáváním, správou, používáním a odvoláváním platnosti kryptografických klíčů a certifikátů. Jednu z možných norem PKI definuje sada internetových standardů RFC (Žádost o komentáře – Request For Comments) popisujících základní využití asymetrické kryptografie na Internetu. Navazujícími normami jsou pak normy pro bezpečnou elektronickou poštu (S/MIME) a norma pro bezpečnou komunikaci (SSL).

2.2.1 Certifikát

Certifikát je struktura obsahující identifikační údaje klienta, veřejný klíč klienta, identifikaci vydavatele, číslo certifikátu, platnost certifikátu a další údaje týkající se zejména vymezení způsobu použití certifikátu. Tato struktura je digitálně podepsána certifikační autoritou (vydavatelem certifikátu). Certifikáty se používají proti podvržení veřejného klíče při komunikaci šifrované pomocí asymetrického algoritmu. Veřejný klíč se neodesílá samotný, ale jako součást certifikátu, který je podepsán soukromým klíčem CA. Příjemce certifikát ověří a v případě, že je vystaven pro něj důvěryhodnou certifikační autoritou a elektronický podpis na certifikátu je v pořádku, může z tohoto certifikátu použít veřejný klíč k šifrování zprávy. V Internetu je certifikát popsán normou [9]. Tato norma je odvozena od doporučení ITU X.509.

2.3 OpenSSL v PHP

Rozšíření OpenSSL v PHP je k dispozici od verze PHP-4.0.4. Toto rozšíření využívá funkce OpenSSL pro tvorbu a ověřování podpisů a pečetění (kódování) a otevírání (dekódování) dat. OpenSSL nabízí mnoho vlastností, které tato extenze v současnosti nepodporuje. Některé z nich mohou být v budoucnu přidány.

3 REALIZOVANÁ APLIKACE

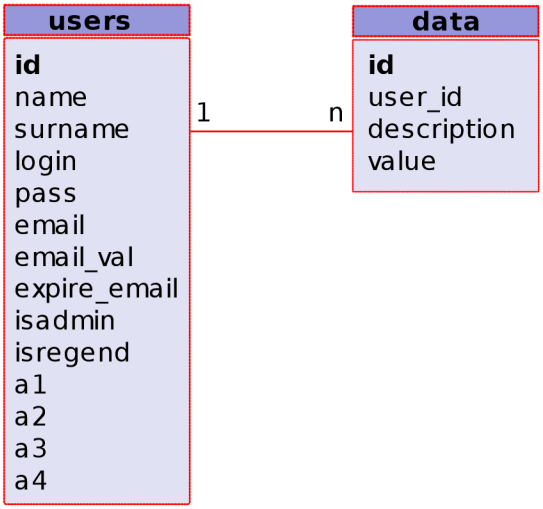
V této kapitole popíši realizovanou webovou aplikaci, která by měla být zabezpečena proti celé škále útoků. K tomuto účelu využiji hashovací funkce a především rozšíření OpenSSL pro skriptovací jazyk PHP. Server, na kterém jsou tyto služby dostupné, je v systému Linux. Webový server Apache je verze 2.2.8-1, PHP verze 5.2.4 a verze knihovny OpenSSL je 0.9.8.

3.1 Popis aplikace

Aplikace je koncipována jako víceuživatelský systém, kde se uživatelé mohou zaregistrovat a po následném potvrzení registrace se mohou přihlásit a ukládat svá data. Ty budou uloženy do relační databáze MySQL a uživatel si je může prohlížet, dále editovat a mazat. Dále bude vytvořen administrátorský účet, kde si bude moci administrátor prohlédnout veškerá data v databázi v původním tvaru.

3.1.1 Návrh databáze

Vzhledem k tomu, že k aplikaci bude přistupovat mnoho uživatelů a ti pak budou moci přidávat svá privátní data, rozhodl jsem se navrhnout databázi následujícím způsobem:



Obr. 3.1: *Databáze s tabulkami users a data.*

V databázi jsou vytvořeny dvě tabulky, a to **users**, ve které uchovávám příslušná potřebná data týkající se uživatelů, a tabulka **data**, kde jsou uloženy zašifrované

citlivé údaje uživatelů. Každý řádek tabulky má svůj primární klíč, podle kterého je jednoznačně definován. V tabulce dat je řádek s klíčem uživatele, abych mohl rozpoznat ke kterému uživateli patří příslušný řádek.

Relace mezi tabulkami je 1/n, neboť uživatel může mít více záznamů. Na obrázku 3.1 je vidět jak jsou tabulky navržené. Každá tabulka obsahuje několik řádek, do kterých se ukládají příslušné údaje. Význam jednotlivých řádků tabulky **users** je vidět v tabulce 3.1. V tabulce 3.2 jsou uvedeny její řádky a jejich účel.

řádek tabulky	**význam**
id	jednoznačný identifikátor řádku
name	jméno uživatele
surname	příjmení uživatele
login	uživatelské jméno pro přihlášení uživatele
pass	heslo uživatele uložené jako salted hash (viz. kapitola 3.3.1)
email	email uživatele
email_val	příznak ověření emailu uživatele
expire_email	zde je uložen čas dokdy bude aktivován účet pomocí emailu
isadmin	příznak určující, zda je uživatel zároveň administrátorem
isregend	příznak k určení, zda registrace proběhla v pořádku až do konce
a1,a2,a3	slouží k uložení řetězců pro salted hash
a4	obsahuje číselnou hodnotu, která byla použita u metody salted hash (kapitola 3.3.1)

Tab. 3.1: Význam řádků tabulky users.

řádek tabulky	**význam**
id	jednoznačný identifikátor řádku
user_id	id uživatele, který řádek přidal
description	popis záznamu (uložený zašifrovaně)
value	vlastní záznam (opět zašifrovaně)

Tab. 3.2: Význam řádků tabulky data.

3.2 Zabezpečení

- zabezpečení Apache pomocí vhodných direktiv v konfiguračním souboru
- zabezpečení pomocí nastavení skriptovacího jazyka (PHP)
- zabezpečení databáze
- zabezpečení dat v databázi
- ošetření proti Cross Site Scripting (XSS)
- zabezpečení komunikace (https)
- dodatečná zabezpečení

3.2.1 Zabezpečení Apache

Jako webový server jsem v diplomové práci použil Apache 2.2, neboť se jedná o nejrozšířenější webový server na světě a je obsažen v distribuci Ubuntu, jež používám jako operační systém.

Hlavní konfigurace Apache se provádí textově v konfiguračním souboru, který se nachází v */etc/apache2/apache2.conf*. Začneme tím, že nastavíme, aby Apache pracoval pod uživatelem apache ve skupině apache. Bylo by nevhodné, aby webový server běžel pod uživatelem root, neboť root má takřka neomezené možnosti měnit nastavení systému a případný útočník by tohoto mohl zneužít. Dále skryjeme verzi apache při generovaných chybových zprávách.

Poté zrušíme výpis verze v HTTP hlavičkách. Toto je důležité především z důvodu bezpečnostních chyb v různých verzích apache. Kdyby útočník věděl přesnou verzi serveru, mohl by zjistit bezpečností chybu a pokusit se jí napadnout. Tyto direktivy můžeme vidět zde:

```
User apache
Group apache
ServerSignature Off
ServerTokens Prod
```

Důležité je především nastavení práv pro adresář se soubory webového serveru, který se nachází v adresáři */var/www/diplomka*.

Nechceme především, aby mohl kdokoliv listovat v adresářích. Toto nastavíme direktivou *-Indexes*, také vypneme symblické odkazy (*-FollowSymLinks*) a dále spouštění CGI skriptů (*-ExecCGI*). Dalším krokem je vypnutí přepsání těchto direktiv souborem .htaccess direktivou *AllowOverride None*:

```
<directory /var/www/diplomka>
Options -Indexes -FollowSymLinks -MultiViews -ExecCGI
AllowOverride None
Order allow,deny
allow from all
DirectoryIndex index.php, index.html, index.htm
</directory>
```

Souborem .htaccess se nastavují pravidla přístupu k souborům a adresářům na serveru.

3.2.2 Zabezpečení skriptovacího jazyka PHP

Skriptovací jazyk PHP je možno nastavit v hlavním konfiguračním souboru, který je umístěn v */etc/php5/apache2/php.ini*. Zde můžeme nastavit mnoho direktiv a vzhledem ke zvýšení bezpečnosti je vhodné mít některé direktivy zapnuté a některé naopak vypnuté. Zde popíši nejdůležitější z nich.

Direktivy, které jsou vypnuté

- **safe_mode** - sám název (bezpečný mód) napovídá, že tato direktiva by měla být spíše zapnutá, ve skutečnosti vypíná některé direktivy, které by z hlediska bezpečnosti měly být zapnuté; ke zvýšení bezpečnosti je tedy vhodnější přispět jinak než touto direktivou
- **display_errors** - tímto vypneme výpisy případných chyb ve zdrojovém kódu do vygenerované stránky a tím zamezíme útočníkovi tyto chyby analyzovat
- **magic_quotes_sybase** - tato direktiva mění ošetřování znaků v SQL dotazu
- **register_globals** - při zapnutí této direktivy můžeme v PHP přistupovat k proměnným odeslaných z formulářů například přes $heslo apod; toto není bezpečné a raději direktivu vypneme; k proměnným z formuláře poté přistupujeme např. pomocí superglobálních proměnných POST, GET apod.; v PHP 6

již nepůjdou zapnout, a tudíž nebude zpětná kompatibilita pro skripty napsané s předpokladem jejího zapnutí

- **expose_php** - vypnutím této direktivy zakážeme zasílání informací o instalované verzi PHP v HTTP hlavičce

Především direktivu register_globals je třeba mít vypnutou. Ta může nabývat dvou hodnot - buď „zapnuto“ nebo „vypnuto“. Pokud je zapnutá, znamená to, že k proměnným z odeslaných formulářů, k proměnným v URL a jiným podobným proměnným je možné přistupovat přímo přes název proměnné, např. přes `$heslo`. Pokud je však vypnutá, tak je možné přistupovat jen přes superglobální pole přes `$_POST["heslo"]` nebo například přes `$_GET["heslo"]`. Z toho vyplývá, že útočník by mohl v URL stránky podvrhnout jakoukoli proměnnou například z URL `http://www.server.cz/test.php?vstup=promenna`.
Skriptu test.php by tedy byla podvržena proměnná `$vstup` a dále by ji zpracovával.

Direktivy, které jsou zapnuté

- **open_basedir** - zde se nastavují adresáře, kam bude mít skript přístup
- **log_errors** - pokud nechceme, aby se zobrazovaly chyby přímo do vygenerované stránky, je vhodné nechat tyto chyby logovat do souboru, přičemž si je můžeme později prohlédnout a vyhodnotit
- **disable_functions** - vypnutí nežádoucích funkcí (např. exec)

Funkce „exec“ dokáže spustit příkaz v jejím parametru. Je tedy velice důležité od ní oddělit uživatelský vstup. Pomocí této funkce by mohl útočník spustit skript, který by zobrazil na vygenerované HTML stránce například soubor s hesly uživatelů systému.

3.2.3 Zabezpečení databáze

Použita je databáze MySQL 5.051. K jejímu zabezpečení jsem nastavil v konfiguračním adresáři */etc/mysql/my.cnf* *bind* adresu na *localhost*, čímž zajistím přístup k mysql pouze ze serveru. Dále vytvořím v konzoli uživatele, který bude moci přistupovat k uloženým datům v databázi. V terminálu tedy napíši:

```
sudo mysql -u root -p
mysql> CREATE DATABASE diplomka;
mysql> GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP,
INDEX, ALTER, CREATE TEMPORARY TABLES, LOCK TABLES
ON diplomka.* TO 'dp_user'@'localhost'IDENTIFIED BY 'password';
mysql> \q
```

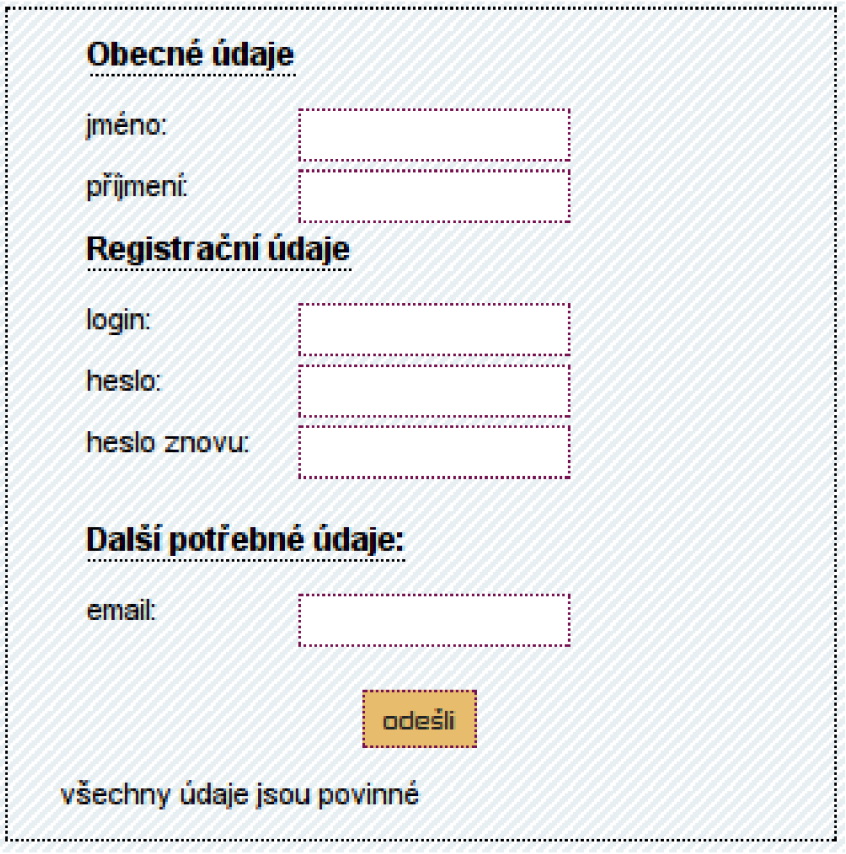
Prvním příkazem se přihlásím do konzole mysql, která je uvozena znaky mysql>. Zde mohu jako uživatel root spravovat databázi. Dalším příkazem vytvořím databázi *diplomka*, poté uživatele, který bude mít k této databázi přístup, s privilegii pro příkazy `SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, INDEX, ALTER, CREATE TEMPORARY TABLES, LOCK TABLES`, což pro aplikaci, kde potřebujeme, aby si mohl uživatel ukládat do této databáze soukromá data, postačuje. Zároveň tím eliminuji možnost pomocí tohoto uživatele (dp_user) přistupovat k jiným tabulkám. Příkazem `\q` opustím konzoli mysql.

3.3 Vlastní aplikace

Nyní mám nakonfigurován Apache, PHP i MySQL a mohu přejít k vlastní aplikaci. Jak již bylo řečeno, jedná se o víceuživatelský systém, kde se mohou uživatelé registrovat. Zde ukáži jak aplikace vypadá a jakým způsobem je zabezpečená proti SQL injection, XSS útokům a jak jsem zabezpečil data v databázi.

Aby bylo možno využívat služby webové aplikace, musíme se nejdříve zaregistrovat. Na obrázku 3.2 vidíme formulář, kde uživatel vyplní své údaje vyžadované při registraci.

Jak vidíme, zadává se i heslo, pomocí kterého se bude moci přihlásit ke svému nově vytvořenému účtu. Heslo se do databáze neukládá přímo, ale ve většině případů se ukládá vytvořením hashe (SHA1) a teprve ten se uloží do databáze. Při přihlašování se hash opět vytvoří a porovná s hashem z databáze. Zásadou bezpečného hesla je především to, aby se nejednalo o běžně používaná slova jako například *práce, škola, hokej, honza* apod. Tyto hesla jsou velmi snadno napadnutelná, neboť útočník, který příslušný hash získá, může využít tzv. slovníkový útok a tato slova zkoušet, dokud hash nebude souhlasit. Toto lze ošetřit metodou zvanou *salted hash*. Díky této metodě nebudou mít 2 uživatelé, kteří mají stejné heslo, stejný hash. Předpokladem je samozřejmě jiný *salt* pro oba uživatele.



Obr. 3.2: *Registrační formulář.*

3.3.1 Salted hash

Princip je takový, že se k heslu vygeneruje náhodný řetězec, který se k němu přidá a teprve z tohoto se vytvoří hash do databáze. Já jsem použil následující salted hash:

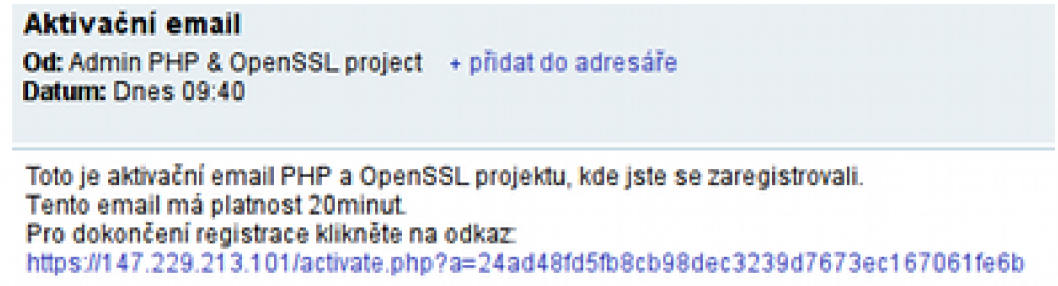
- vygeneruji 3 salty
- jeden vložím dovnitř hesla
- jeden přidám na začátek
- poslední přidám za heslo

Vegeneruji si tedy 3 řetězce pro vytvoření salted hashe: `MTM5OTQzNDA3NAd`, `MTEONTExMTQ3OQf`, `MTg3OTIxNzY3OA6` a uložím je do databáze. Pokud bych je neuložil, nemohl bych heslo ověřit. Také musím uložit vygenerované číslo, které reprezentuje na jaké pozici v heslu bude umístěn první salt. Pokud tedy máme heslo *škola*, řetězec, který použiji pro vytvoření hashe může vypadat například:

`MTE0NTExMTQ3OQfškoMTM5OTQzNDA3NAdlaMTg3OTIxNzY3OA6`.
Toto útočníkovi takřka znemožní výše uvedený typ útoku.

Aktivační email

Pro dokončení registrace je potřeba do 20 minut potvrdit ji pomocí aktivačího emailu. Ten je zaslán ihned po potvrzení formuláře na zadaný email. Aktivační email můžeme vidět na obrázku 3.3



Obr. 3.3: *Aktivační email.*

Po kliknutí na odkaz ve spodní části emailu registraci dokončíme. V URL tohoto odkazu je část `a=24ad48fd5fb8cb98dec3239d7673ec167061fe6b`. Jedná se o parametr, který je vygnerován a uložen ihned po odeslání formuláře registrace. Po kliknutí na odkaz je tato část zkontrolována. Pokud souhlasí a zároveň nevypršel časový limit, je registrace dokončena a uživatel se může přihlásit. Zároveň jsou vygenerovány openssl klíče pro šifrování a dešifrování. Rozšíření openssl v PHP umožňuje šifrovat data veřejným klíčem a dešifrovat privátním.

3.3.2 Systém uložení klíčů

Jak již bylo řečeno, na webovém serveru si budou moci uživatelé ukládat svá citlivá data. Tato data ukládám do databáze zašifrovaně tak, že každý uživatel má 2 klíče: veřejný a soukromý. Pomocí veřejného data zašifruji a následně uložím do databáze. Privátním pak tyto informace zpět dešifruji.

Po vygenerování klíčů je třeba také klíče uložit na bezpečné místo. Bylo by nevhodné ukládat je přímo do adresáře, kde jsou skripty. Proto jsem zvolil adresář jiný.

Vygeneruji klíče pomocí následujících funkcí:

- `openssl_pkey_new`
- `openssl_csr_new`
- `openssl_csr_sign`

Poté klíče uložím, a to následujícím způsobem:

1. z loginu uživatele vytvořím hash: *loginuzivatele_pub.pem* pro veřejný a analogicky *loginuzivatele_pri.pem* pro privátní klíč
2. takto vygenerovaný hash použiji pro název souboru s klíčem

Název souboru s veřejným klíčem může být například:
9fe3305061297dd79f7aebed9c150895dd966580

a pro privátní klíč:
afb0d4336387860d12a3b05b348bf7ce4fdc938c.

Abych zamezil útočníkovi dostat se k těmto klíčům, příslušný adresář je vlastněn uživatelem *apache* a ostatní uživatelé do této složky nemají přístup. Dále jsem také zamezil přihlašování tohoto uživatele do systému.

3.3.3 Ochrana proti SQL injection

Ochrana proti SQL injection je velice důležitá, neboť pomocí ní může útočník získat data z databáze. V PHP je možnost využití direktivy *magic_quotes_gpc* v konfiguračním souboru php.ini, která veškeré hodnoty uzavírá do apostrofů a jsou doplněny o tzv. escape sekvence. Existuje také například možnost využití funkce *addslashes*, ale tu musíme použít při každém vstupu dat od uživatele do aplikace.

Ve výsledné aplikaci jsem využil možnosti *magic_quotes_gpc*, neboť ošetřuje data globálně a nemusím se obávat, že někde zapomenu použít před dotazem do MySQL databáze funkci *addslashes*.

3.3.4 Ochrana proti XSS

Zabezpečení proti XSS popsaných v 1.4.3 je rovněž velmi důležité. Pokud by mohl útočník vložit do skriptu konstrukci, která například zavolá javascript, vytvoří odkaz apod. To je samozřejmě nežádoucí a je třeba to ošetřit. K tomuto jsem využil

funkci *htmlspecialchars*, která přemění speciální znaky využívané v HTML do takové podoby, aby při výstupu nebyla vložena jako HTML entity, ale jako text.

Mějme tedy skript:

```
<?php
$_POST["data_z_formulare"];
//např. <a href='test'>Test</a>
echo $_POST["data_z_formulare"];
?>
```

V tomto skriptu přebírám hodnotu z formuláře zaslané metodou POST, které mohou mít například hodnotu `<a href='test'>Test</a>`. Pokud bych toto uložil do databáze a následně bych řetězec chtěl zobrazit (zde příkazem echo) na vygenerované stránce, prohlížeč by je interpretoval jako HTML entitu a v tomto případě by vytvořil odkaz. V následujícím skriptu je toto nebezpečí ošetřeno:

```
<?php
$_POST["data_z_formulare"];
// např. <a href='test'>Test</a>
$new = htmlspecialchars($_POST["data_z_formulare"], ENT_QUOTES);
echo $new;
?>
```

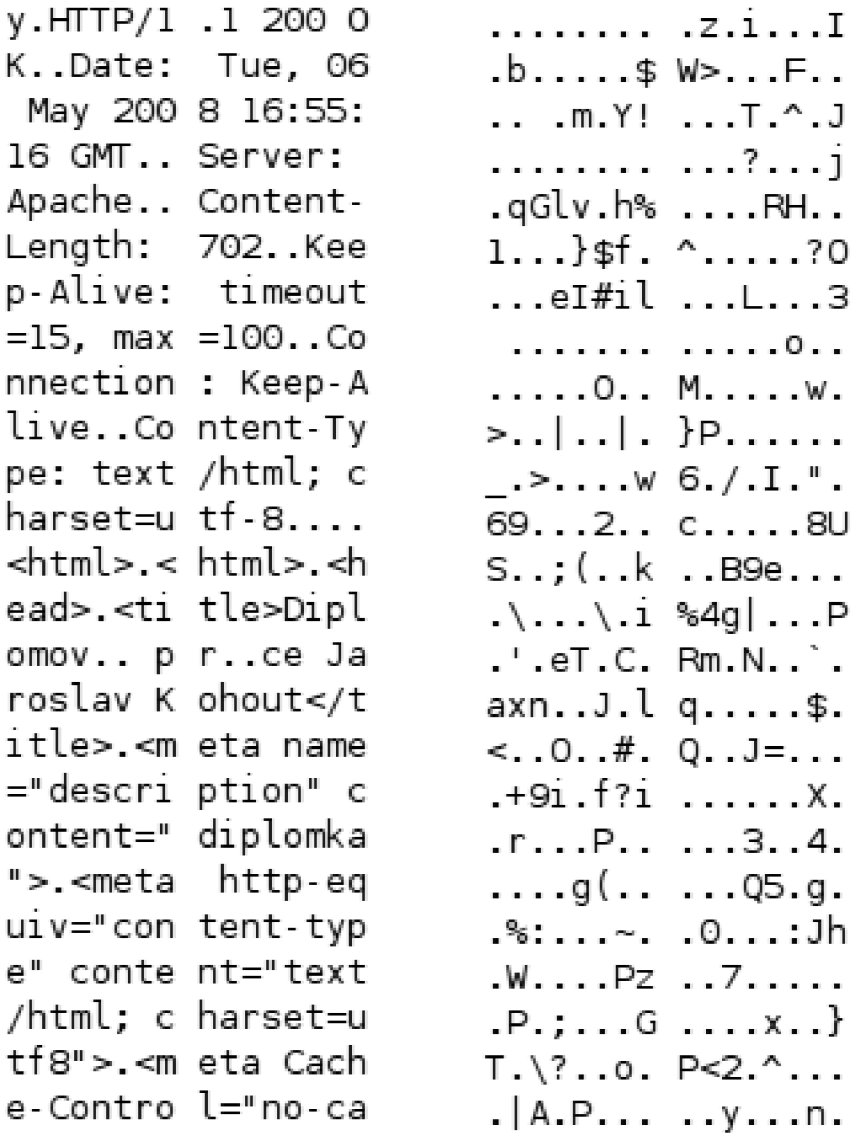
Zde řetězec z formuláře převedu na speciální znaky tak, aby je prohlížeč neinpretoval jako HTML entity, ale jako text. Tentokrát z řetězce
Test
vytvoří funkce htmlspecialchars řetězec
<a href='test'>Test</a&gt
a tím by prohlížeč zobrazil (ne však vykonal) Test.

3.3.5 Zabezpečení komunikace

Pokud chceme zabezpečit komunikaci mezi serverem a klientem, použijeme SSL (popsaný v 2.1). Tím veškerá komunikace probíhá zašifrovaně a útočníkovi při případném odchytu paketů budou tato data bez znalosti klíčů k ničemu. K demonstraci zabezpečení využiji program *Wireshark*, což je zachytávač a analyzátor paketů. Na obrázku 3.4 vidíme zachycené datové pakety. V levé části je vidět paket zachycený při komunikaci přes http protokol. Je z něho rozpoznatelná vygenerovaná HTML stránka. Naproti tomu v pravé části obrázku 3.4 vidíme paket zachycený při komunikaci pomocí https. Na první pohled je zřejmé, že z tohoto paketu útočník nerozezná

přenášený HTML kód.

Pomocí Wiresharku se mi také podařilo odchytit pakety *Server Hello*, *Client Key Exchange, Change Cipher, Encrypt Handshake*, *Change Cipher Spec, Encrypted Handshake Message*, které jsou typické pro protokol https. Jedná se o pakety, kde si server vymění s klientem informace potřebné k šifrování komunikace.



Obr. 3.4: *Zachycené pakety pomocí programu Wireshark.*

Zabezpečením protokolem https jsem chtěl docílit toho, abych útočníkovi znemožnil použít důležité informace, která případně odposlechne při komunikaci (eavesdropping). Jelikož jsou zašifrovaná, jsou pro něho nepoužitelná. V tomto případě

používá prohlížeč k šifrování komunikace certifikát serveru, který je mu serverem automaticky nabídnut.

Ke komunikaci jsem vygeneroval certifikát podepsaný sám sebou (*self-signed certificate*) pomocí příkazu `apache2-ssl-certificate`:

```
sudo apache2-ssl-certificate -days 365
```

Tímto vygeneruji soubor, který bude uložen v `/etc/apache2/ssl/apache.pem`. V parametru `days` příkazu je uvedena doba platnosti certifikátu. Obsah souboru může vypadat následovně:

```
-----BEGIN RSA PRIVATE KEY-----
MIICXQIBAAKBgQCyWrqYQSzSIBviavR/8LSd5/w5QeZ/n0QDQ1uZjjZBwziiT5/1
1ZryKf0aZCUpfAwxl9BtqHw58JVo7eOoGU+Zvq8fbLJpYsuISbgkataWlIw8JERr
.
.
.
.
.
T/0mFTbgaG+r5jsF0YtoKrwschYbbSxio4c659C14J++hcm2MdlBpK4J8DHqKvC+
tuJ/q9CKpvnRX4St2ckCQQDQr/jsD0BxrAHgZ4fhHdZvpCwBd2Ph50k8OjBY2rRj
XPLkB+WwES1CT84Ef3v3R3Jb9dzeFQxYKY0gqIT9PeuV
-----END RSA PRIVATE KEY-----
-----BEGIN CERTIFICATE-----
MIICkzCCAfwCCQDViyJAzErCsTANBgkqhkiG9w0BAQUFADCBjTELMAkGA1UEBhMC
Q1oxDjAMBgNVBAgTBUNlc2tvMQ0wCwYDVQQHEwRCam5vMREwDwYDVQQKEwhWVVQg
.
.
.
.
.
BQUAA4GBAIvRfqbjSsQ93YVf+1zxFSBWIb5OHc6Xbsq3hNcAU+6rxL9frxQ0mfyI
5oV++dihMbb/MA4z59RfVGsJWWUz3Tlmyeix0Gl48hXEr3SuHkoaSVnkIIISEyu2
vhVHE25RR/ACFs+wxVKeAxOe7ov6Wg5EX0Df92mRcWuUbDKsV4i3
-----END CERTIFICATE-----
```

V souboru jsou dvě hlavní části, které jsou odděleny speciálními značkami. Jedná se o sekci, která obsahuje klíč a sekci, která obsahuje certifikát serveru. Také je možné mít klíč a certifikát uloženy v jednotlivých souborech. Tajný klíč se nachází mezi řetězci

`-----BEGIN RSA PRIVATE KEY-----` a

`-----END RSA PRIVATE KEY-----`

zatímco certifikát je uveden mezi

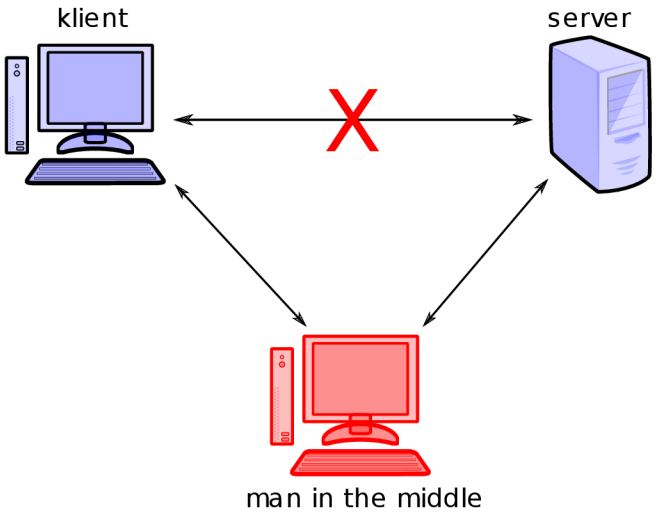
`-----BEGIN CERTIFICATE-----` a

`-----END CERTIFICATE-----`.

Self-signed certifikát

Pro správu, distribuci a uchování klíčů se využívá služeb certifikačních autorit (CA). Ty vstupují jako třetí strana při komunikaci dvou subjektů a svazují dvojici klíčů s jejím vlastníkem. Tento svazek je obsažen v certifikátu. Každá CA má vlastní politiku pro vydávání certifikátů a velkou roli hraje ověření totožnosti vlastníka. Existuje mnoho certifikačních autorit, které za poplatek poskytují své služby, ale vzhledem k tomu, že se jedná o komerční subjekty a za tyto služby se platí, vytvořil jsem si certifikát podepsaný sám sebou.

Komunikaci přes protokol https využívající self-signed certifikát můžeme považovat za bezpečnou proti odposlouchávání (eavesdropping), nikoli však proti útoku zvaného *muž uprostřed* (man in the middle), dále jen MITM. Schéma tohoto útoku lze vidět na obrázku 3.5



Obr. 3.5: *Schéma útoku man in the middle.*

Jak je vidět, útočník je středem komunikace mezi klientem a serverem. V případě komunikace přes *http* je tento typ útoku snazší než přes *https*. Pokud chce útočník napadnout *https* komunikaci pomocí techniky MITM, musí klientovi podvrhnout certifikát. Ve chvíli, kdy dochází k šifrované komunikaci prostřednictvím nedůvěryhodného certifikátu je v prohlížeči uživatele zobrazeno varovné okno. Většinou uživatel certifikát nekontroluje a povolí jej. Tento certifikát však může být právě od útočníka. Proto je důležité, aby si uživatel případný certifikát zkontroloval.

Se serverem pak útočník komunikuje s jeho certifikátem. Self-signed certifikát tedy není proti tomuto útoku odolný, protože není podepsaný žádnou důvěryhodnou

CA. Pro skutečně bezpečné aplikace je třeba zajistit certifikát podepsaný veřejnou CA, která identifikuje účastníky šifrované komunikace. V tomto případě se na straně klienta nezobrazují hlášení o nedůvěryhodném certifikátu a certifikát je považován za důvěryhodný. V momentě útoku se na straně klienta zobrazí opět varovné hlášení, že certifikát je nedůvěryhodný, což může značit útok MITM.

Jednou z možností je také vytvoření certifikátů pro každého klienta systému zvlášť a naimportovat jej do jeho počítače. Toto řešení se využívá spíše ve firemních sítích.

3.3.6 Dodatečná zabezpečení

Ve své aplikaci používám dodatečné zabezpečení pomocí *sessions*. Sessions jako takové nejsou určeny přímo k zabezpečení relace, ale řeší bezstavovost HTTP protokolu. Dají se využít ke sledování pohybu klienta na serveru. Data spojená s probíhající session jsou umístěna většinou v souboru na serveru, a to mimo kořenový adresář serveru. Není však výjimkou ukládat sessions do databáze.

Jedná se o mechanismus, pomocí kterého můžeme například kontrolovat relaci přihlášeného uživatele. Pomocí nich můžeme také přenášet spoustu proměnných, aniž bychom je uvedli v URL stránky. Co se týče přenášení některých parametrů v URL, tak je to ne příliš bezpečné, neboť URL lze jednoduše změnit v prohlížeči.

Příklad nevhodného předávání parametrů v URL:

```
http://www.server.cz/login.php?user=uzivatel
```

V URL předávám parametr *user*, který můžu samozřejmě změnit, a tím se dostat k datům jiného uživatele. Toto je samozřejmě krajní případ nevhodného použití.

Jak již bylo řečeno, v realizované aplikaci používám k zabezpečení především právě sessions, a to následujícím způsobem:

- při přihlášení uživatele do systému vytvořím systém proměnných, které při každém dalším požadavku klienta kontroluji
- jedná se o kontrolu IP adresy, z které se uživatel přihlásil a také kontrola prohlížeče, který použil
- pokud tyto informace při kontrole nesouhlasí, je uživatel automaticky odhlášen

- dalším dodatečným zabezpečením je ošetření nečinnosti klienta, a to jak pomocí session, kdy při každém novém požadavku zkontroluji čas nečinnosti, tak pomocí javascriptu, kdy uživatele automaticky odhlásím po určité době

Tímto jsem docílil poměrně vysoké úrovně zabezpečení. Při realizaci posledního prvku zabezpečení aplikace jsem vycházel z funkce session_regenerate_id, která automaticky mění tzv. SESSION ID relace.

Vytvořil jsem funkci, která vygeneruje při každém požadavku klienta nové id. Toto id je předáváno v url jako parametr *sessid* a je vždy kontrolován. Toto id je hash (SHA512) z vygenerovaného řetězce. Řetězec vytvořím následujícím způsobem:

- při přihlášení uživatele vygeneruji náhodný řetězec o délce 64 znaků vlastní funkcí `sess_regen_id()`
- k řetězci přidám identifikaci prohlížeče a `funkcí session_id()` vygenerované id relace
- z takto složeného řetězce vytvořím hash a předávám ho v URL jako parametr sessid

Takto vytvořený řetězec mimo jiné zabezpečuje podvržení session, v které předávám *login* přihlášeného uživatele. Pokud by se útočník pokusil podvrhnout jinou hodnotu, hash se samozřejmě změní a systém uživatele odhlásí.

3.4 Navržená aplikace

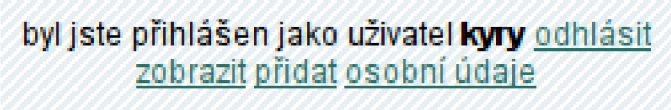
3.4.1 Uživatelská část

Již jsem popsal zabezpečení aplikace a nyní ukáži, jak výsledná aplikace vypadá. Předpokladem pro přihlášení je samozřejmě dokončená registrace včetně potvrzení registračním emailem. Po přihlášení se uživateli zobrazí nabídka, která je vidět na obrázku 3.6.

Uživatel má možnosti:

- odhlásit - tímto se uživatel odhlásí ze systému

- zobrazit - zde uživatel může zobrazit a editovat svá privátní data
- přidat - v této sekci přidává nové záznamy do databáze
- osobní údaje - zobrazí údaje o uživateli zadané při registraci



Obr. 3.6: *Nabídka pro přihlášeného uživatele.*

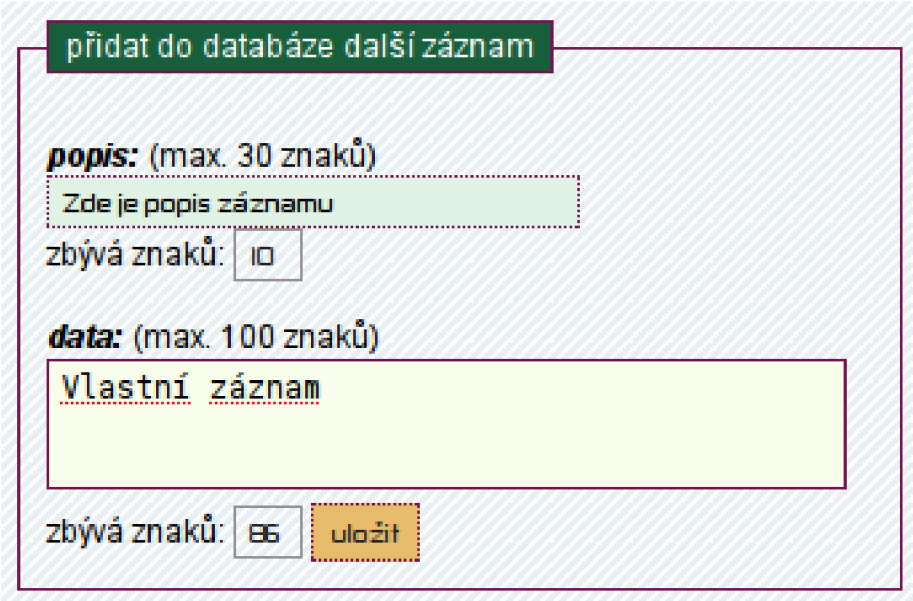
Sekci *přidat* můžeme vidět na obrázku 3.7. Vidíme, že uživatel má možnost zadat 2 údaje, a to **popis** a **data**. Tyto údaje mají omezenou délku, což jsem ošetřil javascriptem. Vlastní přidání záznamů do databáze probíhá prostřednictvím technologie AJAX, která se používá ke změně obsahu stránek bez nutnosti jejich opětovného načtení. Je založena na javascriptu a při zavolání javascriptové funkce **add_new_data** zahájím interakci mezi klientem a serverem. Skript převezme z formuláře hodnoty, které uživatel zadal a metodou POST je předá skriptu **ajax.add.new.data.php**. Skript nejdříve zkontroluje, zda jde o oprávněný požadavek kontrolou sessions a pokud je vše v pořádku, vloží data do databáze. Jako znamení o přidání vrátí řetězec *přidáno*, který se uživateli zobrazí. Řetězec „Zde je popis záznamu“ je v databázi uložen takto:

```
aNNXpExTTCWvg8NvJHwNPF2RdROWcqMGkeNj2GRfWEgTrpr03jzQhyPUGOYiDY
zpAa2WCKi+RrmyOVwwT/QqgjOODBGU7tWofEBoIJ7AhhFSai9DElefWBcNrva/
SD5xDC72tcr+2Op3XRtJuxWwkWAWqnuLQay5cn1UuWoxSLo=
```

a řetězec „Vlastní záznam“ takto:

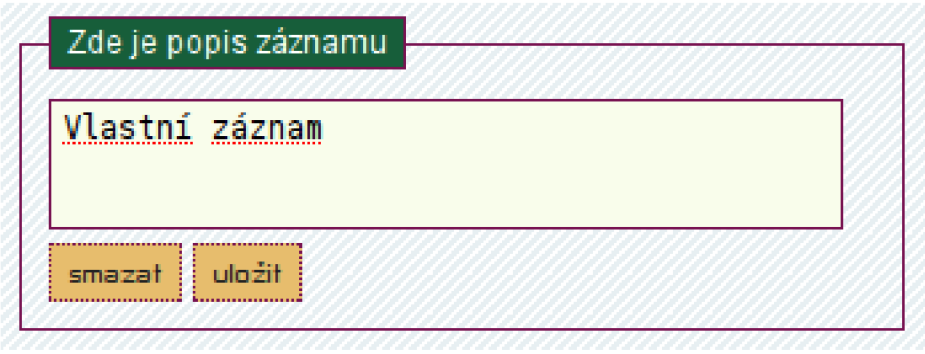
```
iESMafG3HQJSmhuD8A/MPXn1d/osYpgzu8xyQD7kfCjs3DiPenwZHtG5NgqJk
gIgUxsBCTC28cIqqdsKCJb/Vvr2jTIsTLcdAM8DX2I8Bt33S/BNrTPsWP8Y9+
5rd4569aRDVGoq44iWqZQqaV+7NvRT3hVfyvEQOv5iegMM7Xw=
```

Z těchto ukázek je zřejmé, že pokud by se někdo dostal k těmto datům a nedisponoval klíčem k jejich rozšifrování, jsou pro něho naprosto nepoužitelná a zašifrovanou informaci nemá šanci získat.



Obr. 3.7: *Přidání dat uživatele.*

Nyní má uživatel záznam v databázi a v sekci *zobrazit* si ho může prohlédnout. Na obrázku 3.8 je znázorněno, jakým způsobem jsou zobrazeny záznamy v aplikaci. Jsou zde samozřejmě v původním tvaru.



Obr. 3.8: *Zobrazení dat uživatele.*

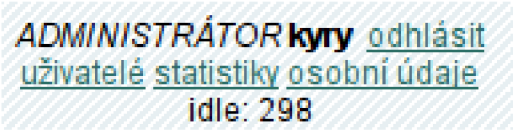
Také jsem ošetřil ztrátu hesla uživatele. Ten má možnost vyplnit údaje, které zadával při registraci a pokud tyto údaje souhlasí, je uživateli odeslán email s novým

heslem.

3.4.2 Administrátorská část

Součástí aplikace je i administrátorská část, kde si může administrátor prohlédnout statistiky uživatelů a také v ní může vidět jejich záznamy v původním (rozšifrovaném) tvaru. Na obrázku 3.9 vidíme menu administrátora. Tím může být každý uživatel, který má v tabulce *users* parametr *isadmin* nastaven na logickou 1. Ve výpisu se to projeví tak, že za uživatelským jménem uživatele je řetězec **(admin)**. Toto můžeme vidět na obrázku 3.10.

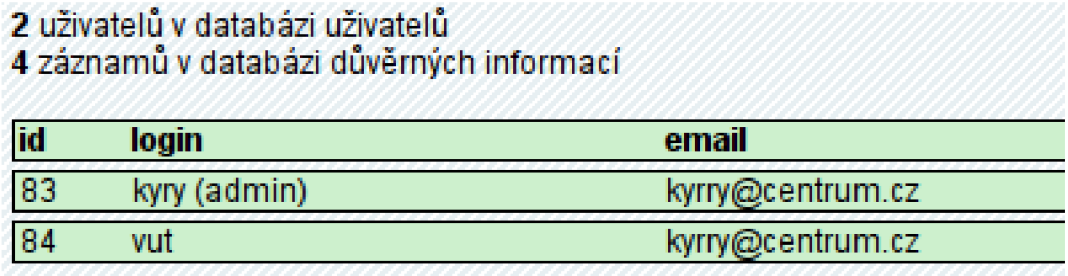
Správce má možnosti *odhlásit, uživatelé, statistiky* a *osobní údaje*. V osobních údajích jsou údaje o administrátorovi podobně jako v uživatelské sekci.



Obr. 3.9: *Menu administrátora.*

Na obrázku 3.10 vidíme sekci uživatelé. Jsou v ní vypsáni všichni zaregistrovaní uživatelé a nejdůležitější informace o nich. Je zde také vidět, kolik je v databázi uživatelů a kolik je záznamů důvěrných informací.

Obrázek 3.11 reprezentuje část *statistiky*. Zde si administrátor může přečíst důvěrná data uživatelů a kolik má každý uživatel záznamů.



Obr. 3.10: *Statistiky uživatelů.*

Obr. 3.11: *Záznamy uživatelů.*

3.5 Popis funkcí zdrojového kódu v PHP

Všechny funkce, které jsou potřeba k běhu programu jsou obsaženy v souboru **inc.func.php**. Některé z nich jsou uvedeny v přílohách a všechny jsou dostupné na přiloženém cd. Zde popíši nejdůležitější z nich a vysvětlím jejich význam.

`sess_regen_id()`

Jedná se o funkci, která vytvoří nové id relace. Využívá se ke generování proměnné sessid, což je popsáno v kapitole 3.3.6.

`time_expired($now)`

Pomocí této funkce kontroluji, zda nebyl uživatel déle než 300 vteřin nečinný. Funkci předávám parametr $now, který reprezentuje aktuální čas získaný funkcí `time()`. Pokud je rozdíl dvou požadavků větší než 300 vteřin, je uživatel odhlášen.

`gen_key($user)`

Touto funkcí vygeneruji pár klíčů pro uživatele předaného v parametru funkce. Toto je popsáno v kapitole 3.3.2.

`crypt_text($user,$text)`

Jako parametry této funkce jsou uživatel a text. Funkce šifruje veřejným klíčem pro daného uživatele řetězec zadaný v parametru. Tato funkce je uvedena v příloze A1.

`decrypt_text($user,$text)`

Inverzní funkce k předchozí funkci. Dešifruje privátním klíčem. Tato funkce je uvedena v příloze A2.

`SendActivatingEmail ($user, $email)`

Pošle aktivační email na adresu zadanou v parametru $email, parametr $user se předává z důvodu vytvoření aktivačního hashe (viz kapitola 3.3.1).

`create_salt_hash($password)`

Vytvoří z hesla $password hash využitím metody salted hash. Podrobněji popsáno v kapitole 3.3.1. Tato funkce je uvedena v příloze A3.

4 ANALÝZA BEZPEČNOSTI APLIKACE

V této kapitole bych chtěl analyzovat vytvořenou webovou aplikaci z hlediska bezpečnosti. Bezpečnost webových aplikací má mnoho hledisek a ve své práci jsem se snažil zaměřit na bezpečnost tak, aby případný útočník měl co nejmenší možnost získat důvědná data. Důležitá je komplexnost řešení a nelze se spoléhat jen na dílčí způsoby zabezpečení (pospáno v kapitole 3.2).

4.1 Uživatelské vstupy, SQL injection, XSS

Aplikace je koncipována tak, že registrovaným uživatelům umožňuje vkládat do systému důvěrná data. To mohou být například údaje o platební kartě, rodné číslo apod. Avšak v tomto bodě bylo třeba zajistit, aby případný útočník nemohl vložit konstrukci, která by při výpisu na stránce způsobila spuštění nežádoucího skriptu. Toto jsem ošetřil pomocí funkce `htmlspecialchars`, která možnost vložení skriptu znemožňuje. Tuto problematiku jsem popsal v kapitole 3.3.4.

SQL injection jsem řešil pomocí nastavení direktivy PHP *magic_quotes_gpc*, která doplňuje veškeré vstupy od uživatele uvozovkami a speciální znaky o tzv. escape sekvence (zpětné lomítko). Tímto jsem docílil toho, že data z uživatelských vstupů budou brána jako řetězec a nikoli další jako parametr SQL dotazu.

4.2 Ukradení spojení

Aby nemohl útočník ukrást spojení, využil jsem kontrolu identity uživatele pomocí IP. Do proměnné *$_SESSION["ip"]* je uložen hash (sha1) IP adresy přihlášeného uživatele a při přechodu na další stránku v systému je tato hodnota kontrolována. Také jsem vytvořil proměnnou *$_SESSION["agent"]*, kde je hash (sha1) identifikující prohlížeč, ze kterého uživatel na stránky přistoupil.

Dalším prvkem zabezpečení je generování dočasného identifikátoru, který je předáván v URL a je vytvářen hashem (SHA512) z náhodného řetězce. Tento řetězec jsem se snažil vytvořit tak, aby bylo prakticky nemožné mechanismus vytváření napodobit. Vygeneroval jsem 64 znaků dlouhý náhodný řetězec a přidal k němu identifikaci prohlížeče a id vygenerované PHP funkcí `session_id`. Platnost takto vygenerovaného identifikátoru je nejdéle 300 sekund (po 300 sekundách nečinnosti systém uživatele automaticky odhlásí).

4.3 Bezpečnost dat v databázi

Databázi jsem zabezpečil tak, aby bylo možno přistupovat k ní pouze z počítače na kterém je server (nastavením bind adresy). Tím jsem ošetřil případný útok zvenčí. Také jsem vytvořil nového uživatele MySQL databáze, který má omezený přístup k databázi. Tento uživatel má práva jen pro tabulky databáze diplomové práce (viz. kapitola 3.2.3).

Důležité bylo také zabezpečení hesla, které jsem ukládal do databáze jako salted hash. Tato metoda je popsána v kapitole 3.3.1.

4.4 Soubory na serveru

Pozornost jsem také věnoval uložení souborů na serveru. V adresáři webového serveru (/var/www) jsem nastavil pravidla tak, aby si nikdo nemohl v tomto adresáři listovat přes webový prohlížeč a při zadání URL vyžadující dokument, který na serveru neexistuje, je automaticky přesměrován na *index.php*.

Dále jsem uložil klíče mimo adresář webového serveru a nastavil práva tak, aby ho mohl číst pouze vlastník. Název souboru jsem volil tak, aby nebylo z názvu patrné, komu patří, čehož jsem dosáhl hashovací funkcí (viz. kapitola 3.3.2).

4.5 Zabezpečená komunikace

Server s klientem komunikují přes protokol https protokol. Použitý certifikát je podepsaný sám sebou (self-signed), a tudíž můžeme komunikaci považovat za bezpečnou proti odposlouchávání (eavesdropping), nikoli však proti technice man in the middle. Při této technice by mohl útočník klientovi podvrhnout vlastní certifikát a být prostředníkem komunikace mezi komunikujícími stranami. Pokud bychom chtěli systém ošetřit i proti tomuto útoku, je třeba nechat si vystavit certifikát ověřený veřejnou CA. Podrobněji popsáno v kapitole 3.3.5.

5 ZÁVĚR

Cílem práce bylo nastudovat možnosti systému OpenSSL v prostředí PHP a implementovat zabezpečení pomocí tohoto rozšíření na systém klient-server, který je určen k ukládání citlivých dat. Také bylo cílem zabezpečit tuto aplikaci proti celé škále útoků vedoucích k získání citlivých dat.

V této práci jsem popsal teorii kryptologie, její historii i současné použití. Vysvětlil jsem principy šifrování, a to jak symetrického, tak i asymetrickému. Tyto principy se dají využít k zabezpečení webových aplikací, což je v dnešní době nutností.

Dále jsem popsal knihovnu OpenSSL, která disponuje prostředky k symetrickému, tak i asymetrickému šifrování a vytvořil jsem v jazyce PHP aplikaci, která pomocí rozšíření právě o tuto knihovnu generovala pár veřejný/soukromý klíč pro uživatele a dle principů asymetrického šifrování šifrovala zadaný text. Také jsem zmínil teorii bezpečnosti webových aplikací, kterou popisuji v kapitolách 1.4.3, 1.4.4 a 1.4.5.

Praktickou část realizované aplikace jsem popsal v kapitole 3. Zde jsem mimo jiné zmínil způsoby zabezpečení proti celé škále útoků, jako například Cross-Site-Scripting a nejdůležitější části zdrojového kódu jsem uvedl v přílohách. K zabezpečení jsem využil možností nastavení direktiv webového serveru Apache, direktiv skriptovacího jazyka PHP, nastavení přístupu do databáze, rozšíření OpenSSL v PHP, ošetření vstupu dat od uživatele funkcí *htmlspecialchars*, zabezpečenou komunikaci (protokol https) a také systém proměnných sessions.

Žádná aplikace nemůže být vždy naprosto bezpečná, neboť nové způsoby útočníků narušit běh aplikace nebo získat z ní citlivá data se vyvíjejí stejně tak, jako mechanismy pro jejich zabezpečení. Proto si nemůžeme být vždy jisti, jakou novou metodu útočník použije. Avšak díky výše zmíněným principům se dají rizika minimalizovat.

LITERATURA

[1] CHANDRA, P., MESSIER, M., VIEGA, J. *Network Security with OpenSSL.* O'Reilly, 2002. 384 s. ISBN: 0-596-00270-X

[2] KABIR, M. J. *APACHE 2 SERVER - Kompletní příručka administrátora.* Computer Press, 2004. 722 s. ISBN: 80-251-0319-6

[3] MALÝ, J., KACÁLEK, J. *Zabezpečení webových aplikací I. - klientské skriptovací jazyky* [online].
ISSN: 1214-9675. Dostupné z URL:
<http://access.feld.cvut.cz/view.php?cisloclanku=2007090001>

[4] MALÝ, J., KACÁLEK, J. *Zabezpečení webových aplikací II. - databáze* [online].
ISSN: 1214-9675. Dostupné z URL:
<http://access.feld.cvut.cz/view.php?cisloclanku=2007080002>

[5] *Kryptologie*, Univerzita Hradec Králové [online]. Dostupné z URL:
<http://kryptologie.uhk.cz>

[6] *OpenSSL funkce v PHP* [online]. Dostupné z URL:
<http://cz2.php.net/openssl>

[7] MENEZES, A. J., VAN OORSCHOT, P. C., VANSTONE, S. A. *Handbook of Applied Cryptography.*
CRC Press, 1996. 816 s. ISBN: 0-8493-8523-7

[8] *Jak na OpenSSL II*, Root.cz [online]. Dostupné z URL:
<http://www.root.cz/clanky/jak-na-openssl-2/>

[9] HOUSLEY, R., FORD, W., POLK, W.,SOLO., D. *Internet X.509 Public Key Infrastructure Certificate and CRL Profile*, RFC 2459.
Internet Engineering Task Force, 1999.

[10] HUSTON, G. *Security Implications of Using the Data Encryption Standard (DES)*, RFC 4772.
Internet Engineering Task Force, 2006.

[11] CHOWN, P. *Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS)*, RFC 3268.
Internet Engineering Task Force, 2002.

[12] KALISKI, B., STADDON, J. *PKCS #1: RSA Cryptography Specifications Version 2.0*, RFC 2437.
Internet Engineering Task Force, 1998.

[13] RIVEST, R. *The MD5 Message-Digest Algorithm*, RFC 1321.
Internet Engineering Task Force, 1992.

[14] EASTLAKE 3rd, D., JONES, P. *US Secure Hash Algorithm 1 (SHA1)*, RFC 3174.
Internet Engineering Task Force, 2001.

[15] BERNERS-LEE, T., CONNOLLY, D. *Hypertext Markup Language - 2.0*, RFC 1866.
Internet Engineering Task Force, 1995.

[16] RESCORLA, E. *HTTP Over TLS*, RFC 2818.
Internet Engineering Task Force, 2000.

[17] FIELDING, R., GETTYS, J., MOGUL, J., FRYSTYK, H., MASINTER, L., LEACH, P., BERNERS-LEE, T. *Hypertext Transfer Protocol – HTTP/1.1*, RFC 2616.
Internet Engineering Task Force, 1999.

[18] DIERKS, T., ALLEN, C. *The TLS Protocol Version 1.0*, RFC 2246.
Internet Engineering Task Force, 1999.

SEZNAM PŘÍLOH

A PŘÍLOHY

A.1 Funkce pro šifrování

```
function crypt_text($user,$text)
{

$_1 = sha1($user.'_pub.pem');
$_2 = sha1($user.'_pri.pem');
$_3 = sha1($user.'_csr.pem');

$publickey_path = '/home/apache/'.$_1;
$privatekey_path = '/home/apache/'.$_2;
$csrStr_path = '/home/apache/'.$_3;

$fp=fopen("$publickey_path","r");
$public_key=fread($fp,8192);
fclose($fp);

openssl_public_encrypt($text,$crypttext,$public_key); //zašifruju veřejným klíčem
$crypted = base64_encode($crypttext);

return $crypted;
}
```

A.2 Funkce pro dešifrování

```
function decrypt_text($user,$crypttext)
{
$_1 = sha1($user.'_pub.pem');
$_2 = sha1($user.'_pri.pem');
$_3 = sha1($user.'_csr.pem');

$publickey_path = '/home/apache/'.$_1;
$privatekey_path = '/home/apache/'.$_2;
$csrStr_path = '/home/apache/'.$_3;


$fp=fopen("$privatekey_path","r");
$private_key=fread($fp,8192);
fclose($fp);

$res = openssl_get_privatekey($private_key,'1234');

$crypttext = base64_decode($crypttext);

openssl_private_decrypt($crypttext,$decrypted,$res); //rozšifruje


return $decrypted;
}
```

A.3 Salted hash

```
function create_salt_hash($password)
{   $salt1 = get_salt(); //funkce pro generování náhodného řetězce
    $salt2 = get_salt();
    $salt3 = get_salt();

//vlozeni saltu dovnitr hesla
    $pass_array = str_split($password);
    $max = strlen($password)-1;
    $random = mt_rand(1,$max);
    $out="";
    for ($i=0; $i<=$max; $i++):
    if ($i==$random){$out=$out.$salt1;}
    $out=$out.$pass_array[$i];
    endfor;

//přidání saltu na začátek a konec
    $tohash=$salt2.$out.$salt3;
    $hash=sha1($tohash);
    return Array(1=>$salt1, 2=>$salt2, 3=>$salt3, 4=>$random, 5=>$hash);

}
```

B OBSAH PŘILOŽENÉHO CD

- **pdf/** - adresář obsahující elektronickou verzi této diplomové práce
- **php/** - adresář obsahující zdrojové kódy diplomové práce