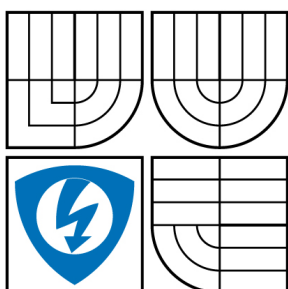


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH
TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

MODERNÍ METODY OVĚŘENÍ IDENTITY UŽIVATELŮ

MODERN METHODS FOR USER AUTHENTICATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

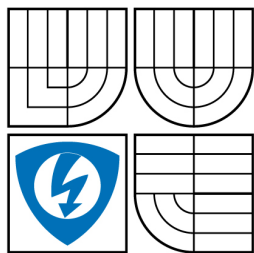
AUTOR PRÁCE
AUTHOR

Bc. DANIEL SÝKORA

VEDOUcí PRÁCE
SUPERVISOR

Ing. JAN HAJNÝ

BRNO 2009



VYSOKÉ UČENÍ
TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student: Bc. Daniel Sýkora

ID: 48015

Ročník: 2

Akademický rok: 2008/2009

NÁZEV TÉMATU:

Moderní metody ověření identity uživatelů

POKYNY PRO VYPRACOVÁNÍ:

Popište současné možnosti ověření uživatele pomocí hesla. Najděte slabiny těchto systémů a vyhledejte aktuální metody a trendy v oblasti. Analyzujte metody s nulovou znalostí, Password-based autentizaci a vyhledejte další nové protokoly. Pokuste se srovnat bezpečnost těchto řešení. Zvolte nejvhodnější metodu pro praktickou implementaci. V praxi ověřte zvolené implementace, popište podrobně všechna přenášená data a udělejte bezpečnostní analýzu protokolů.

DOPORUČENÁ LITERATURA:

- [1] SMITH, Richard E. Authentication: From Passwords to Public Keys. [s.l.] : [s.n.], 2001. 576 s.
- [2] DOSTÁLEK, Libor. Velký průvodce protokoly TCP/IP - Bezpečnost. [s.l.] : [s.n.], 2003. 571 s.

Termín zadání: 9.2.2009

Termín odevzdání: 26.5.2009

Vedoucí práce: Ing. Jan Hajný

prof. Ing. Kamil Vrba, CSc.

Předseda oborové rady

UPOZORNĚNÍ:

Autor diplomové práce nesmí při vytváření diplomové práce porušit autorská práva třetích osob, zejména nesmí zasahovat nedovoleným způsobem do cizích autorských práv osobnostních a musí si být 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.

ABSTRAKT

Diplomová práce se zabývá moderními metodami autentizace uživatelů. V první části práce jsou stručně popsány protokoly, které se používají v současnosti a shrnuty jejich výhody a nevýhody. V teoretickém úvodu jsou pak rozebrány principy autentizace s nulovou znalostí, password-based protokoly a popsán návrh hašovací funkce nové generace. V praktické části jsou popsány konkrétní implementace autentizačních protokolů - Ohta-Okamoto protokol jako zástupce protokolů s nulovou znalostí a SRP (Secure Remote Password), který zastupuje password-based protokoly. V obou případech je popsán postup instalace, proveden rozbor jejich implementace na úrovni zdrojových kódů a ten následně porovnán s přenášenými daty zachycenými programem Wireshark. U protokolu SRP je provedeno ověření bezpečnosti nástrojem AVISPA. V závěru je u obou protokolů shrnuta bezpečnostní analýza.

KLÍČOVÁ SLOVA

autentizace, Ohta-Okamoto, password-based, SRP, Wireshark, AVISPA

ABSTRACT

The main focus of Master's thesis is modern methods for user authentication. In the first part are briefly described currently used protocols and pointed out their advantages and disadvantages. The theoretical introduction analyzes the principles of zero-knowledge authentication, password-based protocols and describes the concept of a new generation hash function. The practical part describes the specific implementation of authentication protocols - Ohta-Okamoto protocol as a representative of the zero-knowledge protocols and SRP (Secure Remote Password), which represents password-based protocols. In both cases, the installation procedure is described following the analysis of their implementation (at the source code level) and then compared with the transmitted data captured by Wireshark. The SRP protocol is verified by AVISPA tool. There is summary of both protocols security analysis in the conclusion.

KEYWORDS

authentication, Ohta-Okamoto, password-based, SRP, Wireshark, AVISPA

SÝKORA, D. *Moderní metody ověření identity uživatelů*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2009. 69 s. Vedoucí diplomové práce Ing. Jan Hajný.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Moderní metody ověření identity uživatelů“ 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 20. 5. 2009

.....
(podpis autora)

PODĚKOVÁNÍ

Děkuji vedoucímu diplomové práce Ing. Janu Hajnému za velmi užitečnou metodickou pomoc a cenné rady při zpracování diplomové práce.

V Brně dne 20. 5. 2009

.....
(podpis autora)

Obsah

Úvod.....	10
1 Současné autentizační metody a protokoly.....	11
1.1 Protokol Kerberos.....	11
1.2 Protokol RADIUS	12
1.3 Protokol Diameter	13
1.4 Shrnutí	13
2 Protokoly s nulovou znalostí.....	14
2.1 Základní princip a důkazy	14
2.2 Fiat-Shamir identifikační protokol	16
2.3 Ohta-Okamoto identifikační protokol s nulovou znalostí	17
3 Password-based protokoly	19
3.1 SRP protokol	20
3.2 SNAPi protokol	23
3.2.1 Popis kryptografické techniky a nastavení parametrů protokolu.....	23
3.2.2 Výhody a nevýhody protokolu.....	25
4 Hašovací funkce v autentizaci	26
4.1 Vlastnosti hašovací funkce	26
4.2 Bezpečnost hašovacích funkcí.....	27
4.3 Moderní hašovací funkce	28
4.3.1 Damgard-Merklovův iterativní princip moderních hašovacích funkcí.....	28
4.3.2 Davies-Meyerova konstrukce kompresní funkce.....	28
4.3.3 Bezpečnost současných hašovacích funkcí.....	29
4.4 Nový koncept konstrukce hašovacích funkcí	29
4.4.1 HMAC a NMAC	29
4.4.2 Speciální bloková šifra a SNMAC.....	30
4.5 DN, HDN.....	31
4.6 Shrnutí	33
5 ZK-SSH - metoda s nulovou znalostí pro OpenSSH.....	34
5.1 OpenSSH	34
5.2 Instalace OpenSSH s autentizací metodou nulové znalosti.....	35
5.3 Analýza implementace metody s nulovou znalostí	37
5.4 Rozbor komunikace klienta se serverem.....	42
5.5 Rozbor přenesených dat	45
6 Autentizační metoda založená na protokolu SRP.....	48
6.1 Instalace programů FTP a telnet s podporou SRP autentizace.....	48

6.2	Analýza implementace SRP autentizace	51
6.3	Rozbor zachycených dat během vytvoření FTP spojení	54
6.4	Bezpečnostní analýza protokolu SRP.....	58
6.4.1	Protokol SRP v jazyce HLPSL	60
6.4.2	Výsledky hodnocení bezpečnosti nástrojem SPAN.....	61
7	Zhodnocení bezpečnosti zvolených protokolů	62
7.1	Autentizační metoda s nulovou znalostí.....	62
7.2	Autentizace protokolem SRP	63
8	Závěr	64
	Literatura.....	65
	Seznam zkratek.....	68
A	Obsah přiloženého média.....	69

Seznam obrázků

Obr. 1.1: Struktura sítě Kerberos	11
Obr. 1.2: Struktura sítě RADIUS	12
Obr. 2.1: Princip nulové znalosti, vlevo a) volba cesty, vpravo b) před dveřmi	14
Obr. 2.2: Princip nulové znalosti, P zná heslo	15
Obr. 4.1: Vlastnost jednosměrnosti funkce	26
Obr. 4.2: Bezkoliznost 1. řádu	26
Obr. 4.3: Bezkoliznost 2. řádu	27
Obr. 4.4: Damgard-Merklova iterativní hašovací funkce	28
Obr. 4.5: Davies-Meyerova konstrukce kompresní funkce	28
Obr. 4.6: Definice hašovací funkce NMAC	30
Obr. 4.7: Definice hašovací funkce HMAC	30
Obr. 4.8: Definice SNMAC, založená na SBŠ a NMAC	31
Obr. 4.9: Obecné schéma DN	31
Obr. 4.10: Sloupcová transformace	32
Obr. 5.1: Nastavení zachytávání paketů v programu Wireshark	45
Obr. 5.2: Navázání TCP spojení	46
Obr. 5.3: Nešifrovaná data v paketech č. 8 a 9	46
Obr. 5.4: Zachycené pakety procesu autentizace	47
Obr. 6.1: Připojení klienta k FTP serveru	55
Obr. 6.2: Výpis dat první části vytvoření FTP spojení programu Wireshark	56
Obr. 6.3: Zadání uživatelského jména na straně FTP klienta	56
Obr. 6.4: Zachycené pakety druhé části procesu FTP spojení	57
Obr. 6.5: Závěrečná fáze FTP spojení	57
Obr. 6.6: Zachycené pakety v závěrečné fázi FTP spojení	58
Obr. 6.7: Výměna zpráv v nástroji SPAN pomocí MSC	59
Obr. 6.8: Hlavní okno nástroje SPAN	60

Úvod

S problematikou ověřování identity uživatelů neboli jejich autentizací se v dnešním světě moderních informačních technologií setkáváme stále častěji. Běžný uživatel přichází každý den do styku s různými systémy, které neposkytují služby každému, ale pouze určité skupině uživatelů, kteří se musí patřičně autentizovat. Nároky nejsou kladeny pouze na správce těchto systémů a služeb, kteří musí zajistit dostatečnou úroveň zabezpečení pomocí vhodných metod, ale rovněž na samotné uživatele. Ačkoli se zdá být prokázání své identity například použitím hesla triviální a bezpečné, s rostoucím výkonem současných výpočetních systému stoupá i počet dříve pouze teoreticky možných útoků a metody, které byly považovány za bezpečné již byly prolomeny, případně se nedoporučuje jejich použití v systémech vyžadující nejvyšší úroveň zabezpečení.

Cílem této diplomové práce je prostudovat moderní metody autentizace uživatelů a zhodnotit jejich výhody a případně nedostatky. Konkrétně se pak zaměřit na metodu s nulovou znalostí a password-based autentizaci a vybrat mezi nimi vhodnou metodu pro praktickou implementaci. Tu v praxi ověřit a podrobně popsat všechna přenášená data a na závěr provést bezpečnostní analýzu použitých protokolů.

V první části teoretického úvodu je stručně popsána problematika autentizace a jsou uvedeny protokoly, které se používají v současnosti, včetně zdůraznění jejich hlavních nedostatků. V dalších třech kapitolách jsou pak postupně rozebrány protokoly s nulovou znalostí, password-based protokoly a použití hašovacích funkcí v autentizaci. U každé metody jsou uvedeny konkrétní příklady použitých protokolů.

V praktické části jsou tyto protokoly podrobně rozebrány. Postupně je u každé implementace uveden návod k instalaci, provedena analýza zdrojových kódů a rozebrány přenášená data. U metody autentizace s nulovou znalostí jsou navíc uvedeny výpisy tzv. *debug* zpráv mezi klientem a serverem. U protokolu SRP je popsáno ověření bezpečnosti nástrojem AVISPA. V závěrečné kapitole je shrnutí bezpečnostní analýzy obou protokolů, resp. autentizačních metod založených na těchto protokolech.

1 Současné autentizační metody a protokoly

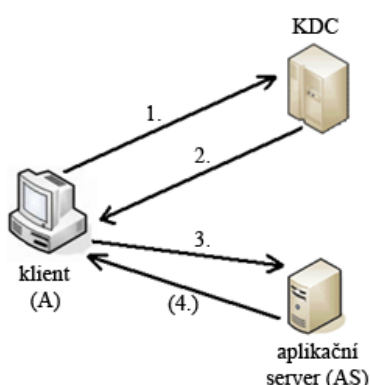
Autentizace je proces, při kterém se ověřuje, zda je uživatel nebo entita opravdu ten, za koho se vydává. Rozlišujeme tři, resp. čtyři způsoby, jak je možné provést uživatelskou autentizaci:

- podle toho, co uživatel zná (heslo, PIN,...),
- podle toho, co uživatel má (například identifikační kartu),
- podle toho, čím uživatel je (biometrické vlastnosti uživatele - otisk prstu, snímek oční duhovky či sítnice apod.),
- podle toho, co uživatel umí (umí správně odpovědět na náhodně vygenerovaný kontrolní dotaz).

V praxi je nejrozšířenější první typ a tedy používání hesla případně v kombinaci s porovnáváním biometrických údajů, kdy mluvíme o tzv. *vícefaktorové autentizaci* [1]. Metoda s použitím hesla v sobě nese výhodu v podobě jednoduché implementace do různých typů aplikací, ale současně nevýhody v náročné správě hesel, zapomenutí hesla uživateli, náchylnost na odposlech a další. Přesto se používá v drtivé většině aplikací. V další části uvedu nejčastěji používané protokoly v současné době.

1.1 Protokol Kerberos

Kerberos je síťový autentizační protokol založený na principech symetrické kryptografie a tedy sdíleného tajemství. Nezbytnou součástí struktury sítě je tzv. *KDC* (Key Distribution Center), server Kerberos, který bezpečně uchovává data o uživateli a službách v síti a zajišťuje bezpečnou distribuci klíčů. Princip činnosti autentizace uživatele se skládá z výměny zpráv mezi klientem (A), aplikačním serverem (S) a autentizačním serverem (KDC) a je uveden pod obr. 1.1, který zobrazuje strukturu sítě Kerberos. Detaily jednotlivých zpráv a podrobný popis protokolu je možné najít v RFC 1510 [23], nebo v [3].



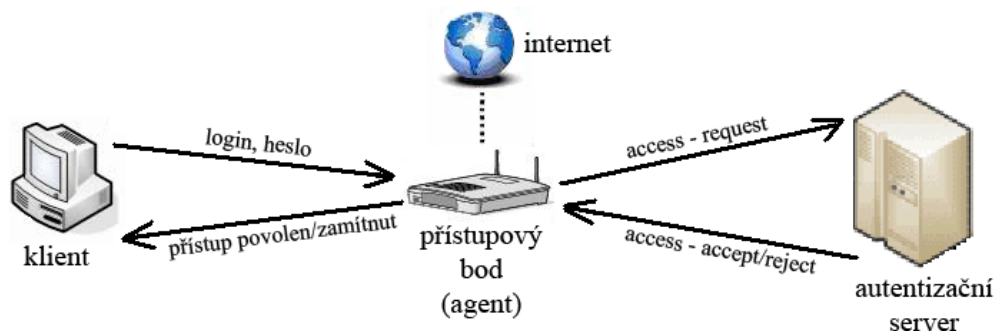
Obr. 1.1: Struktura sítě Kerberos

Princip je následující: Klient požaduje přístup k určité službě, kterou poskytuje aplikační server. Nejdříve se ale musí úspěšně autentizovat u autentizačního serveru *KDC*, který klientovi vydá tzv. *tiket*. Ten obsahuje klíč relace mezi klientem a aplikačním serverem, přihlašovací údaje klienta a hodnotu *L*, což je čas vypršení platnosti klíče relace. Výhodou použití tiketu je, že jednak má omezenou platnost trvání

a pak že informace určené pro klienta jsou šifrovány sdíleným klíčem mezi klientem a KDC, kdežto informace pro aplikační server jsou šifrovány sdíleným klíčem mezi aplikačním serverem a KDC. Navíc po ustanovení klíče relace se již tajný uživatelův klíč nepoužívá.

1.2 Protokol RADIUS

RADIUS (Remote Authentication Dial In User Service) je síťový autentizační protokol určený k řízení vzdáleného přístupu. V současné době je běžně zabudován do síťových prvků typu router či switch od řady výrobců a prakticky určuje standard pro vzdálenou autentizaci. Oproti protokolu Kerberos zajišťuje autentizaci klienta pro vstup do sítě, kdežto Kerberos se používá uvnitř sítě pro autentizaci přístupu ke službám. Vznikl na základě potřeby centralizované správy uživatelských údajů klientů, kteří se pomocí přihlašovacích údajů a hesla snaží připojit k internetu či naopak do privátní sítě. Uchovávat tyto údaje a jejich změny od tisíců či milionů uživatelů v síťových prvcích není reálné. Proces autentizace uživatele probíhá obdobně jako u protokolu Kerberos výměnou zabezpečených zpráv, jak je možné vidět na obr. 1.2.



Obr. 1.2: Struktura sítě RADIUS

Každá radius zpráva (paket) se skládá z těchto částí:

- *Code* - určuje typ zprávy (Access-Request, Access-Accept, Access-Reject a další).
- *Identifier* - identifikátor zprávy pro porovnání odpovídající odpovědi k požadavku.
- *Length* - délka zprávy.
- *Authenticator* - jedinečné náhodné číslo délky 16 oktetů (128 bitů). Slouží pro zabezpečení proti opakovaným útokům.
- *Attributes* - doplňující atributy, které v případě zprávy Access-Request jsou:
 - *User-Name* - jméno uživatele,
 - *User-Password* - je zašifrované heslo, které vznikne sřetězením $c_1, c_2, \dots, c_n$, kde $c_n = P_n \text{ XOR MD5}(S + c_{n-1})$. c_0 je rovno autentizační hodnotě, P_n jsou 128 bitové bloky uživatelského hesla (pouze v případě, že je heslo delší než daných 128 bitů) a S je sdílené heslo serverem. Pouze tato část zprávy je zabezpečena, zbytek je nechráněný.

Protokol RADIUS poskytuje určitou ochranu proti odposlouchávání a aktivním útokům, která je ve většině případů dostačující. Jsou však známy některé jeho slabiny a je možné na něj vést útoky ať už ze strany uživatele, či serveru. O některých konkrétních útocích je

možné se dočíst v [25], celá specifikace je poté v RFC 2865 [24]. Přesto je dnes protokol stále běžně používán, i když už existuje jeho zdokonalený nástupce v podání protokolu Diameter.

1.3 Protokol Diameter

Protokol Diameter je používán pro přístup k síti nebo pro IP mobilitu. Hlavní koncept tvoří základní protokol, který může být rozšířen pro poskytování AAA (*authentication, authorization and accounting*, česky *autentizace, autorizace a účtování*) služeb novým přístupovým technologiím. Může pracovat jak lokálně tak i v roamingu. Je odvozen z protokolu RADIUS, proti kterému obsahuje řadu vylepšení. Není s ním sice zpětně kompatibilní, ale má implementovanu jeho podporu. Principiálně se od něj moc neliší, proto uvedu pouze některé z nejdůležitějších rozdílů, podrobný popis je možné najít v RFC 3588 [27].

Hlavní rozdíly mezi protokoly Diameter (D) a RADIUS (R):

- D používá spolehlivý transportní protokol TCP (případně SCTP), R používá UDP.
- D podporuje zprávy inicializované serverem (například zprávu ukončení relace).
- D má větší adresní prostor pro atributy - 16,777,215 oktetů oproti 255 oktetům u R.
- D podporuje speciální uzly nazývané agenty, kteří zajišťují činnosti snadno odvoditelných z jejich názvů - Relay, Proxy, Redirect a Translation.
- R i D podporují statickou konfiguraci uzlů, D navíc dynamické vyhledávání uzlů.
- D umožňuje oznamování chyb.
- D je snadno rozšiřitelný o nové příkazy a atributy.

1.4 Shrnutí

Výše popsané protokoly jsou v současnosti brány jako standardy, které se dnes velmi často používají a to i přesto, že nemají prokazatelné důkazy bezpečnosti a skrývají řadu slabin. Příkladem může být použití hašovací funkce MD5 u protokolu RADIUS, u které jsou již delší dobu známy úspěšné útoky, a proto už není považována za bezpečnou. Otázkou ale zůstává, jakou úroveň zabezpečení požadujeme. Stejně tak se v posledních letech masivně rozšířilo používání bezdrátových a mobilních zařízení, pro něž současné protokoly (s výjimkou Diameteru) nebyly primárně vyvinuty. Použití protokolu Kerberos je z jeho principu vhodné na zabezpečení uvnitř sítě, ale nikoli pro síť používající prostředky mobilní komunikace pro přístup do sítě. V další části této práce se proto zaměřím na nové moderní metody ověření uživatelů a jejich možné použití v budoucnosti.

2 Protokoly s nulovou znalostí

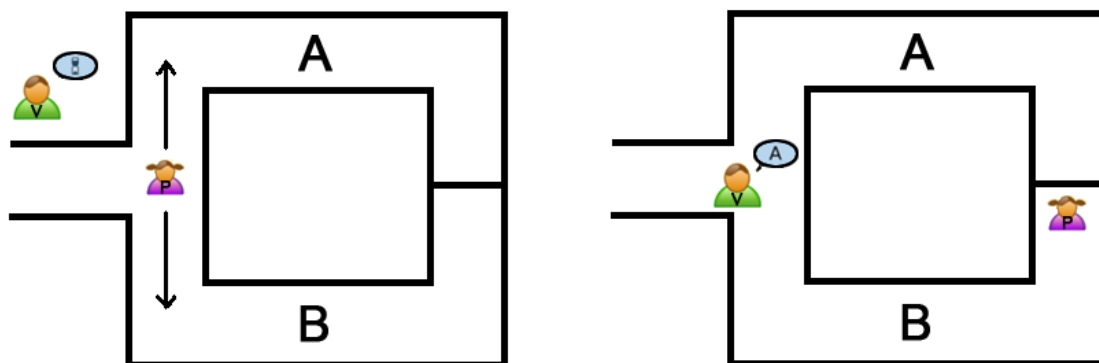
Téměř všechny autentizační protokoly jsou založeny na ověřování tajné informace jejím přenosem většinou nezabezpečenou sítí. To je ovšem nejrizikovější místo z hlediska bezpečnosti a tím pádem nejzranitelnější část pro případného útočníka. Znalost tajné informace (hesla) je totiž klíčová pro rozeznání správného uživatele od útočníka. I když se v protokolech uvedených v předchozí kapitole používají různé metody šifrování přenosu hesel, stále uniká určitá část informace. Přitom k ověření uživatele není potřeba přenášet celé tajné heslo, ale pouze informaci, zda uživatel toto heslo zná - přenáší se pouze 1 bit. Protokoly tudíž neznají uživatelské tajné heslo a proto jsou nazývány protokoly s nulovou znalostí. Vlastnictví hesla pouze uživatelem a nikoli serverem je důležitá bezpečnostní vlastnost těchto protokolů. V této kapitole uvedu základní princip a důkazy takovýchto protokolů a rozeberu několik konkrétních řešení.

2.1 Základní princip a důkazy

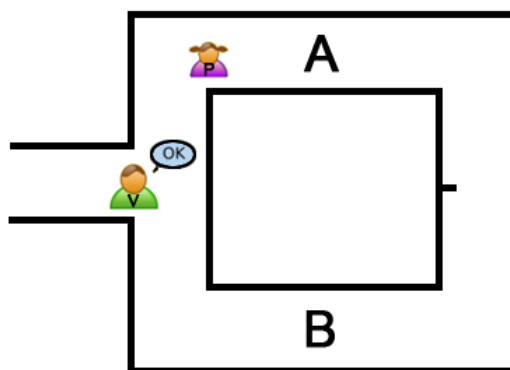
Pro pochopení základního principu protokolů s nulovou znalostí se v literatuře většinou používá schematické znázornění pomocí jeskyně [4]. V tomto příkladu vystupují dva subjekty - uživatel, který vlastní tajnou informaci ke svému ověření a server, který umožňuje přistupovat k určitým službám, ale až po ověření uživatele. Často se subjekty pojmenovávají, například Alice (uživatel) a Bob (server), pro náš případ je ale vhodnější použít z anglického Prover (vlastník důvěrné informace - uživatel, klient) a Verifier (ověřovatel - server) označování P a V.

Na obr. 2.1 jsou postupně zobrazeny 3 situace v jeskyni, na jejím konci jsou dveře, které jdou otevřít pouze za pomoci tajného hesla. Princip je následující:

1. P vstoupí do jeskyně a náhodně si zvolí cestu A nebo B, V čeká venku (viz obr. 2.1a).
2. V vstoupí do jeskyně, počká před rozdělením a náhodně určí P cestu, kudy se má vrátit - obr. 2.1b.
3. Nyní mohou nastat 2 situace:
 - a) cesta vstupu P a požadovaná cesta výstupu je shodná - P se vrátí stejnou cestou, aniž by musel použít tajné heslo a tím pádem ho ani nemusí znát
 - b) cesta vstupu P a požadovaná cesta výstupu se liší - P vysloví tajné heslo k otevření dveří a vystoupí požadovanou stranou - obr. 2.2.



Obr. 2.1: Princip nulové znalosti, vlevo a) volba cesty, vpravo b) před dveřmi



Obr. 2.2: Princip nulové znalosti, P zná heslo

Pro funkčnost protokolu je nezbytné jeho mnohonásobné opakování. Pak můžeme dojít k závěru, že pokud P zná tajné heslo, tak vystoupí z jeskyně vždy správnou cestou, kterou V požaduje. Pokud P heslo nezná, tak je jeho úspěšnost pouze 50%, což při n -násobném opakování protokolu dává dostačující malou pravděpodobnost (2^{-n}), že P oklame V.

Výhodou protokolů s nulovou znalostí je možnost dokázat jejich bezpečnost. K tomuto účelu se zavádí tzv. *interaktivní důkazový systém*, který převádí předchozí abstraktní model do matematické roviny a popisuje komunikaci P a V jako komunikaci dvou Turingových strojů vzájemně propojených komunikační páskou. Princip je stejný jako u příkladu s jeskyní, kde P a V spolu komunikují a výsledkem je přijetí/zamítnutí, které vydá V.

Tvrzení: *P a V tvoří interaktivní důkazový systém pro jazyk L s bezpečnostním parametrem k (k je délka vstupního řetězce, C je konstanta), pokud jsou splněny následující podmínky:*

- **Úplnost:** *Pro všechny $x \in L$ platí, že V akceptuje P s pravděpodobností větší než $1 - \frac{1}{C^k}$.*
- **Spolehlivost:** *Pro všechny P' a pro všechny $x \notin L$ platí, že V akceptuje P' s pravděpodobností menší než $\frac{1}{C^k}$.*

Takto nadefinovaný systém ještě nesplňuje podmínku nulové znalosti, proto je nutné ji zde zmínit. Myšlenka této podmínky spočívá v nahrazení uživatele P, který zná tajnou informaci, simulátorem, který ji nezná.

Tvrzení: *Interaktivní důkazový systém pro jazyk L má vlastnost nulové znalosti, pokud pro každého polynomiálního ověřovatele V' existuje polynomiální simulátor S takový, že pravděpodobnostní rozdělení komunikace (S, V') a (P, V) je statisticky stejné. Takový systém se poté nazývá Statistický interaktivní důkazový systém s nulovou znalostí.*

Pozn.: Systém nazýváme též Výpočtový interaktivní důkazový systém s nulovou znalostí, pokud pravděpodobnostní rozdělení komunikace (S, V') a (P, V) je výpočtově nerozeznatelné.

Existence simulátoru, který nezná tajnou informaci, a přesto je schopen úspěšně simulovat protokol nám zaručuje, že danou tajnou informaci ani nemůže uvolnit. Celá problematika důkazů protokolů s nulovou znalostí je mimo rozsah této práce a je

podrobně rozebrána například v [5] nebo [9]. Základní vlastnosti a princip jsou rovněž popsány v [6].

V této části práce jsem popsal základní vlastnosti protokolů s nulovou znalostí. Jsou to úplnost, spolehlivost a existence simulátoru. Většina takovýchto systémů se zabývá nějakými výpočetně složitými problémy. Některé konkrétní příklady těchto protokolů podrobněji popíši v následující části mé práce.

2.2 Fiat-Shamir identifikační protokol

Mezi nejznámější protokol s nulovou znalostí je označován Fiat-Shamir identifikační protokol, publikovaný již v roce 1986 [2]. Ačkoli od jeho uvedení uplynulo již řadu let, stále je v různých publikacích uváděn jako základ pro moderní protokoly a z jeho principů vychází řada modifikací, které byly ještě nedávno používány a některé se používají dodnes. Proto jsem se rozhodl zařadit jej do mé práce, i když slovo „moderní“ se s ním už moc nespojuje.

Účelem protokolu je zajistit, aby se uživatel (prover - P) mohl identifikovat jakémukoliv ověřovateli (verifier – V) a to za splnění podmínky nulové znalosti, kdy tajné heslo zná pouze P a ověřovatel se jej nedozví. Zajištění bezpečnosti vychází z obtížné faktorizace RSA modulu. Průběh protokolu se skládá ze dvou částí: inicializace a vlastní běh protokolu.

Inicializace:

- Někáká důvěryhodná autorita (například centrální bezpečnostní server) vygeneruje k -bitový RSA modul $n = p \cdot q$, kde p, q jsou různá velká prvočísla, která jsou uchována v tajnosti.
- Uživatel P vybere svůj tajný klíč s z množiny Z_n a vypočítá $v = s^2 \bmod n$, kde mod znamená operaci modulo, a následně pošle v autoritě, která zveřejní (n, v) .

Vlastní běh protokolu:

Nyní předpokládejme, že uživatel P se chce identifikovat ověřovateli V. Proto se mezi sebou domluví na počtu průběhů protokolu t (obvykle se uvádí $t = 80$ a větší).

- P zvolí náhodné $r \in Z_n$ (závazek). P dále spočítá $x = r^2 \bmod n$ (důkaz) a pošle x V.
- V zvolí náhodný bit $b \in (0,1)$ (výzva) a pošle ho zpátky P.
- P vypočítá $y = (r \cdot s^b) \bmod n$ (odpověď) a pošle y zpátky ověřovateli.
- V ověří $y^2 \equiv x \cdot v^b \pmod{n}$. Pokud shoda platí, V tento cyklus akceptuje, v opačném případě ho odmítne.

Pokud je protokol akceptován ve všech t -cyklech, je možné považovat protokol za úspěšně ukončený a uživatel P je tak ověřen V. Jak bylo uvedeno v kapitole 2.1, protokol musí splňovat podmínky úplnosti, spolehlivosti a musí existovat simulátor S (rovněž blíže popsáný výše).

Důkaz úplnosti:

Platí, že $x \equiv r^2 \pmod{n}$, $v \equiv s^2 \pmod{n}$, $y \equiv r \cdot s^b \pmod{n}$ z čehož vyplývá, že $y^2 \equiv (r \cdot s^b)^2 \equiv r^2 \cdot (s^2)^b \equiv x \cdot v^b \pmod{n}$, CBD.

Důkaz spolehlivosti:

Předpokládejme, že P úspěšně projde prvním cyklem protokolu. Protože P dopředu neví, jaký bit b bude následovat v dalších cyklech, musí mít připravené obě odpovídající hodnoty y , a to y_0 odpovídající $y_0^2 = x \pmod{n}$ (pro případ $b = 0$) a y_1 odpovídající $y_1^2 = x \cdot v \pmod{n}$. Tedy P musí znát obě hodnoty, nebo musí doufat, že bude „mít štěstí“, což za předpokladu, že V volí b náhodně, dává pravděpodobnost $\frac{1}{2}$. Pokud P zná hodnoty y_0 a y_1 , pak také zná $s = y_1 \cdot \overline{y_0}$ kde $\overline{y_0}$ je inverzní hodnota k y_0 , a hodnota s odpovídá $s^2 = v \pmod{n}$. Takže pokud P uspěje ve všech t -cyklech, musí buďto znát $s^2 = v \pmod{n}$ nebo P měl v každém kole „štěstí“, což se mohlo stát s pravděpodobností $\frac{1}{2^t}$. V případě, že $t=80$ a víc, je tato pravděpodobnost zanedbatelná.

Důkaz nulové znalosti:

Pro důkaz této vlastnosti je nutné popsat, jak můžeme nasimulovat komunikaci mezi P a V. V jednom cyklu protokolu se objevují hodnoty x , b a y s tím, že platí $y^2 \equiv x \cdot v^b \pmod{n}$. K simulování komunikace náhodně vybereme $y \in Z_n$, pak pro $b = 0$ bude $x = y^2 \pmod{n}$, nebo pro $b = 1$ bude $x = (y^2 \cdot \bar{v}) \pmod{n}$, kde $\bar{v}$ je inverzní k $v \pmod{n}$. Nyní již není složité dokázat, že trojice (x, b, y) této simulace bude mít stejné rozložení pravděpodobnosti jako trojice v případě pravého ověřovatele. Z předpokladu, že výpočet odmocniny modulo n je výpočetně složitý a za určitých podmínek neproveditelný, můžeme prohlásit protokol za bezpečný: pro útočníka je prakticky nemožné spočítat s ze znalostí n a v .

Jak jsem již zmínil, protokol Fiat-Shamir se stal základem pro řadu protokolů. Postupně se vyvíjely různá vylepšení řešící jak časovou náročnost, tak nároky na paměť z důvodů použití v paměťových kartách (jednalo se o novou technologii autentizace a byly kladeny nároky na nízkonákladové karty používající „pomalou“ paměť typu EEPROM). Více je možné se o těchto vylepšeních dozvědět v článku Cryptanalysis of a Zero-Knowledge Identification Protocol of Eurocrypt '95 [7]. Z protokolu Fiat-Shamir vychází i Ohta-Okamoto identifikační protokol, který má tu výhodu, že není omezen patenty, což umožnilo jeho využití v projektu OpenSSH [8].

2.3 Ohta-Okamoto identifikační protokol s nulovou znalostí

Inicializace:

- Uživatel P si určí RSA modul n (součin dvou velkých prvočísel), k náhodných celých čísel $S_i \in Z_n$, jako svůj tajný klíč a také malé celé číslo L . Celkově je tedy jeho tajný klíč $(S_1, \dots, S_k, n, L)$.
- P nyní spočítá jeho veřejný klíč $I_i = S_i^L$ pro všechny $1 \leq i \leq k$. Celkově tedy zveřejní $(I_1, \dots, I_k, n, L)$.

Vlastní běh protokolu:

Stejně jako u předchozího protokolu, pro zajištění bezpečnosti je nutné protokol opakovat v t -cyklech.

1. P vybere $R \in Z_n$ a vypočítá $X = R^L$, které pošle ověřovateli V.
2. V vybere náhodně $(e_1, \dots, e_k) \in Z_L$, výběr pošle jako vektor výzvy uživateli P.

3. P vypočítá Y

$$Y = R \cdot \prod_{j=1}^k (S_j)^{e_j} \pmod{n} \quad (2.1)$$

a pošle Y jako svou odpověď.

4. Ověřovatel akceptuje cyklus, pokud je splněna následující rovnost

$$Y^L = X \cdot \prod_{j=1}^k (I_j)^{e_j} \pmod{n}. \quad (2.2)$$

Po t úspěšných cyklech, ověřovatel V akceptuje proces autentizace uživatele P. Pravděpodobnost, že útočník může oklamat důvěřujícího ověřovatele je L^{-kt} , což je dostatečné, jelikož u sekvenčního protokolu může být nejhůře L^{-t} (pro $k = 1$). V již zmiňovaném projektu ZK-SSH je počítáno s možným útokem za polynomiálními hranicemi, proto se požaduje, aby pravděpodobnost možného oklamání ověřovatele byla $\frac{1}{2^{80}}$ a menší.

Pro konkrétní příklad zvolíme $L = 4, k = 10$, tedy 10 tajných klíčů a $t = 4$ cykly protokolu. Vhodnými úpravami dostáváme

$$L^{-kt} = \left(\frac{1}{4}\right)^{10 \cdot 4} = \left(\left(\frac{1}{2}\right)^2\right)^{40} = \left(\frac{1}{2}\right)^{80}.$$

Hodnotu parametru n se doporučuje volit alespoň 2048 bitů, jakožto dostatečnou rezervu vzhledem k rychlosti vývoje v oblastech faktorizace a kryptografie.

3 Password-based protokoly

Proces autentizace obecně vychází z vlastnictví soukromého klíče drženího za účelem ověření uživatele. Takovýto klíč je obvykle vybírán náhodně z vhodné množiny a má délku řádově stovky až tisíce bitů. Praxe ale ukázala, že lidé jsou schopni si zapamatovat heslo délky 7-8 znaků, což při použití všech písmen (malých a velkých) a číslic 0-9 dává pouze 48 bitů náhodnosti. Z toho můžeme usoudit, že pouze takto krátká hesla a z nich odvozené klíče nemohou být dostatečně bezpečné pro použití v kryptografických algoritmech. Jinými slovy, útočník může hrubým útokem v relativně velmi krátké době projít všechny možné kombinace hesel. Blíže se o tvorbě dostatečně bezpečných a dlouhých hesel můžete dočíst v článku „Vypočítejte si kvalitu hesla“ od V. Klímy [10] případně v souvisejících článcích na autorových webových stránkách [11].

Kryptografické klíče jsou většinou uloženy v zabezpečených pamětech počítačů, nebo se používají speciální útoku-odolné servery, případně chytré paměťové karty. To může být ale mnohdy nevhodné nebo příliš drahé. Proto bylo žádoucí přijít s metodou zabezpečení komunikace vycházející ze znalosti krátkých a zapamatovatelných hesel. Tato myšlenka se z důvodů hrubého útoku zdála být nereálná, až do roku 1989, kdy byl představen první password-based protokol. Protokol byl založen na předpokladu, že klient (v aplikacích typu klient-server) znal kromě sdíleného hesla i veřejný klíč serveru. Později byla představena třída protokolů, která ani tento předpoklad nevyžadovala. Jednalo se o tzv. EKE (*Encrypted Key Exchange*) protokoly autorů Bellare a Merrita. Myšlenkou protokolu je, že iniciátor protokolu (klient) vybere krátkodobý veřejný klíč a použije sdílené heslo k zašifrování tohoto klíče. Server dokáže dešifrovat tento veřejný klíč a použít ho k zabezpečení klíče relace, kterou požaduje klient. Za předpokladu, že veřejné klíče jsou náhodné řetězce, útočník při snaze odhalit heslo hrubou silou nebude schopen rozpoznat, který krátkodobý klíč byl použit. A i kdyby našel ten správný klíč, nemohl by ho použít k odhalení klíče relace, protože není možné získat tajný klíč ze znalosti veřejného klíče. Při shrnutí těchto poznatků dostáváme tyto vlastnosti password-based protokolů:

- Klienti vlastní heslo s malou entropií. Jinak řečeno, útočník je schopen odhalit toto heslo hrubou silou v reálném čase.
- Offline slovníkový útok není přípustný. To znamená, že pasivní odposlouchávač, který si zaznamená jednu nebo několik relací, nemůže získat dostatečný počet možných hesel.
- Online slovníkový útok není přípustný. To znamená, že aktivní útočník nemůže zneužít protokol k tomu, aby získal dostatečný počet možných hesel. Může ovšem odstranit nejméně jedno heslo za každý cyklus protokolu pokusem o zamaskování se tímto heslem. V ideálním případě by to mělo být to jediné, co může útočník získat.

Protokol EKE se stal podobně jako protokol Fiat-Shamir základem pro řadu variant odvození, protože originální EKE protokol v sobě nespécifikoval šifrovací algoritmus převodu hesla na relevantní klíč. Celkově lze říct, že password-based protokoly můžeme rozdělit do tří kategorií: v prvním případě se využívá princip Diffie-Helmanova problému (problém diskretního logaritmu), druhá skupina se opírá o protokol RSA a tedy použití faktorizace u modula n . Do třetí kategorie pak patří protokoly, které nepasují do předchozích dvou.

V některých protokolech se používá varianta, ve které server nevlastní přímo sdílené heslo, ale pouze jeho obraz, který získá za použití určené jednocestné funkce. V takovém případě při prolomení bezpečnosti severu nemusí být heslo prozrazeno. Proto taky můžeme password-based protokoly rozdělit podle principu sdílení tajného hesla. V případě, že klient i server sdílejí přímo tajné heslo, používají se metody symetrického šifrování a jde o protokoly typu EKE (např. DH-EKE, A-EKE, B-SPEKE a další [3]). Navrhnout takový protokol je snadnější, ale přináší v sobě nevýhodu při prolomení bezpečnosti serveru a tím přímo získáním tajného hesla. V druhém případě, kdy server uchovává pouze obraz tajného hesla, toto riziko pomíjí, ovšem za cenu složitějšího návrhu a rovněž nutnost použití asymetrických výpočtů. Takové protokoly patří do skupiny AKE (*Asymmetric Key Exchange*) a na rozdíl od EKE protokolů nešifrují žádnou výměnu zpráv při běhu protokolu. Místo šifrování se používají předdefinované matematické vztahy v kombinaci s výměnou krátkodobých náhodných hodnot s použitím tajných klíčů. Nepoužití šifrování je výhodné například z těchto důvodů:

- Zjednodušení protokolu vyhnutím se potřeby domluvit se na šifrovacím algoritmu, jinými slovy se protokol v takovém případě stává závislý na jednom konkrétním šifrovacím algoritmu.
- Slabé šifrování má za následek oslabení celého autentizačního protokolu. Při použití vlastního hesla (i když šifrovaného) se protokol stává náchylný na řadu útoků.
- Softwarové a hardwarové použití šifrovacích algoritmů může porušovat některé právní zákony.

Bližší popis obecného AKE protokolu, jeho matematické odvození a generování není v rozsahu této práce a více je možné se dočíst v [12]. Jednou z možných interpretací vycházející z AKE protokolů je SRP (*Secure Remote Password*) protokol navržený Thomasem Wu [13] jako jednodušší, efektivnější a bezpečnější varianta k EKE protokolům.

3.1 SRP protokol

Podle výše zmíněného rozdělení patří tento protokol do skupiny využívající princip protokolu Diffie-Helmana s tím, že server neuchovává kopii hesla, ale pouze jeho obraz. Protože se jedná o autentizační protokol používaný při ověření klienta serverem a získáním klíče k dané relaci, zavedu pro lepší srozumitelnost popisu protokolu označení klienta C (z angl. Client) a serveru S (Server).

Specifikace parametrů:

Všechny výpočty prováděné v protokolu musí náležet do konečného pole $GF(n)$. Jinak řečeno, všechny výpočty budou redukovány operací modulo n . Náhodně generovaný veřejný parametr n musí být dostatečně velké prvočíslo (nejméně 1024 bitů) navíc pro maximální bezpečnost s vlastností $n = 2p + 1$, kde p je rovněž prvočíslo. Druhým známým parametrem je pak g - generátor pole $GF(n)$. Je to primitivní odmocnina (primitivní kořen) modulo n , a tedy platí, že pro každé celé číslo a takové, že $NSD(a, n) = 1$, existuje celé číslo k takové, že $g^k \equiv a \pmod{n}$. Ostatní symboly použité v průběhu protokolu jsou přehledně popsány v tab. 3.1.

Tab. 3.1: Legenda použitých symbolů password-based protokolů

n	Velké prvočíslo (obvykle nejméně 1024 bitů). Všechny výpočty jsou počítány mod n .
P	Uživatelské heslo krátké délky.
g	Generátor.
s	Náhodný řetězec použitý jako uživatelské zasolení.
x	Tajný klíč odvozený z hesla a zasolení.
v	Obraz hesla uložený na serveru.
u	Náhodně volený veřejný parametr.
a, b	Krátkodobé tajné klíče, náhodná celá čísla, vybírána C resp. S.
A, B	Odpovídající veřejné klíče g^a resp. g^b .
S	Sdílené tajemství vypočtené C resp. S.
K	Odvozený klíč k dané relaci.
$H_i()$	Jednocestná hašovací funkce. U funkcí $H_1, H_2, \dots$ se většinou požaduje, aby byly nezávislé a náhodné.

Jak již bylo napsáno, klient a server neuchovávají v paměti stejné heslo. K prokázání hesla P se serverem, klient náhodně vybere zasolení s a vypočítá

$$x = H(s, P) \text{ a } v = g^x.$$

Server uloží hodnoty v a s jako obraz klientova hesla a zasolení. Pro připomenutí, výpočet obrazu v je jako ostatní výpočty redukován modulo n . x je zahazeno z důvodu ekvivalence k prostému textu hesla P . V tab. 3.2 je uveden běh protokolu s popisem jednotlivých kroků pod tabulkou.

Tab. 3.2: Schéma SRP protokolu

	Klient		Server
1.		$\xrightarrow{\text{Uživatelské jméno}}$	(vyhledání s, v)
2.	$x = H(s, P)$	$\xleftarrow{s}$	
3.	$A = g^a$	$\xrightarrow{A}$	
4.		$\xleftarrow{B, u}$	$B = v + g^b$
5.	$S = (B - g^x)^{a+ux}$		$S = (Av^u)^b$
6.	$K = H(S)$		$K = H(S)$
7.	$M_1 = H(A, B, K)$	$\xrightarrow{M_1}$	ověření M_1
8.	ověření M_2	$\xleftarrow{M_2}$	$M_2 = H(A, M_1, K)$

1. Klient pošle Serveru jeho uživatelské (přihlašovací) jméno, (např. klient).
2. Server vyhledá pro dané jméno příslušný obraz hesla v a jeho zasolení s , které pošle klientovi nazpět. Ten si vypočítá svůj tajný klíč x s použitím skutečného hesla P a zasolení s .
3. Klient vygeneruje náhodné číslo a , $1 < a < n$ a vypočítá svůj krátkodobý veřejný klíč $A = g^a$ a pošle ho serveru.

4. Server vygeneruje náhodné číslo b , $1 < b < n$ a vypočítá svůj krátkodobý veřejný klíč $B = v + g^b$ a pošle ho klientovi spolu s náhodně vygenerovaným parametrem u .
5. Klient i server si nyní vypočítají sdílené tajemství $S = g^{ab+bu x}$ s použitím dostupných hodnot. Pokud se klientovo heslo použité v kroku 2 shoduje s originálním heslem použitým při vytváření obrazu v , budou se obě hodnoty S shodovat.
6. S využitím hašovací funkce vypočítají obě strany kryptograficky silný klíč relace.
7. Klient pošle serveru M_1 jako důkaz, že má správný klíč relace. Server si M_1 vypočítá a porovná, zda se shoduje se zasláným M_1 od klienta.
8. Server pošle klientovi M_2 jako důkaz, že má správný klíč relace. Klient si rovněž M_2 vypočítá a porovná, zda se shoduje se zasláným M_2 od serveru.

V krocích 7 a 8 dochází k vzájemnému ověření znalosti správného klíče relace pomocí jednocestné hašovací funkce. Více o těchto funkcích a příkladu použití v kapitole 4.

Poznámky k protokolu:

- Nezbytně nutná přítomnost $v = g^x$ při výpočtu krátkodobého veřejného klíče $B = v + g^b$. Při jeho vynechání a tedy zjednodušení protokolu se otvírají dveře pro potenciální aktivní slovníkový útok, kdy se útočníkovi, který se maskuje za server, podaří přesvědčit klienta a úspěšně provést autentizaci. Klient následně vypočítá sdílené tajemství S , z něho klíč relace K a pošle útočníkovi M_1 jako důkaz, že zná K . Útočník, protože si nemůže M_1 ověřit, pošle klientovi, že heslo je nesprávné. V uvažovaném případě vynechání v má ovšem nyní útočník potřebný dostatek parametrů k pokusům rozluštit heslo slovníkovým útokem.
- Dalším zajímavým parametrem je náhodná veřejná hodnota u , kterou volí server a je součástí při odvozování společného sdíleného tajemství S . Účelem tohoto parametru je zajistit, že klient zná své pravé heslo a ne pouze jeho obraz. V případě, že potenciální útočník zná obraz hesla a nějak se mu podaří získat u , může poslat v první zprávě $A = g^a v^{-u}$ místo původního $A = g^a$. Server nyní vypočítá $S = (A v^u)^b = (g^a v^{-u} v^u)^b = g^{ab}$. Tato hodnota není závislá na žádném dlouhodobém tajném klíči (x) což ale znamená, že útočník může úspěšně dokončit protokol pouze se znalostí v . Z tohoto faktu vyplývají 2 podmínky pro parametr u : a) u nesmí být konstanta (je zřejmé, že rovněž $u \neq 0$) a b) server nesmí zveřejnit hodnotu u dříve než získá A od klienta.
- Původní protokol SRP se skládá ze tří komunikačních cyklů mezi klientem a serverem, tedy celkem ze šesti zpráv. Vhodným uspořádáním přenosů parametrů při dodržení předchozí podmínky lze redukovat počet cyklů na 2 a tedy celkem 4 zprávy. Takový protokol pak nazýváme optimalizovaný.

Jednou z hlavních stinných stránek protokolu SRP je nedostatek formálních důkazů bezpečnosti. Nicméně autor protokolu ukázal, že bezpečnost protokolu vůči pasivním odposlouchávačům je zajištěna ve smyslu, že pokud existuje takové orákulum, které dokáže najít klíč relace pouze se znalostí přenesených zpráv mezi klientem a serverem (tedy A, B a u), dokáže vyřešit Diffie-Hellmanův problém. Výše popsaná verze protokolu je označována jako SRP-3. V roce 2002 vydal Thomas Wu vylepšení

protokolu [14] nesoucí označení SRP-6 resp. 6A, ve kterých se zabývá diskuzí parametrů B a u , optimalizací a implementací protokolu. Protokol vznikl pod hlavičkou projektu The Stanford SRP Authentication Project a je veden jako open source. Na stránkách projektu [13] je možné zdarma stáhnout zdrojové kódy spolu se zabezpečenými verzemi Telnetu a FTP. Poslední verze, která je k dispozici ke stažení, je z poloviny roku 2007 a zahrnuje fixaci chyby Telnet serveru, popsanou v [15]. V navazující části mé práce bych se chtěl blíže zaměřit na bezpečnost a popis upravené verze protokolu (SRP-6a) a s využitím implementace podrobit protokol praktickému ověřování.

3.2 SNAPI protokol

Protokol SNAPI (Secure Network Authentication with Password Identification) byl prezentován v roce 1999 a jeho autorem je Philip MacKenzie, který se podílel na spolupráci u několika dalších podobných projektů a důkazů jejich bezpečnosti [16]. Tento protokol vychází z protokolu OKE (Open Key Exchange) [3], u kterého však byly prokázány nedostatky vedoucí k možnému odhalení hesla. Jde o password-based protokol, který je zástupcem druhé skupiny a opírá se tedy o principy RSA protokolu. Poskytuje jak autentizaci, tak vzájemnou výměnu klíče relace mezi uživatelem/klientem a serverem. Protokol má dokázanou bezpečnost vůči aktivním útočníkům a tedy nejlepším útokem je tedy prosté uhodnutí hesla. V původní verzi má server uloženou přímo kopii hesla a při prolomení jeho bezpečnosti se tedy útočník heslo přímo doví. Proto byla později představena rozšířená verze tohoto protokolu označována jako SNAPI-X (eXtensible), ve které server uchovává pouze obraz hesla a je tak posílena bezpečnost celého protokolu.

3.2.1 Popis kryptografické techniky a nastavení parametrů protokolu

Kvůli důkazům a dodržení bezpečnosti je nutné stanovit určitá konkrétní omezení při volbě použitých hašovacích funkcí a rovněž při náhodném generování parametrů u RSA veřejného klíče.

Veřejný a tajný RSA klíč:

Dvojice veřejný klíč (N, e) a tajný klíč (N, d) musí splňovat:

- $N = P \cdot Q$, kde P, Q jsou velké prvočísla,
- $NSD(e, \phi(N)) = 1$, kde $NSD(a, b)$ znamená největší společný dělitel a, b ,
- $ed \equiv 1 \pmod{\phi(N)}$, kde $\phi(N) = (P - 1)(Q - 1)$.

Parametry bezpečnosti:

V protokolu se objevují dva parametry k, l , kde k značí „hlavní“ bezpečnostní parametr základní hašovací funkce a tajného klíče (řekněme, že $k = 128$ nebo 160 bitů) a $l > k$ je bezpečnostní parametr pro RSA, resp. pro veřejný klíč ($l = 1024$ bitů).

Hašovací funkce:

V protokolu se objevují 4 hašovací funkce. h, h', h'' jsou funkce s krátkým výstupem délky k bitů. H je funkce s dlouhým výstupem $\eta \geq l + k$. Je požadováno, aby všechny funkce byly nezávislé a aby se chovaly jako náhodné orákulum pro zachování důkazů bezpečnosti. Příklady možného výběru funkcí uvádí autor v [16]. Další podmínkou je stanovení množiny $S_N = \{p: p \leq 2^\eta - (2^\eta \bmod N) \text{ a } NSD(p, N) = 1\}$.

RSA omezení:

Protokol SNAPi je parametrizován omezením možných hodnot veřejného klíče. Tyto omezení musí být testovány v průběhu protokolu. Jak bylo napsáno výše, musí platit

$$NSD(e, \phi(N)) = 1, \quad (3.1)$$

čehož můžeme dosáhnout jednou ze 4 podmínek uvedených v [16]. Pro jednoduchost vyberu pouze jednu:

$$N \in [2^{l-2}, 2^l], e \in [2^l, 2^{l+1}], e \text{ je prvočíslo.} \quad (3.2)$$

Autor ještě uvádí doporučení pro výběr e jako malého prvočísla s malým počtem „1“ v binárním zápise ($3 \equiv 11, 17 \equiv 10001, 65537, \dots$), ale není to podmínkou.

Tab. 3.3: Schéma protokolu SNAPi

Server (S)	Klient (C)
Krok 1: Vybere $m \in_R \{0,1\}^k$ a RSA veřejný klíč N, e	
	$\xrightarrow{K, m, (N, e)}$
Krok 2:	Pokud $N \notin [2^{l-2}, 2^l]$, $e \notin [2^l, 2^{l+1}]$ nebo e není prvočíslo, pak ukonči protokol, jinak $\mu \in_R \{0,1\}^k, a \in_R \mathbb{Z}_N^*$ $p = H(N e m \mu C S \pi)$ - Pokud $p \notin S_N$ pak $q = a$ jinak $q = pa^e \bmod N$
	$\xleftarrow{\mu, q}$
Krok 3: Pokud $\mu \notin_R \{0,1\}^k$ nebo $NSD(q, N) \neq 1$, pak ukonči protokol, jinak	
$p = H(N e m \mu C S \pi)$ - pokud $p \notin S_N$ pak ukonči protokol, jinak $a = (q/p)^d \bmod N,$ $r = h(N e m \mu C S q a)$	
	$\xrightarrow{r}$
Krok 4:	Pokud $p \notin S_N$ nebo $r \neq h(N e m \mu C S q a)$ pak ukonči protokol, jinak $t = h'(N e m \mu C S q a)$ $K = h''(N e m \mu C S q a)$
	$\xleftarrow{t}$
Krok 5: Pokud $t \neq h'(N e m \mu C S q a)$ pak ukonči protokol, jinak $K = h''(N e m \mu C S q a)$	

Schéma protokolu SNAPI-X je totožné se schématem protokolu SNAPI až na rozdíl inicializace. Server si totiž uchovává v paměti pouze obraz klientova hesla a jeho zasolení. Princip je následující: klient si vypočítá $x = H'(C, S, \pi, s)$ a pošle serveru obraz $X = g^x$ a zasolení s . Při zahájení protokolu pak server pošle klientovi zasolení a svoje údaje (S) a zvolí RSA parametry (N, e, d). Zbytek protokolu pak probíhá podle schématu v tab. 3.3.

3.2.2 Výhody a nevýhody protokolu

- Účinnost protokolu SNAPI (SNAPI-X) je srovnatelná s protokoly SPEKE, SRP a SSH co do počtu kroků a počtu veřejných kryptografických operací.
- Na rozdíl od zmíněných protokolů existují důkazy bezpečnosti (v modelu náhodného orákula).
- Jako většina password-based protokolů jsou protokoly otevřené k snadné implementaci.
- Původní verze nebyla navržena pro případný útok na bezpečnost serveru. To řeší až rozšíření SNAPI-X, ovšem za cenu horší efektivity.
- Důkazy bezpečnosti spadají pouze pod model náhodného orákula (což je v dané chvíli dostačující), ovšem stále existuje možnost prolomení použitých hašovacích funkcí. Protokol je ovšem navržen obecně a se zachováním uvedených podmínek je možné použít libovolnou kryptograficky silnější hašovací funkci (viz kapitola 4).

4 Hašovací funkce v autentizaci

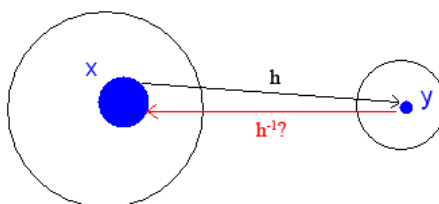
V předchozích kapitolách byly uvedeny rozdílné principy autentizace s příklady protokolů z nich vycházejících. Ačkoli jde o různé protokoly, téměř vždy v sobě obsahují použití hašovací funkce, ať už v obecné podobě, nebo konkrétní s určitými požadavky při výběru (viz např. u protokolu SNAP). V této kapitole se proto blíže zaměřím právě na spojitost hašovacích funkcí s autentizací, rozeberu jejich současné použití a jejich nedostatky v bezpečnosti a v závěru se budu věnovat konceptu návrhu hašovací funkce nové generace představenou předním českým kryptologem Vlastimilem Klímou.

4.1 Vlastnosti hašovací funkce

Hašovací funkce $h: D \rightarrow R$, kde $|D| > |R|$ je funkce, která převádí vstupní posloupnost bitů (bytů) libovolné délky na posloupnost pevné délky n . Výsledná posloupnost se označuje jako *haš* (otisk) a zpravidla nabývá délky 128, 160, 192 nebo 256 bitů. Přitom jsou požadovány tyto vlastnosti funkce: jednosměrnost (synonymum jednocestnost), bezkoliznost a komprese, která už je přímo v definici hašovací funkce.

1. Vlastnost jednosměrnosti

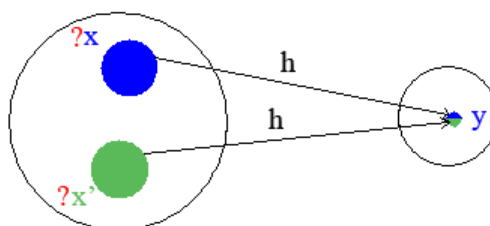
Funkce $f: X \rightarrow Y$ má vlastnost jednosměrnosti, pokud platí, že je snadné z jakékoliv hodnoty $x \in X$ vypočítat $y = f(x)$, ale současně pro libovolné $y = f(x)$ nelze (resp. je to výpočetně nemožné) najít $x \in X$ takové, že $y = f(x)$ (viz obr. 4.1).



Obr. 4.1: Vlastnost jednosměrnosti funkce

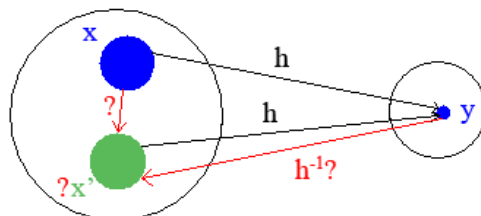
2. Bezkoliznost - rozlišujeme 2 typy bezkoliznost

- a. *bezkoliznost 1. řádu* - jinak označována jako odolnost proti kolizím říká, že je výpočtově neschůdné nalézt jakékoliv dva rozdílné (byť naprosto nesmyslné) vstupy $x \neq x'$, které hašují na shodný výstup; tj. $x \neq x'$ takové, že $h(x) = h(x') = y$ (volný výběr obou vstupů). Tato vlastnost se využívá především k digitálním podpisům, kde se nepodepisuje celá (mnohdy dosti dlouhá) zpráva, ale pouze její haš (viz obr. 4.2).



Obr. 4.2: Bezkoliznost 1. řádu

- b. *bezkoliznost 2. řádu* - je výpočtově neschůdné nalézt jakýkoliv druhý vstup, který má shodný výstup (haš) jako jakýkoliv určený první vstup; tj. pro dané x nelze nalézt druhý argument $x \neq x'$, přičemž platí $h(x) = h(x') = y$. Těto vlastnosti se rovněž říká Odolnost druhého argumentu (viz obr. 4.3).



Obr. 4.3: Bezkoliznost 2. řádu

Ačkoli se může zdát, že obě vlastnosti bezkoliznosti jsou stejné, není tomu tak. Danost zprávy x (prvního argumentu) je totiž při zkoušení odolnosti druhého argumentu určena nějakou, na útočnickovi nezávislou, podmínkou. Z pohledu útočníka je x již dáno a musí pro něj hledat x' . Kdežto při obecné (silné) odolnosti proti kolizím si však útočník může volit x i x' libovolně s jediným cílem, totiž aby hašovaly na shodnou hodnotu (lhostejno jakou). Právě tohoto předpokladu se využívá při realizaci útoků. Je dobré ještě zmínit časté nepochopení bezkoliznosti a to v tom smyslu, že kolizí existuje ohromné množství, což vyplývá z porovnání velikostí množin vstupů ($1 + 2^1 + 2^2 + \dots + 2^D = 2^{D+1} - 1$) a možných výstupů (např. u MD5 jen 2^{128}), ale jejich nalezení je výpočetně nemožné.

Z hlediska bezpečnosti je potřeba nadefinovat pojem *náhodného orákula* jako libovolného stroje, který na nový vstup odpovídá náhodným výběrem výstupu z množiny možných výstupů. Pokud však zadáme zprávu, pro kterou orákulum již někdy hodnotu generovalo, pak sáhne do své paměti a poskytne stejnou hodnotu jako dříve. Ideální hašovací funkce by pak měla mít matematicky shodné vlastnosti, jaké vykazuje náhodné orákulum.

4.2 Bezpečnost hašovacích funkcí

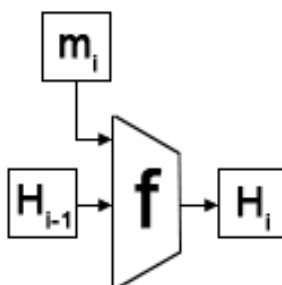
Pokud budeme chápat hašovací funkce jako náhodné orákulum, pak při definování $f: \{0,1\}^D \rightarrow \{0,1\}^n$ bude složitost nalezení vzoru k danému hašovacímu kódu rovna 2^n . Pokud je tedy nalezena cesta, jak vzory nalézat jednodušeji, hovoříme o prolomení hašovací funkce. Tato situace nastává, pokud uvažujeme vlastnost odolnosti proti kolizím. V případě bezkoliznosti 1. řádu nastává tzv. *narozeninový paradox*, který říká, že pro 50% pravděpodobnost toho, že je s vámi ve společnosti někdo, kdo se narodil ve shodný den v roce (léta se neuvažují, tedy 365 dní) jako právě vy, je zapotřebí, aby tato společnost čítala alespoň 254 osob. Pokud se však spokojíte s 50% pravděpodobností toho, že se v nějaké skupině nachází libovolné dvě osoby se shodnými narozeninami, pak stačuje, aby v ní bylo 23 osob. Z narozeninového paradoxu tedy vyplývá, že pro 50% pravděpodobnost nepotřebujeme $1/2$ prvků z množiny ($1/2 \cdot 2^n$), nýbrž pouze odmocninu z celkového počtu prvků ($1/2 \cdot 2^{n/2}$). Pak tedy složitost nalezení vzoru k danému hašovacímu kódu bude rovna $2^{n/2}$.

4.3 Moderní hašovací funkce

U moderních hašovacích funkcí vycházíme z předpokladu, že budeme hašovat velmi dlouhou zprávu. To by bylo velmi náročné, proto je nutné tuto zprávu rozdělit, a to na předem daný počet bloků pevně stanovené délky. Na tuto pevnou délku se rozdělené části zpráv doplňují bitem 1 a poté potřebným množstvím bitů 0, aby bylo zaručeno jednoznačné odejmutí a nedocházelo ke kolizím. V případě Damgard-Merklova principu se přidává na konec každého bloku délka celé vstupní zprávy.

4.3.1 Damgard-Merklovův iterativní princip moderních hašovacích funkcí

Všechny současné moderní funkce využívají princip iterativní hašovací funkce s využitím kompresní funkce, který navrhli Damgard a Merkl nezávisle na sobě. Vstupem kompresní funkce je aktuální blok zprávy m_i a výstupem pak určitá hodnota H_i pevné délky, které se říká *kontext*, který je pak současně vstupem v dalším kroku iterace, viz obr. 4.4. Počáteční hodnota kontextu H_0 se nazývá inicializační hodnota a je dodefinována jako konstanta v popisu každé iterativní hašovací funkce.

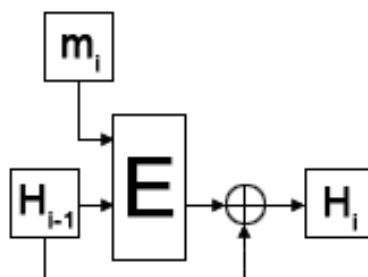


Obr. 4.4: Damgard-Merklova iterativní hašovací funkce

4.3.2 Davies-Meyerova konstrukce kompresní funkce

Vychází z myšlenky využití blokové šifry jako kompresní funkce, jelikož kvalitní bloková šifra by se měla rovněž chovat jako náhodné orákulum a měla by zajistit jednocestnost, která je navíc v případě Davies-Mayerovy konstrukce posílena přičtením vzoru před výstupem (viz obr. 4.5) a je dána vztahem [18]:

$$H_i = f(H_{i-1}, m_i) = E_{m_i}(H_{i-1}) \text{ xor } H_{i-1} \quad (4.1)$$



Obr. 4.5: Davies-Meyerova konstrukce kompresní funkce

4.3.3 Bezpečnost současných hašovacích funkcí

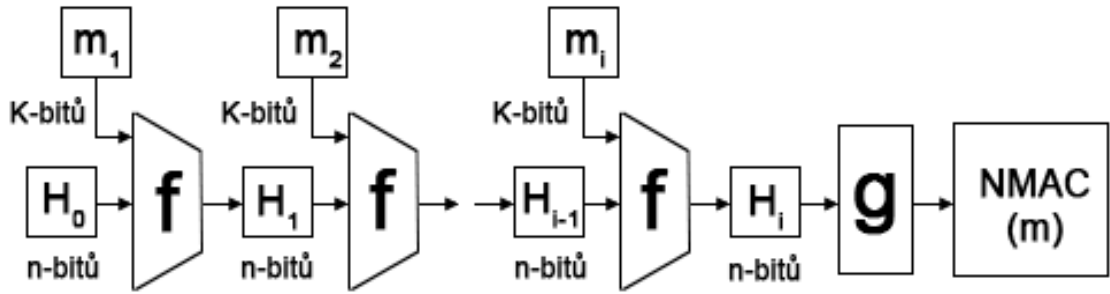
Jak již bylo uvedeno, všechny současné hašovací funkce vycházejí z Damgard-Merklova principu iterativní kompresní funkce. U této metody bylo dokázáno, že pokud je kompresní funkce bezkolizní, je i výsledná hašovací funkce bezkolizní, což požadujeme. Útoky se tedy soustředí na hledání kolizí právě v kompresní funkci a ne celé hašovací funkci. Tato kolize spočívá v nalezení inicializační hodnoty H a dvou různých bloků B_1 a B_2 tak, že $f(H, B_1) = f(H, B_2)$. První dokumentovaný útok na známou hašovací funkci MD5 je znám od roku 1993, šlo však pouze o návrh a jeho realizace byla velmi nákladná (cca 10 miliónů dolarů). S mnohem známějším a efektivnějším útokem přišel čínský tým Wangové představený a prezentovaný na konferenci Crypto 2004 [17]. S ještě rychlejší metodou nalézání kolizí přišel o rok později V. Klíma, která vychází z útoků Wangové, přesto se liší. Autor se ji podrobně věnuje v několika článcích uvedených na svých stránkách věnovaných pouze těmto útokům [19]. Všechny tyto útoky jsou vedeny na bezkoliznost 1. řádu, tedy že si útočník sám vybere dvě vhodné zprávy vedoucí na stejnou haš. Bližší popis těchto útoků je mimo rozsah této práce a je možné se o nich více dočíst v uvedené literatuře. Obecně lze však říci, že nalezením kolizí byla porušena bezpečnost hašovací funkce MD5, stejně jako později publikované útoky na funkce SHA-0 a SHA-1.

4.4 Nový koncept konstrukce hašovacích funkcí

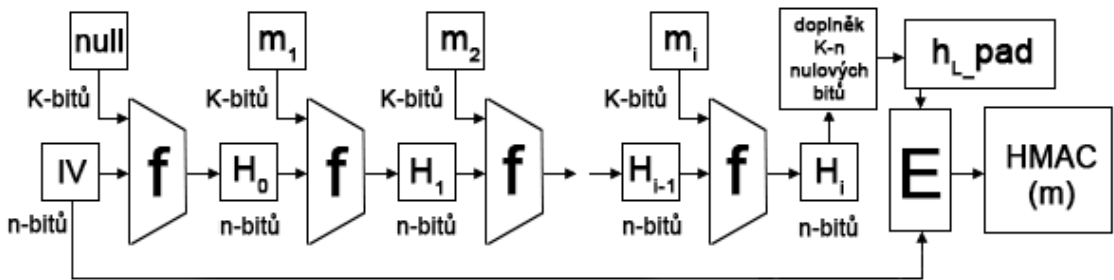
Ačkoli se bloková šifra uvedená v kapitole 4.3.2. jeví jako vhodný konstrukční prvek pro bezpečnou hašovací funkci, je právě ona šifra největší slabinou. Vyplývá to z podstaty původního návrhu a účelu blokové šifry, která se používala na převod otevřeného textu na šifrovaný s pomocí tajného (pro útočníka neznámého) klíče a naopak. V kompresních funkcích však nic utajeného není a útočník zná všechny vstupy, které do funkce vstupují, a co víc, může s nimi dokonce manipulovat nebo si je volit. Což je ale důležitý rozpor v použití blokových šifer. A právě na tomto rozporu je založena většina útoků vedoucí k nalezení kolizí. Proto se celý kryptografický svět zabývá otázkou, čím nahradit klasickou blokovou šifru. S jedním takovým konceptem nového kryptografického primitiva přišel český kryptolog V. Klíma a pojmenoval ho *speciální bloková šifra* (SBŠ).

4.4.1 HMAC a NMAC

Jde o dvě sobě podobné klasické iterativní hašovací funkce, ve kterých je ovšem navíc použita závěrečná operace g , jakožto obrana proti třetímu generickému útoku - útoku prodloužením zprávy. Podrobný popis a definice funkce HMAC je v [20], pro účely této práce stačí obrázek (obr. 4.6 a obr. 4.7), ze kterých jde princip činnosti obou funkcí snadno poznat. H_i jsou kontexty, m_i jednotlivé bloky zprávy a h_{L_pad} je daná konstanta pro HMAC.



Obr. 4.6: Definice hašovací funkce NMAC

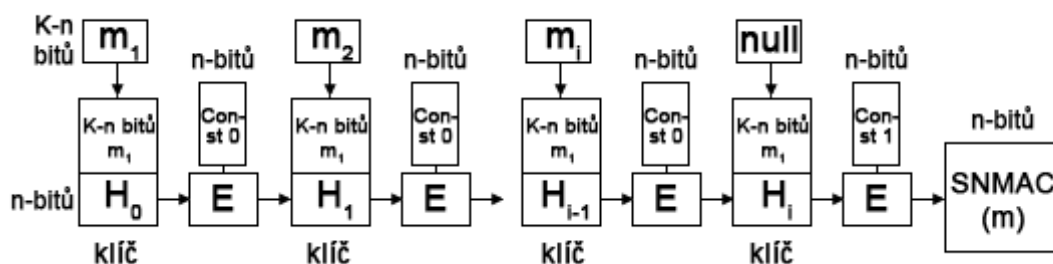


Obr. 4.7: Definice hašovací funkce HMAC

Tyto konstrukce mají dokázané vlastnosti jak v nekonečnu, tak pro konečné rozměry. Proto můžeme říct, že útočník by k nalezení kolize nebo vzoru u konstrukcí NMAC/HMAC potřeboval tolik operací, jako kdyby místo nich byla náhodná orákula. Odolnost proti kolizím a věty o bezpečnosti těchto funkcí jsou uvedeny v [21].

4.4.2 Speciální bloková šifra a SNMAC

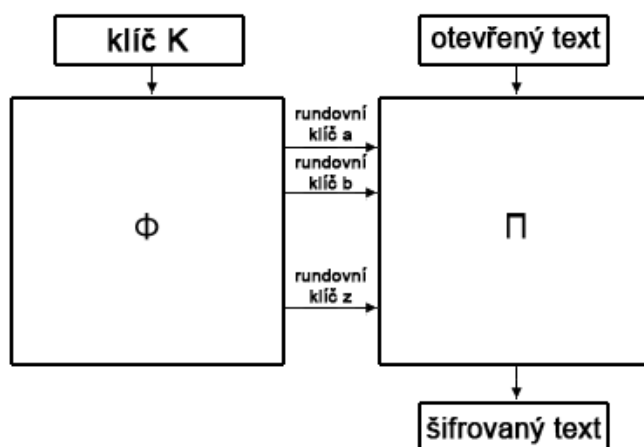
Řekněme, že všechny vstupy kompresní funkce (tj. datový blok m_i a kontext h_{i-1}) převedeme na otevřený text $X = h_{i-1}||m_i$ a klíč k bude konstantní. Pak máme funkci $f: E_{Const_0}(X)$. Tato funkce ovšem nezaručuje jednocestnost. Klasické blokové šifry nám zaručují, že ze znalosti otevřeného a šifrovaného textu nelze určit klíč - a právě toho využijeme. Konstrukci $f: E_X(Const_0)$ nazýváme speciální blokovou šifrou, kde všechny proměnné bity jsou klíčem blokové šifry a ta je použita pouze s konstantním otevřeným textem. Na obr. 4.8 můžeme vidět definici funkce SNMAC [21] vycházející z NMAC a SBŠ s n -bitovým blokem a K -bitovým klíčem. Používá kompresní funkci f a závěrečnou úpravu g . Hodnoty $Const_0$ a $Const_1$ jsou dva různé konstantní otevřené texty a hodnota $Null$ je řetězec $K - n$ nulových bitů.



Obr. 4.8: Definice SNMAC, založená na SBŠ a NMAC

4.5 DN, HDN

Speciální blokové šifry DN a z nich odvozené speciální hašovací funkce HDN vznikly na podnět NBÚ poté, co byly objeveny generické slabiny moderních hašovacích funkcí (popsané v kapitole 4.4). Smyslem těchto projektů nebylo vylepšovat a zesilovat existující hašovací funkce a jejich principy, ale navrhnout (pokud možno) nový princip a funkci, která bude mít vysokou bezpečnost, založenou (pokud možno) na prokazatelných tvrzeních. Z časových důvodů vznikl nejprve koncept speciální blokové funkce a z něho odvozena SNMAC. Později byly tyto poznatky využity při konstrukci šifry DN (z angl. double net - dvojitá síť) jako základ HDN. V této kapitole popíšeme pouze princip konstrukce šifry DN, formální popis včetně důkazů bezpečnosti je uveden v obsáhlém článku [22] a je mimo rozsah této práce.



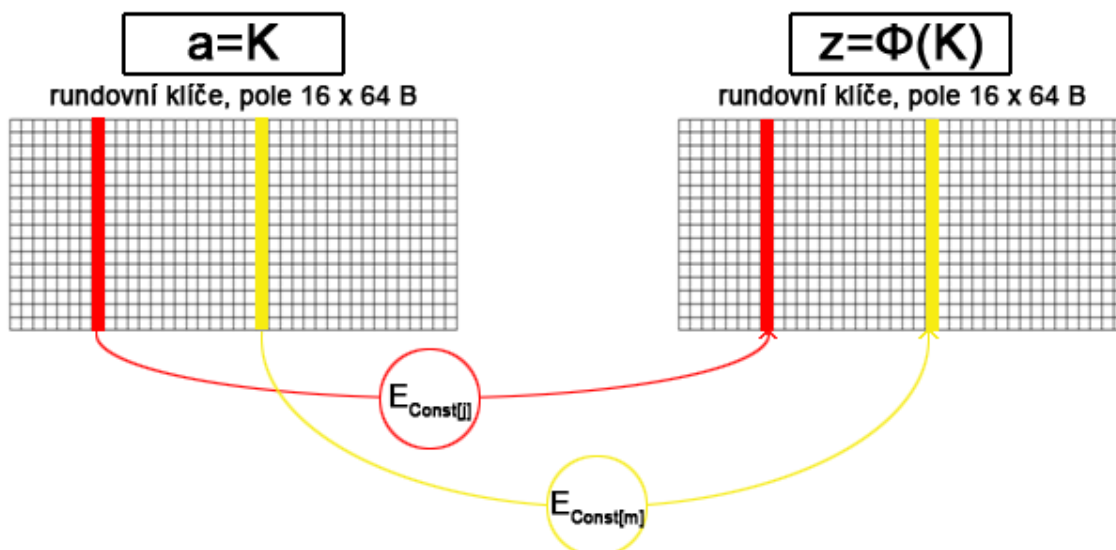
Obr. 4.9: Obecné schéma DN

Z obr. 4.9 je zřejmé, že šifra DN se skládá ze dvou funkcí, expanze klíče Φ a součinnové šifry Π . Oběma těmito funkcím by se přitom měla věnovat stejná pozornost. Současné blokové šifry použité např. v DES, AES fungují tak, že funkce Φ poskytuje 16 proměnných $a, b, \dots, z$, které jsou kopií vybraných bitů klíče K . Funkce Π je součinnová šifra 16-ti rund, v nichž proměnné $a, b, \dots, z$ působí jako rundovní klíče. Nevýhodou je, že jsou tyto proměnné pouze jednoduchou funkcí klíče K (u AES je příprava klíče trochu složitější). Proto od funkce Φ požadujeme, aby jednotlivé rundovní klíče byly nezávislé a pak tedy součin rundovních transformací ve funkci Π byl rovněž nezávislý a

výsledek tvořil kvalitní blokovou šifru. Nezávislost klíčů se řeší na vyšší úrovni, a to skládáním blokových šifer, u nichž skutečně můžeme volit nezávislé klíče.

Funkce Φ

Vstupem funkce Φ je šifrovací klíč K a výstupem je pole rundovních klíčů $a, b, \dots, z$. Šifru Π pak definujeme jako součin blokových šifer B s rundovními klíči $a, \dots, z$, tj. $\Pi = B_z \cdot \dots \cdot B_b \cdot B_a$. Za předpokladu nezávislosti klíčů a a z postačí pro zjednodušení definovat $\Pi = B_z \cdot B_a$, kde $a = K$ a $z = \Phi(K)$ a současně a a z jsou na sobě nezávislé. Hašovací kód u HDN má délku 512 bitů, bloková šifra B má 16 rund. Proměnnou a si můžeme zapsat jako pole o rozměrech 16×64 bajtů a nazveme jej *pole rundovních klíčů*. Řádky se nazývají *malé rundovní klíče* a odpovídající rundy blokové šifry B *malé rundy*. Nejdůležitějším bodem je ale vytvoření klíče z tak, aby byl nezávislý na a . Tady se používá tzv. *sloupcová transformace* s použitím kvalitní blokové šifry E s délkou bloku 16 bajtů. Pak $z = \Phi(K) = (E_{Const[1]}(sloupec_1(K)), E_{Const[2]}(sloupec_2(K)), \dots, E_{Const[64]}(sloupec_{64}(K)))$ je nezávislá na K (viz obr. 4.10).



Obr. 4.10: Sloupcová transformace

Funkce Π

Funkce Π , jak již bylo zmíněno, je definována jako součin blokových šifer B . Ty jsou stejně jako sloupcová transformace konstruovány na bázi vnořených SP sítí. Ty jsou na nejnižší úrovni tvořeny substitučními S-boxy, které převádějí bajt na bajt. Ty jsou stavebními prvky první úrovně SP sítě = 1 XS-box, který se skládá z vrstvy 4 S-boxů, lineárně-difúzní úrovně tvořené maticí typu MDS (maximum distance separable) typu 4×4 bajty, přičtení rundovního klíče a další vrstvy 4 S-boxů. Vzniklý XS-box je stavebním prvkem XXS-boxu, ten XXXS-boxu atd. Vnořováním vytváříme větší a větší SP síť až do potřebného rozměru - šíře 64 bajtů. Použití SP-sítí vede k důkazům odolnosti vůči diferenciální a lineární kryptoanalýze, které jsou podrobně rozebrány v [22].

4.6 Shrnutí

Hašovací funkce jsou silným nástrojem v oboru kryptologie. V praxi mají velmi široké uplatnění a používají se například při digitálním podpisu, pro detekci chyb (kódy CRC apod.) a také v autentizaci. S rostoucími výpočetními možnostmi ale současně roste počet funkcí, u kterých je možné najít kolizi a to buď reálně, nebo pouze teoreticky v brzké budoucnosti. Takové funkce jsou označovány za prolomené a jsou to např. MD4, MD5, SHA-0, RIPEMD a HAVAL-128. Tyto funkce by se neměly používat tam, kde se jedná o nepopiratelnost, tedy např. u digitálního podpisu. U funkce SHA-1 již bylo oznámeno, byť jen teoretické nalezení kolizí za 2^{69} operací a proto se plánuje rovnou přechod na některou z SHA-2, které se liší podle délky hašovacího kódu (SHA-224, 256, 384, 512) a dosud jsou považovány za bezpečné. Jelikož současně používané iterativní funkce nesplňují bezpečnostní podmínky, uvažuje se nad novým konceptem hašovacích funkcí. Jedním z takových konceptů je hašovací funkce HDN vycházející ze speciální blokové šifry SNMAC, jejíž základní princip jsem uvedl v této kapitole.

5 ZK-SSH - metoda s nulovou znalostí pro OpenSSH

Pro praktické ověření principu a fungování autentizační metody s nulovou znalostí jsem se rozhodl použít implementaci vytvořenou v rámci projektu ZK-SSH [8]. Autoři zde představují novou možnost autentizace během procesu vytvoření SSH spojení. Je jí metoda s nulovou znalostí, jejímž základem je upravený protokol Fiat-Shamir, konkrétně pak protokol Ohta-Okamoto, který z něj vychází. Teoretický rozbor tohoto protokolu je v kapitole 2.3. Jako verze SSH byla zvolena OpenSSH.

V úvodu této kapitoly popíši současné možnosti OpenSSH se zaměřením na autentizaci, uvedu funkční postup správného nastavení klienta a serveru pro podporu autentizační metody s nulovou znalostí, kterou podrobím teoretickému rozboru a v závěru rozeberu a popíši data přenášená během procesu vytvoření SSH spojení.

5.1 OpenSSH

OpenSSH je soubor programů určený k bezpečnému způsobu přihlášení ke vzdálenému systému prostřednictvím protokolu SSH (Secure Shell) v počítačových sítích, které používají TCP/IP. Programy bez podpory zabezpečení jako je *telnet*, *rlogin* a *ftp* nahrazují programy *ssh* a *scp*. Součástí je také *sshd* - SSH server démon, *ssh-keygen* - nástroj vytvářející RSA a DSA klíče, které se používají pro ověřování uživatele a další.

OpenSSH je open source verze, která není omezena žádnou licencí a jejíž zdrojový kód je zdarma dostupný a je možné ho upravovat. Má mnoho vlastností, o kterých je možné se dočíst na stránkách projektu [26]. Z hlediska této práce jsem se zaměřil pouze na možnosti autentizace. Mezi nejpoužívanější metody patří:

- „publickey“ - metoda přihlášení s použitím veřejného klíče, obvykle pomocí DSA nebo RSA klíče. Podpora této metody je vyžadována a všechny implementace jí musí podporovat.
- „password“ - metoda přímou výměnou hesla. Nemusí být podporována.
- „hostbased“ - založena na ověření podpisu klienta serverem (klient podepisuje svým tajným klíčem a server ověřuje veřejným klíčem klienta).
- „keyboard-interactive“ - univerzální metoda, při které klient reaguje na výzvy zasílané serverem.
- „GSSAPI“ - metody s podporou externích mechanismů jako je např. Kerberos.

Mezi další metody patří taky „none“, tedy možnost spojení bez autentizace, ale ta se silně nedoporučuje používat. Podporu použití konkrétní metody je možné nastavit v konfiguračních souborech klienta (*ssh_config*) a serveru (*sshd_config*).

5.2 Instalace OpenSSH s autentizací metodou nulové znalosti

Pro analýzu implementace Ohta-Okamoto autentizačního protokolu s nulovou znalostí jsem zvolil operační systém linux, konkrétně pak distribuci Debian a to v aktuální stabilní verzi 5.0 [28]. Pro účely testování jsem se rozhodl využít možnosti virtuálního operačního systému, který lze spouštět například pomocí zdarma dostupného přehrávače VMware Player [29].

Na webových stránkách projektu ZK-SSH [8] jsou k dispozici ke stažení zdrojové kódy OpenSSH ve verzi 4.0 a rovněž *patch* soubor, v kterém jsou uloženy doplňující kódy rozšiřující standardní autentizační metody o metodu nulové znalosti. Autoři projektu uvádějí, že jejich implementace není testována na novějších verzích OpenSSH a současně jde pouze o koncept, a proto by neměla být používána v komerčních produktech. V následující části práce popíši v několika krocích postup pro správnou instalaci a nastavení této implementace pod uvedeným operačním systémem. Jen doplním, že celý postup je prováděn v příkazovém interpretu zadáváním uvedených příkazů.

1. Stažení a rozbalení souborů ze stránek projektu jako uživatel *root*.

```
wget ftp://ftp.openbsd.org/pub/OpenBSD/OpenSSH/portable/openssh-4.0p1.tar.gz
wget http://zk-ssh.cms.ac/patches/zk-ssh-4.0p1-r76.patch
tar xzvf openssh-4.0p1.tar.gz
```

2. Pro konfiguraci zdrojových kódů je potřeba stáhnout vhodné knihovny (zlib1g-dev - komprimační knihovna, libssl-dev - kryptografická knihovna (libcrypto a libssl), build-essential - obsahuje nástroj *make*).

```
apt-get install zlib1g-dev libssl-dev build-essential
```

3. Aplikování patch souboru na zdrojové kódy OpenSSH.

```
patch -p0 < zk-ssh-4.0p1-r76.patch
```

4. Vytvoření libovolného uživatele (např. *siki*) a jeho domovského adresáře a hesla, vytvoření adresáře *zkssh* v domovském adresáři.

```
useradd siki -md /home/siki -s /bin/bash
passwd siki
mkdir /home/siki/zkssh
```

5. Kompilace OpenSSH 4.0 za použití prefixu, čímž zůstane neporušena stávající verze SSH (příkazy se spouštějí stále s právy uživatele *root* v adresáři */root/openssh-4.0p1*).

```
./configure --prefix=/home/siki/zkssh
make
```

make install

6. Generování klíčů a po výzvě jejich uložení do adresáře */home/siki/.ssh*.

`mkdir /home/siki/.ssh` - vytvoření adresáře *.ssh* („.“ znamená, že je skrytý)
`/root/zkssh/bin/ssh-keygen -t oo-zk` - generování klíčů typu (-t) „oo-zk“

7. Přidání veřejného klíče do souboru autorizovaných klíčů.

`cat ~/.ssh/id_oo_zk.pub >> ~/.ssh/authorized_keys`

8. Úprava konfiguračních souborů klienta a serveru tak, aby podporovaly autentizační metodu s nulovou znalostí. Zde uvedu pouze hlavní změny, oba upravené konfigurační soubory včetně okomentování změn jsou součástí elektronické přílohy.

Klient - soubor */home/siki/zkssh/etc/ssh_conf*

Port 2222 - nastaví port, ke kterému se má klient připojit na straně serveru (defaultně 22)

`IdentityFile ~/.ssh/id_oo_zk` - soubor, kde je uložena identita uživatele
`ZeroKnowledgeAuthentication yes` - povolení autentizační metody s nulovou znalostí

Server - soubor */home/siki/zkssh/etc/sshd_conf*

Port 2222 - na tomto portu bude server poslouchat

`UsePrivilegeSeparation no` - vypnutí privilege separation - podmínka autorů pro správnou funkčnost

`ZeroKnowledgeAuthentication yes` - povolení autentizační metody s nulovou znalostí

9. Jako *root* nastavit práva pro domovský adresář uživateli a jeho skupině

`chown siki:siki /home/siki -R`

10. Následující dva příkazy slouží pro spuštění ssh serveru a přihlášení klienta z druhého terminálu. Parametr *-ddd*, resp. *-vvv* slouží pro výpis zpráv v okně terminálu z celého průběhu komunikace a slouží například pro kontrolu v případě, že se klientovi nepodaří navázat spojení. Podrobněji jsou tyto „debug“ zprávy popsány v kapitole 5.4.

`/home/siki/zkssh/sbin/sshd -ddd`
`/home/siki/zkssh/bin/ssh -vvv localhost`

5.3 Analýza implementace metody s nulovou znalostí

V kapitole 5.1 byly uvedeny některé autentizační metody, které jsou implementovány v OpenSSH. Systémy na principu veřejného klíče (RSA, DSA) poskytují přiměřenou úroveň bezpečnosti z pohledu útoku během přenosu informace a možností potencionálního útočníka vydávat se za klienta nebo server. Nicméně jsou tyto metody typu výzva-odpověď oslabeny z principu náležitosti do oblasti polynomické časové roviny, čímž usnadňují třetí straně (útočníkovi) vydávat se za někoho jiného. Právě tento problém řeší metoda s nulovou znalostí, jejíž implementací se budu zabývat v této kapitole.

Komunikace mezi klientem a serverem probíhá na základě výměny zpráv definovaných ve tvaru SSH_MSG_SERVICE_TYP_ZPRAVY. V případě autentizace jsou to konkrétně zprávy, popsané v tab. 5.1. Jak je uvedeno v kapitole 2.3, komunikace protokolu Ohta-Okamoto probíhá v cyklech, kde jedním cyklem jsou myšleny 4 kroky (důkaz, výzva, odpověď, akceptování/zamítnutí cyklu). V projektu ZK-SSH se ale rozhodli z důvodu přehlednosti komunikace rozdělit tento cyklus napůl a jedním cyklem se rozumí dvojice výzva-odpověď. Pokud je tedy počet cyklů podle protokolu Ohta-Okamoto roven $t=4$, bude v komunikaci podle ZK-SSH cyklů 8. Z pohledu zdrojového kódu jsou pro autentizaci klíčové tyto soubory: na straně serveru - *sshd.c* (hlavní soubor SSH) a *auth2.c* (číslo 2 znamená, že jde o verzi SSH2), na straně klienta pak soubory *ssh.c* a *sshconnect2.c*. V těchto souborech je definována komunikace klienta se serverem s odkazem na soubory se zdrojovými kódy jednotlivých autentizačních metod. Pro podporu autentizace s nulovou znalostí jde pak konkrétně o soubor *auth2-zk.c*, funkce protokolu Ohta-Okamoto jsou definovány v souboru *ohta-okamoto.h* a implementovány v *ohta-okamoto.c*. Za zmínku stojí ještě hlavičkový soubor *openssl/bn.h*, který definuje funkce kryptografické knihovny OpenSSL, díky které je možné provádět aritmetické operace s mnohabitovými čísly. Kompletní seznam a popis funkcí je možné najít v dokumentaci ke knihovně na webových stránkách projektu OpenSSL [32].

Tab. 5.1: Typy autentizačních zpráv

Název (kdo posílá: K-klient, S-server)	Význam
SSH2_MSG_USERAUTH_REQUEST(K)	požadavek na autentizaci
SSH2_MSG_USERAUTH_SUCCESS(S)	autentizace úspěšná, není potřeba další metody
SSH2_MSG_USERAUTH_FAILURE(S)	zamítnutí autentizace
SSH2_MSG_USERAUTH_ZK_OK(S)	akceptování metody ZK, server posílá výzvu (challenge)
SSH2_MSG_USERAUTH_ZK_NEXT_ROUND(S)	pokud je počet cyklů menší než 7, může začít další

Jak již bylo uvedeno, komunikace probíhá v cyklech. Ty se při úspěšném pokusu opakují, výjimkou je však zahájení autentizace a její ukončení.

1. Cyklus (ROUND 0)

Zahájení autentizace je definováno ve funkci `int userauth_zk(Authctxt *authctxt)`. V této funkci se nejprve do proměnné `authctxt` (typu struktury) nakopírují hodnoty získané z privátních a veřejných klíčů, dále pak typ autentizace, jméno serveru a další. Následuje výpočet hodnoty důkazu (X - witness). Serveru se odešle paket jako zpráva typu `SSH2_MSG_USERAUTH_REQUEST`, který obsahuje spolu s důkazem hodnoty exponentu (L), modulu (n), hodnoty veřejných klíčů ($I_1, \dots, I_k$) a jejich počet (k).

```
debug("((ZK)) ROUND 0");

/* P: Issue a witness */
OO_witness(R, X, &private);
debug2("((ZK)) Sending witness to server");

/* assemble userauth request for zk */
packet_start(SSH2_MSG_USERAUTH_REQUEST);
packet_put_cstring(authctxt->server_user);    /* user name in UTF-8 encoding */
packet_put_cstring(authctxt->service);         /* „userauth-request“ */
packet_put_cstring(authctxt->method->name);    /* „zk@cms.ac“ */
packet_put_int(0);                             /* set round counter to 0 */
packet_put_bignum2(X);                         /* add witness */

/* add public key */
packet_put_bignum2(public.exp);                /* exponent (L) */
packet_put_bignum2(public.n);                 /* modulus */
packet_put_int(public.num);                   /* number of identities k */
for (i=0; i<public.num; ++i)
    packet_put_bignum2(public.l[i]); /* identity */
packet_send();
```

Výpočet hodnoty důkazu ($X = R^L \bmod n$) probíhá na začátku každého (sudého) cyklu pomocí funkce `OO_witness(R, X, &private)` implementované v souboru *ohta-okamoto.c*.

```
/* X = náhodné bity délky n */
if (!BN_rand(X, BN_num_bits(private->n), -1, 0))
    goto err;
/* R = X mod n */
if (!BN_mod(R, X, private->n, ctx))
    goto err;
/* X = R^L mod n */
if (!BN_mod_exp(X, R, private->exp, private->n, ctx))
    goto err;
```

Na straně serveru je v souboru *auth2-zk.c* definována funkce `static int userauth_zk(Authctxt *authctxt)`. Na základě vyhodnocení podmínky „číslo cyklu modulo 2“, tedy pokud je číslo cyklu sudé, server nakopíruje získané hodnoty do své proměnné `authctxt`, vygeneruje výzvu (*challenge*) a tu pak v paketu pošle zpátky klientovi jako zprávu `SSH2_MSG_USERAUTH_ZK_OK`.

```

/* V: Issue a challenge */
challenge = OO_challenge(&public);

debug2("((ZK)) Sending challenge for user %s", authctxt->user);

/* send challenge to client */
packet_start(SSH2_MSG_USERAUTH_ZK_OK);
packet_put_int(round);
for (i=0; i<public.num; ++i)
    packet_put_bignum2(challenge[i]);
packet_send();

```

Generování výzvy probíhá ve funkci `challenge = OO_challenge(&public)`. Výsledkem je pole náhodně vygenerovaných čísel typu `BIGNUM`. Jejich počet (proměnná `public->num`) je definovaný v hlavičkovém souboru *ohta-okamoto.h*. Implicitně je nastavená hodnota 10.

```

for (i=0; i < public->num; i++) {
    challenge[i] = BN_new();
    if (challenge[i] == NULL)
        goto err;
    if(!BN_rand(challenge[i], bits, -1, 0))
        goto err;
}

```

2. Cyklus (ROUND 1)

Klient po obdržení zprávy `SSH2_MSG_USERAUTH_ZK_OK` volá funkci `void input_userauth_zk_ok(int type, u_int32_t seq, void *ctxt)`, ve které se vypočítá odpověď (*response*) na výzvu, kterou poslal server a ta se odešle jako zpráva `SSH2_MSG_USERAUTH_REQUEST`. Číslo cyklu v tomto případě odlišuje tuto zprávu od prvního cyklu, kde se zpráva jmenovala stejně.

```

/* P: Respond to the challenge w.r.t. the witness) */
response = OO_response(challenge, R, &private);

debug2("((ZK)) sending response");

/* send responses to server's challenges */
packet_start(SSH2_MSG_USERAUTH_REQUEST);
packet_put_cstring(authctxt->server_user);
packet_put_cstring(authctxt->service);
packet_put_cstring(authctxt->method->name);
packet_put_int(round+1);
packet_put_bignum2(response);
packet_send();

```

Výpočet odpovědi (*response*) má na starost funkce `OO_response(challenge, R, &private)` implementována opět v souboru *ohta-okamoto.c*. Tento výpočet složený z mnohonásobného součinu mocnin (podle rovnice 3.1) je složitý, proto se zde používá

tzv. *Mongomeryho multiplikační metoda*, která je použita ve funkcích knihovny OpenSSL. Detailní rozbor metody je mimo rozsah této práce, více informací o algoritmu výpočtu je možné najít v [34]. Výpočet probíhá samostatně ve funkci `_OO_multiply_up`, na kterou se výše napsaná funkce odkazuje.

```
/* cyklus výpočtu odpovědi, len=10, t1,t2 jsou pomocné proměnné */
for (i=0; i < len; i++) {
    if (!BN_is_zero(exp[i])) {
        if(!BN_mod_exp(t1, base[i], exp[i], n, ctx))
            goto err;
        if(!BN_to_montgomery(t2, t1, mctx, ctx))
            goto err;
        if(!BN_mod_mul_montgomery(t1, res, t2, mctx, ctx))
            goto err;
        if(BN_copy(res, t1) == NULL)
            goto err;
    }
}
return t1;
```

Server vyhodnotí podmínku „číslo cyklu modulo 2“. V tomto případě jde o lichý cyklus, a proto server ověří odpověď a v případě, že se nejedná o poslední pokus (číslo cyklu < 7) a ověření proběhlo v pořádku, zašle nazpět zprávu `SSH2_MSG_USERAUTH_ZK_NEXT_ROUND`. V případě posledního cyklu a kladného ověření je autentizace přijata za úspěšnou, dojde k uvolnění paměti potřebné pro uložení veřejných klíčů a klientu se odešle zpráva `SSH2_MSG_USERAUTH_SUCCESS`. Může pokračovat další fáze vytvoření SSH spojení. Při neúspěchu se odešle zpráva `SSH2_MSG_USERAUTH_FAILURE` a informace o dalších možnostech autentizace.

```
if (OO_verify_response(challenge, response, X, &public))
{
    debug("((ZK)) Round %d: Success", round);
    if (round < 7){
        debug("((ZK)) Starting next round");
        packet_start(SSH2_MSG_USERAUTH_ZK_NEXT_ROUND);
        packet_put_int(round);
        packet_send();
        packet_write_wait();
        authctxt->postponed = 1;}
    else{
        debug("((ZK)) Authentication successfull, logging in...");
        authenticated = 1;
        authctxt->postponed = 0;

        /* cleanup, free memory holding public key */
        free(public.l);
        BN_free(public.exp);
        BN_free(public.n);}
}
```

```

if (authenticated == 1)
{
    packet_start(SSH2_MSG_USERAUTH_SUCCESS);
    packet_send();
    packet_write_wait();
    authctxt->success ==1;}
else{
    methods = authmethods_get();
    packet_start(SSH2_MSG_USERAUTH_FAILURE);
    packet_put_cstring(methods);
    packet_send();
    packet_write_wait();
}

```

Ověření (*verify*) probíhá pomocí funkce `OO_verify_response(challenge, response, X, &public)` a to tak, že se vypočítají a porovnají hodnoty na levé a pravé straně rovnosti. Na levé straně rovnosti je výpočet ze získané klientovy odpovědi jednoduchý. Na pravé straně musí server vypočítat složitou rovnici 3.2, k čemuž se opět použije Mongomeryho metoda.

```

/* levá strana, left = (response^exp) mod n, tedy left = Y^L mod n */
BN_mod_exp(left, response, public->exp, public->n, ctx);
/* pravá strana se odkazuje na funkci _OO_multiply_up */
right = _OO_multiply_up(public->l, challenge, public->num, X, public->n);
/* vyhodnocení rovnosti*/
if (!BN_cmp(left, right))
    return 1;
else
    return 0;

```

Další cykly probíhají obdobně s jediným rozdílem, a to že v lichých cyklech posílá klient serveru s identifikací pouze nově vypočtenou hodnotu svědka spolu s číslem cyklu, což je definováno ve funkci `void input_userauth_zk_next_round(int type, u_int32_t seq, void *ctxt)`.

```

/* P: Issue a witness */
OO_witness(R, X, &private);
debug2("((ZK)) Sending witness to server");

/* assemble userauth request for zk */
packet_start(SSH2_MSG_USERAUTH_REQUEST);
packet_put_cstring(authctxt->server_user);
packet_put_cstring(authctxt->service);
packet_put_cstring(authctxt->method->name);
packet_put_int(round+1);          /* set round counter */
packet_put_bignum2(X);           /* add witness */
packet_send();

```

5.4 Rozbor komunikace klienta se serverem

V této kapitole se zaměřím na rozbor komunikace klienta se serverem na základě výpisu tzv. *debug* zpráv zachycených jako výstup v terminále klienta a serveru. Je důležité zmínit, že komunikaci začíná a dále pak řídí klient, server pouze naslouchá a reaguje na klientovy výzvy. Celý proces SSH spojení lze rozdělit do několika částí:

1. Inicializace spojení
2. Výměna a domluva možných bezpečnostních technik
3. Výměna klíčů
4. Autentizace uživatele
5. Sestavení spojení

Rozbor celé komunikace není předmětem této práce, proto ke každé části uvedu pouze pár zpráv a zaměřím se hlavně na část autentizace. Zprávy začínající textem ((ZK)) se vztahují k autentizaci s nulovou znalostí. Kompletní výpis zpráv je pak možné najít na přiloženém CD.

Ad. 1) během inicializace si klient se serverem vymění informace například o použité verzi SSH, jakou verzi protokolu používá klient a vlastnímu navázání spojení.

```
Klient:  OpenSSH_4.0p1, OpenSSL 0.9.8g 19 Oct 2007
Server:  debug1: Client protocol version 2.0; client software version OpenSSH_4.0
Klient:  debug1: Connecting to localhost [127.0.0.1] port 2222.
          debug1: Connection established.
```

Ad. 2) v další části dochází k výměně znalostí algoritmů pro šifrování, zajištění integrity dat a případné kompresi. Výsledkem je pak dohoda, které algoritmy se použijí.

```
Klient:  debug1: kex: client->server aes128-cbc hmac-md5 none
Server:  debug1: kex: server->client aes128-cbc hmac-md5 none
```

Ad. 3) proces výměny klíčů je proveden pomocí metody Diffie-Hellman (generování nového DH klíče, domluvení počtu bitů a následné výměně s potvrzeným ukončením). Od konce této fáze již začíná šifrovaná komunikace.
Poznámka:

- „kex“ je zkratkou pro key exchange, tedy výměnu klíčů.

```
Klient:  debug2: dh_gen_key: priv key bits set: 124/256
Server:  debug2: bits set: 520/1024
Klient:  debug2: set_newkeys: mode 0
Server:  debug1: KEX done
```

Ad. 4) před vlastní autentizací uživatele dochází nejdříve k autentizaci serveru, a to kontrolou podpisu otisku jeho klíče (tzv. *fingerprint*) v souboru `~/.ssh/known_hosts`.

```
debug3: check_host_in_hostfile: filename /home/siki/.ssh/known_hosts
debug3: check_host_in_hostfile: match line 1
```

Klient: debug1: Host 'localhost' is known and matches the RSA host key.
debug1: Found key in /home/siki/.ssh/known_hosts:1
debug1: ssh_rsa_verify: signature correct

Proces autentizace se skládá z 5 kroků:

- Žádost klienta o autentizaci.

```
Klient: debug1: SSH2_MSG_SERVICE_REQUEST sent
Server: debug1: userauth-request for user siki service ssh-connection method
        zk@cms.ac
Klient: debug2: service_accept: ssh-userauth
        debug1: SSH2_MSG_SERVICE_ACCEPT received
```

- Výběr autentizační metody.

```
debug1: Authentications that can continue: publickey,password,keyboard-
        interactive,zk@cms.ac
Klient: debug3: preferred zk@cms.ac,publickey,keyboard-interactive
        debug3: authmethod_lookup zk@cms.ac
        debug3: authmethod_is_enabled zk@cms.ac
```

- Inicializace veřejného a soukromého klíče klienta.

```
debug3: ((ZK)) userauth_zk: Identity file 1: /home/siki/.ssh/id_oo_zk
debug1: ((ZK)) userauth_zk: Trying to load private key
Klient: debug3: ((ZK)) userauth_zk: Try 0 for key 1
        debug3: ((ZK)) userauth_zk: Reading key 1
        debug1: ((ZK)) userauth_zk: Trying private key: /home/siki/.ssh/id_oo_zk
        debug2: ((ZK)) key_load_private_oo_zk: Finished loading public key
```

- Následuje 8 cyklů podle protokolu Ohta-Okamoto. První cyklus je specifický, neboť v něm kromě výměny definovaných hodnot dochází na straně serveru k ověřování uživatelova veřejného klíče (uloženého v souboru `~/.ssh/authorized_keys`).

Round 0:

Klient - posílá serveru důkaz (*witness*).

```
Klient: debug1: ((ZK)) ROUND 0
        debug2: ((ZK)) Sending witness to server
        debug2: we sent a zk@cms.ac packet, wait for reply
```

Server - ověřuje veřejný klíč klienta, přijme jeho důkaz a pošle nazpět výzvu (*challenge*).

```
debug2: ((ZK)) Round 0
debug1: ((ZK)) Received public key for user siki:
debug1: ((ZK)) trying public key file /home/siki/.ssh/authorized_keys
debug1: ((ZK)) user_key_allowed2: matching key found: file
/home/siki/.ssh/authorized_keys, line 1
Server: ((ZK)) user_key_allowed2: Found matching OO-ZK key:
12:f6:cc:ce:ed:15:dd:09:8f:f9:74:20:a3:34:46:52
debug1: ((ZK)) Received key for user siki is trusted
debug2: ((ZK)) Received witness
debug2: ((ZK)) Sending challenge for user siki
```

Round1:

Klient - přijme výzvu od serveru, vypočítá a pošle nazpět odpověď (*response*).

```
debug1: ((ZK)) ROUND 1
Klient: debug2: ((ZK)) Read challenge (from server); answering challenge
debug2: ((ZK)) sending response
```

Server - přijme klientovu odpověď, kterou ověří a vyhodnotí. V kladném případě může začít další cyklus.

```
debug2: ((ZK)) Round 1
Server: debug2: ((ZK)) Response received
debug1: ((ZK)) Round 1: Success
debug1: ((ZK)) Starting next round
```

- Další cykly probíhají analogicky až do cyklu 7, kde v případě úspěšného ověření na straně serveru dojde k autentizaci uživatele.

```
Server: debug1: ((ZK)) Round 7: Success
Server: debug1: ((ZK)) Authentication successfull, logging in...
Accepted zk@cms.ac for siki from ::ffff:127.0.0.1 port 41932 ssh2
Klient: debug1: Authentication succeeded (zk@cms.ac).
```

Ad. 5) Poslední fáze se týká finálního sestavení spojení.

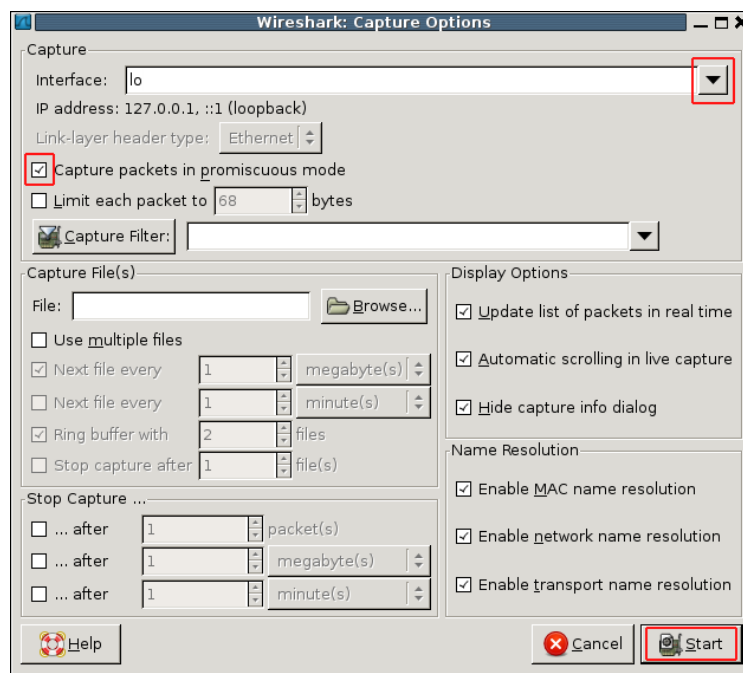
```
Klient: debug1: channel 0: new [client-session]
debug3: ssh_session2_open: channel_new: 0
```

```
Server: debug1: channel 0: new [server-session]
debug1: session_new: init
```

5.5 Rozbor přenesených dat

V této kapitole se zaměřím na rozbor dat, která si vyměňují klient a server během procesu vytvoření SSH spojení se zaměřením na část autentizace. K zachycení komunikace jsem se rozhodl použít program Wireshark [30]. Jde o jeden z nejpoužívanějších síťových paketových analyzátorů na světě, navíc je šířen pod GPL licenci, takže je možné jej používat zdarma legálně. Jeho hlavní funkcí je zachytávání komunikace na určitém síťovém zařízení (rozhraní) a tu pak dekodovat podle RFC norem a zobrazovat ve zvoleném formátu. Jeho využití je například při řešení potíží na síti, testování bezpečnosti komunikace nebo při hledání chyb v implementacích protokolů.

Nastavení zachytávání komunikace provedeme v okně Capture Options (viz obr. 5.1), dostupné z nabídky menu Capture → Options nebo kliknutím na druhou ikonu zleva pod nabídkou menu. Jelikož jsou klient i server spouštěni současně na téže počítači, vybereme v seznamu Interface možnost *lo*, což nám určuje rozhraní místní smyčky, tzv. *loopback*, čemuž odpovídá i IP adresa 127.0.0.1 (ta se doplní automaticky). Zbývá povolit možnost zachytávání v tzv. *promiskuitním módu* (Capture packet in promiscuous mode), při kterém program zachytává všechny pakety, které projdou skrz dané síťové zařízení. Tlačítkem *Start* spustíme zachytávání paketů.



Obr. 5.1: Nastavení zachytávání paketů v programu Wireshark

Během celého procesu vytvoření SSH spojení program zachytil celkem 49 paketů. Výpis všech zachycených paketů je mimo rozsah této práce a je možné jej najít v elektronické podobě na přiloženém CD. Všechny pakety mají stejnou zdrojovou a cílovou IP adresu (dáno zvoleným rozhraním místní smyčky) a jde o TCP pakety.

Klientovi byl přidělen port 52489 a server komunikuje (naslouchá) na portu 2222. První tři pakety se vztahují k navázání TCP spojení, nenesou tedy žádná data a mají postupně nastaveny příznaky ACK, SYN+ACK, ACK (obr. 5.2).

No. .	Source	Destination	Protocol	Data(B)	Info
1	127.0.0.1	127.0.0.1	TCP	0	52489 > EtherNet/IP-1 [SYN] Seq=0 Win=32792 Len=0 MSS=16396 TSV=347052 TSER=0 WS=6
2	127.0.0.1	127.0.0.1	TCP	0	EtherNet/IP-1 > 52489 [SYN, ACK] Seq=0 Ack=1 Win=32768 Len=0 MSS=16396 TSV=347052
3	127.0.0.1	127.0.0.1	TCP	0	52489 > EtherNet/IP-1 [ACK] Seq=1 Ack=1 Win=32832 Len=0 TSV=347052 TSER=347052
4	127.0.0.1	127.0.0.1	TCP	20	EtherNet/IP-1 > 52489 [PSH, ACK] Seq=1 Ack=1 Win=32768 Len=20 TSV=347053 TSER=347053
5	127.0.0.1	127.0.0.1	TCP	0	52489 > EtherNet/IP-1 [ACK] Seq=1 Ack=21 Win=32832 Len=0 TSV=347054 TSER=347053

Obr. 5.2: Navázání TCP spojení

V paketech č. 8 a 9 dochází k vzájemné výměně podporovaných bezpečnostních technik. Jak jde vidět na obr. 5.3, přenos ještě není šifrovaný a je tam možno rozpoznat přenášená data.

No. .	Source	Destination	Protocol	Data(B)	Info
8	127.0.0.1	127.0.0.1	TCP	640	EtherNet/IP-1 > 52489 [PSH, ACK] Seq=21 Ack=21 Win=32768 Len=640 TSV=347054 TSER=347054
9	127.0.0.1	127.0.0.1	TCP	640	52489 > EtherNet/IP-1 [PSH, ACK] Seq=21 Ack=661 Win=34112 Len=640 TSV=347054 TSER=347054
10	127.0.0.1	127.0.0.1	TCP	0	EtherNet/IP-1 > 52489 [ACK] Seq=661 Ack=661 Win=34048 Len=0 TSV=347064 TSER=347054
11	127.0.0.1	127.0.0.1	TCP	24	52489 > EtherNet/IP-1 [PSH, ACK] Seq=661 Ack=661 Win=34112 Len=24 TSV=347064 TSER=347064
12	127.0.0.1	127.0.0.1	TCP	0	EtherNet/IP-1 > 52489 [ACK] Seq=661 Ack=685 Win=34048 Len=0 TSV=347064 TSER=347064

0000	00 00 00 00 00 00 00 00	00 00 00 00 08 00 45 00	E.
0010	02 b4 8f 52 40 00 40 06	aa ef 7f 00 00 01 7f 00	...R@.@.
0020	00 01 08 ae cd 09 4c 17	c5 10 4c 2f 07 c4 80 18	L..L/...
0030	02 00 00 a9 00 00 01 01	08 0a 00 05 4b ae 00 05	K...
0040	4b ae 00 00 02 7c 0b 14	2c 0b ab ad ae 0d 89 06	K....].
0050	9b 50 1e 4f 8d 7d fd 0b	00 00 00 59 64 69 66 66	.P.O.}...Ydiff
0060	69 65 2d 68 65 6c 6c 6d	61 6e 2d 67 72 6f 75 70	ie-hellm an-group
0070	2d 65 78 63 68 61 6e 67	65 2d 73 68 61 31 2c 64	-exchang e-shal,d
0080	69 66 66 69 65 2d 68 65	6c 6c 6d 61 6e 2d 67 72	iffie-he llman-gr
0090	6f 75 70 31 34 2d 73 68	61 31 2c 64 69 66 66 69	oup14-sh al,diffi
00a0	65 2d 68 65 6c 6c 6d 61	6e 2d 67 72 6f 75 70 31	e-hellma n-group1
00b0	2d 73 68 61 31 00 00 00	0f 73 73 68 2d 72 73 61	-shal... ssh-rsa
00c0	2c 73 73 68 2d 64 73 73	00 00 00 87 61 65 73 31	,ssh-dss aes1
00d0	32 38 2d 63 62 63 2c 33	64 65 73 2d 63 62 63 2c	28-cbc,3 des-cbc,
00e0	62 6c 6f 77 66 69 73 68	2d 63 62 63 2c 63 61 73	blowfish -cbc,cas
00f0	74 31 32 38 2d 63 62 63	2c 61 72 63 66 6f 75 72	tl28-cbc ,arcfour
0100	2c 61 65 73 31 39 32 2d	63 62 63 2c 61 65 73 32	,aes192- cbc,aes2
0110	35 36 2d 63 62 63 2c 72	69 6a 6e 64 61 65 6c 2d	56-cbc,r ijndael-
0120	63 62 63 40 6c 79 73 61	74 6f 72 2e 6c 69 75 2e	cbc@lysa tor.liu.
0130	73 65 2c 61 65 73 31 32	38 2d 63 74 72 2c 61 65	se,aes12 8-ctr,ae
0140	73 31 39 32 2d 63 74 72	2c 61 65 73 32 35 36 2d	sl92-ctr ,aes256-
0150	63 74 72 00 00 00 87 61	65 73 31 32 38 2d 63 62	ctr....a es128-cb
0160	63 2c 33 64 65 73 2d 63	62 63 2c 62 6c 6f 77 66	c-2des-cbc blowf

Obr. 5.3: Nešifrovaná data v paketech č. 8 a 9

Přenos se stává šifrovaným až po domluvě konkrétní metody šifrování a výměně klíčů protokolem Diffie-Helman. Proto je podle obsahu dat prakticky nemožné určit význam následujících paketů. Nicméně ze znalosti principu protokolu s nulovou znalostí (probíhá v cyklech) a zachycených paketech s periodicky se opakující velikostí dat lze téměř s jistotou říci, které pakety se týkají procesu autentizace a jakou informaci by měly obsahovat. Autentizační metoda podle protokolu Ohta-Okamoto začíná paketem č. 23 a končí paketem č. 38 (obr. 5.4).

No. .	Source	Destination	Protocol	Data(B)	Info
20	127.0.0.1	127.0.0.1	TCP	48	EtherNet/IP-1 > 52489 [PSH, ACK] Seq=1277 Ack=893 Win=35328 Len=48 TSV=347078
21	127.0.0.1	127.0.0.1	TCP	64	52489 > Ethernet/IP-1 [PSH, ACK] Seq=893 Ack=1325 Win=36672 Len=64 TSV=347078
22	127.0.0.1	127.0.0.1	TCP	80	EtherNet/IP-1 > 52489 [PSH, ACK] Seq=1325 Ack=957 Win=35328 Len=80 TSV=347078
23	127.0.0.1	127.0.0.1	TCP	1680	52489 > Ethernet/IP-1 [PSH, ACK] Seq=957 Ack=1405 Win=36672 Len=1680 TSV=347078
24	127.0.0.1	127.0.0.1	TCP	80	EtherNet/IP-1 > 52489 [PSH, ACK] Seq=1405 Ack=2637 Win=49280 Len=80 TSV=34707
25	127.0.0.1	127.0.0.1	TCP	208	52489 > Ethernet/IP-1 [PSH, ACK] Seq=2637 Ack=1485 Win=36672 Len=208 TSV=3470
26	127.0.0.1	127.0.0.1	TCP	32	EtherNet/IP-1 > 52489 [PSH, ACK] Seq=1485 Ack=2845 Win=49280 Len=32 TSV=34707
27	127.0.0.1	127.0.0.1	TCP	208	52489 > Ethernet/IP-1 [PSH, ACK] Seq=2845 Ack=1517 Win=36672 Len=208 TSV=3470
28	127.0.0.1	127.0.0.1	TCP	80	EtherNet/IP-1 > 52489 [PSH, ACK] Seq=1517 Ack=3053 Win=49280 Len=80 TSV=34707
29	127.0.0.1	127.0.0.1	TCP	208	52489 > Ethernet/IP-1 [PSH, ACK] Seq=3053 Ack=1597 Win=36672 Len=208 TSV=3470
30	127.0.0.1	127.0.0.1	TCP	32	EtherNet/IP-1 > 52489 [PSH, ACK] Seq=1597 Ack=3261 Win=49280 Len=32 TSV=34707
31	127.0.0.1	127.0.0.1	TCP	208	52489 > Ethernet/IP-1 [PSH, ACK] Seq=3261 Ack=1629 Win=36672 Len=208 TSV=3470
32	127.0.0.1	127.0.0.1	TCP	80	EtherNet/IP-1 > 52489 [PSH, ACK] Seq=1629 Ack=3469 Win=49280 Len=80 TSV=34707
33	127.0.0.1	127.0.0.1	TCP	208	52489 > Ethernet/IP-1 [PSH, ACK] Seq=3469 Ack=1709 Win=36672 Len=208 TSV=3470
34	127.0.0.1	127.0.0.1	TCP	32	EtherNet/IP-1 > 52489 [PSH, ACK] Seq=1709 Ack=3677 Win=49280 Len=32 TSV=34707
35	127.0.0.1	127.0.0.1	TCP	208	52489 > Ethernet/IP-1 [PSH, ACK] Seq=3677 Ack=1741 Win=36672 Len=208 TSV=3470
36	127.0.0.1	127.0.0.1	TCP	80	EtherNet/IP-1 > 52489 [PSH, ACK] Seq=1741 Ack=3885 Win=49280 Len=80 TSV=34708
37	127.0.0.1	127.0.0.1	TCP	208	52489 > Ethernet/IP-1 [PSH, ACK] Seq=3885 Ack=1821 Win=36672 Len=208 TSV=3470
38	127.0.0.1	127.0.0.1	TCP	32	EtherNet/IP-1 > 52489 [PSH, ACK] Seq=1821 Ack=4093 Win=49280 Len=32 TSV=34708
39	127.0.0.1	127.0.0.1	TCP	64	52489 > Ethernet/IP-1 [PSH, ACK] Seq=4093 Ack=1853 Win=36672 Len=64 TSV=34708
40	127.0.0.1	127.0.0.1	TCP	48	EtherNet/IP-1 > 52489 [PSH, ACK] Seq=1853 Ack=4157 Win=49280 Len=48 TSV=34708

Obr. 5.4: Zachycené pakety procesu autentizace

Jde tedy celkem o 16 paketů, což odpovídá 4 cyklům podle protokolu Ohta-Okamoto, resp. 8 cyklům podle projektu ZK-SSH. V tab. 5.2 jsou uvedeny všechny autentizační pakety a jejich teoretický význam.

Tab. 5.2: Pakety procesu autentizace a jejich teoretický význam

Číslo paketu	23	24	25	26	27	28	29	30
Data v paketech	1680	80	208	32	208	80	208	32
Teoret.význam	W0	CH0	R0	VR0	W1	CH1	R1	VR1
ZK-SSH	ROUND 0		ROUND 1		ROUND 2		ROUND 3	
Ohta-Okamoto	ROUND 0				ROUND 1			
Číslo paketu	31	32	33	34	35	36	37	38
Data v paketech	208	80	208	32	208	80	208	32
Teoret.význam	W2	CH2	R2	VR2	W3	CH3	R3	VR3
ZK-SSH	ROUND 4		ROUND 5		ROUND 6		ROUND 7	
Ohta-Okamoto	ROUND 2				ROUND 3			
Wx - witness, CHx - challenge, Rx - response, VRx - verify response, ROUND x - cyklus, x - pořadí cyklu								

V tabulce si můžeme všimnout, že s výjimkou paketu č. 23 se velikosti datové části paketů opakují. To ale přesně odpovídá teoretickému rozboru dat v kapitole 5.3, kde se v prvním paketu přenášejí spolu s důkazem (X - witness) a identifikačními údaji serveru navíc hodnoty modulu, exponentu a veřejných klíčů.

6 Autentizační metoda založená na protokolu SRP

Password-based protokol Secure Remote Password popsaný v kapitole 3.1 tvoří základ projektu The Stanford SRP Authentication Project [13]. Zabývá se použitím hesla jako bezpečné metody autentizace a integruje ji do nových a existujících síťových aplikací. Hlavním cílem je pak zdokonalení této metody a zajištění její větší bezpečnosti použitím protokolu SRP, který nabízí unikátní spojení bezpečnosti pomocí hesla, je uživatelsky přívětivý a není omezen licencemi.

Na stránkách projektu jsou k dispozici zdarma ke stažení jak zdrojové kódy implementace, tak zkompilevané programy pro různé operační systémy a to díky tomu, že je projekt veden jako open source. Samotný projekt se zaměřuje na bezpečné verze programů telnet a FTP. Na stránkách je však odkaz na další projekty používající protokol SRP, například v kombinaci s protokolem TLS nebo jako jednu z autentizačních metod v programu OpenSSH. K dispozici je rovněž dokumentace k projektu a protokolu SRP a nově navíc odkaz na demo protokolu, vytvořené v JavaScriptu, kde si může uživatel přímo v prohlížeči vyzkoušet, jak protokol funguje a jak se mění vypočtené hodnoty po zadání vlastního hesla.

V této kapitole se zaměřím na popis instalace programů telnet a FTP s podporou protokolu SRP vytvořených ve zmiňovaném projektu, rozeberu implementaci autentizační části a porovnáám se zachycenými daty při praktickém testování těchto programů.

6.1 Instalace programů FTP a telnet s podporou SRP autentizace

Existuje několik možností, jak postupovat při instalaci a podporované jsou tři operační systémy. Je možné použít buď zdrojové kódy a provést postup kompilace sám (doporučuji pouze zkušeným uživatelům, jelikož je během procesu kompilace nutné vhodně nastavit několik doplňujících knihoven a celý postup je časově náročný), nebo použít již hotové kompilace. Já jsem se rozhodl pro druhou možnost a jako operační systém jsem zvolil Linux v distribuci Debian verze 4.0.

Na stránkách projektu jsou k dispozici ke stažení dva druhy instalačních balíčků: „SRP“ balíček obsahuje pouze zkompilevané programy pro klienta (telnet, ftp) a server (telnetd a ftpd). Je určen především pro klienta, který si s programy pro něj určenými vystačí. Balíček „EPS“ (Exponential Password Suite) obsahuje zkompilevané programy *login*, *su*, *passwd* a *tcopnf* a je určen ke konfiguraci systému tak, aby podporoval autentizační mechanismy protokolu SRP. Správce systému tedy potřebuje oba balíčky. Tento balíček je navíc ve dvou verzích - s podporou PAM (viz níže) a bez podpory PAM.

Postup instalace:

Před vlastní instalací zjistíme, zda systém podporuje tzv. PAM (Pluggable Authentication Module). Ten umožňuje systémovému administrátorovi přidávat, měnit nebo mazat autentizací mechanismy různých služeb a to pouze úpravou konfiguračního souboru a instalací tzv. „PAM modulů“. Současně není potřeba měnit stávající programy *passwd* a *su*. Pokud systém obsahuje soubor */etc/pam.conf* je pravděpodobné, že PAM podporuje a je možné provést instalaci podle postupu A. V opačném případě je třeba postupovat podle B.

Poznámka:

- pokud nebude uvedeno jinak, instalaci provádím jako *root*.

Postup A (s podporou PAM):

1. Stažení a rozbalení balíčků

```
wget ftp://srp.stanford.edu/pub/srp/binaries/1.5.1/eps-i386-linux6-pam.tar.gz
wget ftp://srp.stanford.edu/pub/srp/binaries/1.5.1/srp-i386-linux6.tar.gz
tar xzvf eps-i386-linux6-pam.tar.gz
tar xzvf srp-i386-linux6.tar.gz
```

2. Nakopírování PAM modulů do adresáře */lib/security*

```
cp ~/eps-i386-linux6-pam/pam_eps_auth.so /lib/security
cp ~/eps-i386-linux6-pam/pam_eps_passwd.so /lib/security
```

3. Přejít na EPS systém:

- v souboru */etc/pam.conf* vyhledat všechny výskyty podobné těmto:

```
DEFAULT auth required /lib/security/pam_unix_auth.so
DEFAULT password required /lib/security/pam_unix_passwd.so
```

- ve všech takových případech je potřeba přidat (ne nahradit!) nový řádek s odkazem na nové moduly

```
DEFAULT auth required /lib/security/pam_eps_auth.so
DEFAULT password required /lib/security/pam_eps_passwd.so
```

- tímto zajistíme, že se po spuštění programu *passwd* změní jak heslo ve starém UNIX formátu, tak EPS heslo

Poznámka:

- je možné, že soubor */etc/pam.conf* odkazuje na adresář */etc/pam.d*. V tomto případě je potřeba provést výše uvedené změny v souborech */etc/pam.d/auth* a */etc/pam.d/passwd*

4. Nakopírování programu *tconf* do */sbin* a změna práv

```
cp ~/eps-i386-linux6-pam/tconf /sbin
```

chmod 744 /sbin/tconf

5. Spuštění programu *tconf*. Tím se vytvoří soubor */etc/tpasswd.conf* - při výzvě: „Generate a (n)ew field or use a (P)redefined field?“ můžeme zvolit buď *P*, pak program nabídne předem vygenerované dvojice hodnot pro Modul (v rozsahu 256 - 2048 bitů) a Generátor (prvočísla 2,5,7). Konkrétní dvojici je možné vybrat volbou čísla 1-11. Při volbě *n* umožní program vygenerovat si vlastní hodnoty Modulu - možné velikosti (v bitech): 257 (minimální, pouze na testování), 384, 512, 768, 1024 (PGP úroveň bezpečnosti), 1536 a 2048 (maximální podporovaná úroveň bezpečnosti).

Poznámka:

- generování 1024 bitů dlouhého prvočísla zabralo zhruba 30s, vygenerování prvočísla délky 2048 bitů se mi ani po 5 minutách nepodařilo, což může být dáno mj. použitím virtuálního obrazu OS a tím značně snížené výpočetní výkonnosti.

6. Každý uživatel systému si nyní musí vygenerovat nové heslo (pomocí programu *passwd*), čímž se aktualizuje soubor */etc/tpasswd.conf*. Časová náročnost této operace se odvíjí od počtu uživatelů systému, ale je nutná pro přechod k dalšímu kroku.

7. Instalování upravených verzí programů *ftp* (*ftpd*) a *telnet* (*telnetd*) - instalací je myšleno nahrazení stávající verze zmíněných programů jejich upravenou verzí distribuovanou v balíčku SRP. Standardní umístění klientských programů je v adresáři */usr/bin* a umístění serverových programů (tzv. démonů) je možné zjistit např. příkazem:

```
grep ftp /etc/inetd.conf
```

Celá cesta k adresáři je v řádku výpisu na 6. místě. Je doporučené původní programy zálohovat např. přidáním přípony „*orig*“.

```
mv /usr/bin/ftp /usr/bin/ftp.orig  
cp ~/srp-i386-linux6/ftp /usr/bin/ftp
```

8. Povolení SRP autentizace jako jediné metody ověření uživatele - implicitně se programy při neúspěšné SRP autentizaci přepnou na klasickou (nezabezpečenou) autentizaci heslem. Přidáním parametru „-a valid“ na konec řádku záznamu o programu *telnet* v souboru *inetd.conf*, resp. „-a“ u programu *ftp* zajistíme podporu pouze SRP autentizace.

```
telnet stream tcp nowait root /usr/sbin/in.telnetd in.telnetd -a valid  
ftp stream tcp nowait root /usr/sbin/in.ftpd in.ftpd -a
```

Poznámka:

- po této změně je nutné restartovat *inetd* démona.

```
# /etc/init.d/openbsd-inetd restart
```

9. Dokončení přechodu na EPS systém:

- v souboru */etc/pam.conf* smažeme všechny záznamy odkazující na moduly *pam_unix_auth.so* a *pam_unix_passwd.so*. Jsou to ty, ke kterým jsme ve 3. kroku přidávali nové řádky (zůstanou pouze tyto).

10. Smazání původních záznamů o heslech

- v rámci optimálního zabezpečení je vhodné vymazat, případně přepsat (přidáním koncovky EPS) soubory */etc/passwd* a */etc/shadow*.

11. Nepovinný, avšak doporučený krok

- před odhlášením a následným přihlášením do systému je vhodné otestovat funkčnost nových hesel např. programem *ftp* (příkazem: *# ftp localhost*) nebo *telnet*.

Postup B (bez podpory PAM):

1. Stažení a rozbalení balíčků

```
# wget ftp://srp.stanford.edu/pub/srp/binaries/1.4/eps-i386-linux5.tar.gz
# wget ftp://srp.stanford.edu/pub/srp/binaries/1.5.1/srp-i386-linux6.tar.gz
# tar xzvf eps-i386-linux5.tar.gz
# tar xzvf srp-i386-linux6.tar.gz
```

2. Zálohování současných a nakopírování nových programů *passwd*, *login* a *su* např. takto:

```
# mv /bin/passwd /bin/passwd.orig
# cp ~/eps-i386-linux5/passwd /bin
# chmod 4711 /bin/passwd
```

- stejný postup opakovat s ostatními programy

3. Ostatní kroky jsou shodné s postupem A a to od 4 kroku.

6.2 Analýza implementace SRP autentizace

Balíček SRP-2.1.2, který je možné stáhnout z webových stránek projektu SRP autentizace [13], obsahuje zdrojové kódy implementace autentizační metody uložené v adresáři *libsrp*. Spolu s nimi obsahuje balíček zdrojové kódy k programům *telnet* a *ftp*, ale rovněž jejich zkompileované verze, které jsou v adresáři *inst*. Součástí je také knihovna *libkrypto*, která obsahuje definice vybraných proudových a blokových šifer a

jednocestné hašovací funkce. Samozřejmě nemůže chybět adresář s řádnou dokumentací. V této kapitole rozeberu část zdrojový kódů implementace týkající se autentizace, která bude sloužit k porovnání s reálným průběhem.

Před vlastním rozбором kódů je potřeba objasnit, jak byl projekt vyvíjen a jakým způsobem je možné ovlivnit průběh a realizaci metody. Základem správného běhu metody je kryptografická knihovna, která umožní zpracovávat aritmetické operace s velkými čísly definované v SRP protokolu. Podporovány jsou následující knihovny: OpenSSL, GNU MP, LibTomCrypt, LibTomMath, MPI a CryptoLib, které se liší nabízenými funkcemi a taky rychlostí výpočtů operací. Při kompilaci ze zdrojových kódů je možné si některou zvolit, doporučena je knihovna OpenSSL.

Stěžejním souborem metody SRP je *srp.h*, kde jsou definovány hlavní funkce společné pro klienta i server. V závislosti na verzi protokolu SRP se pak dále odkazují na soubory *srp6_client.c* a *srp6_server.c* pro verzi SRP-6/6a nebo na *rfc2945_client.c* a *rfc2945_server.c* pro verzi SRP-3. Protokol SRP-6/6a má následující vylepšení:

1. Při výpočtu klíče relace na straně klienta je hodnota g^x vynásobena třemi SRP-6a: hodnota g^x je vynásobena hašem z H a g
2. Hodnota u je brána jako haš z A a B namísto prvních 32 bitů haše z B

Jelikož jsou funkce a proměnné mezi sebou provázány odkazy a uvedením určité funkce ze souboru *srp.h* by bylo potřeba provést rozbor několika úrovní funkcí až ke kryptografickým knihovnám, kde jsou definovány požadované aritmetické operace, uvedu v následující části pouze názorný příklad ustanovení autentizace a bezpečného spojení mezi klientem a serverem. Zmiňované aritmetické operace se neprovádějí s celými hodnotami, ale pouze s jejich haši. K jejich uložení slouží struktura *SHA1-CTX*, která odkazuje na reálné hodnoty uložené v proměnné *srt*, taktéž typu struktura. V prvku „hash“ struktury *SHA1-CTX* je uložen hodnota haše, která se bude odesílat druhé straně. Prvek „ckhash“ pak uchovává hodnotu haše, kterou očekává od druhé strany. Ve struktuře jsou navíc další 2 prvky, které uchovávají předešlé hodnoty pro zpětnou kompatibilitu. Následující příklad se vztahuje k protokolu SRP-6 a je přehledně popsán v tab. 6.1.

Tab. 6.1: SRP autentizace a komunikace mezi klientem a serverem

Klient	Server
SRP_initialize_library();	SRP_initialize_library();
Inicializace knihoven pro generování náhodných čísel, výpočty aritmetických operací a další funkce.	
SRP * c_srp = SRP_new(SRP6_client_method());	SRP * s_srp = SRP_new(SRP6_server_method());
Klient i server volají funkci <i>SRP_new</i> , definovanou v souboru <i>srp.h</i> , která oběma komunikujícím stranám inicializuje novou strukturu <i>srp</i> , ve které bude uložen autentizační kontext během procesu vytvoření spojení. Prvky struktury jsou např. modul, generátor, zasolení a další.	
	SRP_set_server_lookup(s_srp, SRP_SERVER_system_lookup());
Definování vyhledávací metody. Server potřebuje vědět, jakou metodu má použít pro přiřazení uživatelského jména ke správnému heslu. To záleží na způsobu, jakým procesem a v jakém formátu byly hesla vytvořeny a kde jsou uloženy.	
char * username = prompt_for_username(); send_to_server(username); $\Rightarrow$	char * username = recv_from_client();
Klient posílá serveru svoje uživatelské jméno.	
SRP_set_username(c_srp, username);	SRP_set_username(s_srp, username);
Uložení uživatelského jména do autentizační struktury na obou stranách.	
str N = recv_from_server(); str g = recv_from_server(); str salt = recv_from_server();	$\Leftarrow$ send_to_client(s_srp $\rightarrow$ modulus); $\Leftarrow$ send_to_client(s_srp $\rightarrow$ generator); $\Leftarrow$ send_to_client(s_srp $\rightarrow$ salt);
Server posílá klientovi hodnoty modulu, generátoru a zasolení.	
SRP_set_params(c_srp, N.data, N.len, g.data, g.len, salt.data, salt.len);	
Přiřazení získaných hodnot do struktury.	
char * password = prompt_for_password(); SRP_set_auth_password(c_srp, password);	
Výpočet tajného hesla x ze zasolení (s), uživatelského jména (U) a skutečného hesla (P) pomocí hašovací funkce H . Toto heslo se uloží do struktury a bude použito při výpočtu klíče relace. $x = H(s \parallel H(U \parallel ":" \parallel P))$. Tato funkce je definována i na straně serveru, ale používá se pouze v testovacím režimu.	
cstr * pub = cstr_new(); SRP_gen_pub(c_srp, &pub); send_to_server(pub); $\Rightarrow$	cstr * pub = cstr_new(); SRP_gen_pub(s_srp, &pub); $\Leftarrow$ send_to_client(pub);
Na každé straně se vygenerují náhodné tajné čísla (a, b) a vypočítají veřejné klíče (A, B), které si strany mezi sebou vymění. Klient vypočítá $A = g^a \bmod n$. Server vypočítá $B = kv + g^b \bmod n$, kde pro SRP-6 je $k = 3$.	
recv_from_server(pub); cstr * skey = cstr_new(); SRP_compute_key(c_srp, &skey, pub $\rightarrow$ data, pub $\rightarrow$ length);	recv_from_client(pub); cstr * skey = cstr_new(); SRP_compute_key(s_srp, &skey, pub $\rightarrow$ data, pub $\rightarrow$ length);

Obě strany vypočítají ze získaných veřejných klíčů A, B klíč relace ($skey$). V proměnné $skey$ je uložen haš klíče relace, reálná hodnota je dostupná v $srp \rightarrow key$. Vlastní výpočet klíče relace detailně rozeberu pod touto tabulkou. Klient vypočítá $(B - g^x)^{a+ux} \bmod n$. Server vypočítá $(Av^u) \bmod n$.	
V následujících krocích probíhá ověřování klíče relace a potvrzení, že obě strany mají stejný klíč.	
<pre>cstr * resp = cstr_new(); SRP_respond(c_srp, &resp); send_to_server(resp); =></pre>	<pre>cstr * resp = cstr_new(); read_from_client(resp); if(!SRP_OK(SRP_verify(s_srp, resp->data, resp->len))) abort_with_error(); SRP_respond(s_srp, &resp); <= send_to_client(resp);</pre>
<pre>recv_from_server(resp); if(!SRP_OK(SRP_verify(c_srp, resp->data, resp->len))) abort_with_error();</pre>	
V případě, že je klíč relace úspěšně ověřen na obou stranách, je možné jej začít používat k zabezpečené komunikaci. Na závěr je potřeba uvolnit alokovanou paměť.	
<pre>cstr_free(resp); cstr_free(pub); /* ... */ SRP_free(c_srp);</pre>	<pre>cstr_free(resp); cstr_free(pub); /* ... */ SRP_free(s_srp);</pre>

Výpočet klíče relace na straně klienta:

- je definován v souboru *srp6_klient.c* v rámci funkce *srp6_client_key_ex*. Probíhá postupnými výpočty použitím funkcí z kryptografické knihovny podle vzorce $key = (B - g^x)^{a+ux} \bmod N$. Protože je vzorec výpočtu příliš složitý, jsou zavedeny pomocné proměnné, pomocí nichž se vzorec rozdělí na elementární výpočty, které již jsou definovány v knihovně. Jednotlivé proměnné a mezivýpočty jsou dobře popsány ve zdrojovém kódu.

Výpočet klíče relace na straně serveru:

- je definován v souboru *srp6_server.c* ve funkci *srp6_server_key*. Stejně jako u výpočtu na straně klienta dochází k zavedení pomocných proměnných a postupných výpočtech vedoucí na finální vzorec $key = (A * v^u)^b \bmod n$.

6.3 Rozbor zachycených dat během vytvoření FTP spojení

K zachycení a následnému rozboru komunikace mezi klientem a serverem během procesu vytvoření FTP spojení jsem použil stejný program, jako u autentizační metody s nulovou pravděpodobností - Wireshark. FTP server jsem nainstaloval na virtuálním operačním systému spuštěném pomocí přehrávače VMware Player podle postupu uvedeném v kapitole 6.1. Pro otestování vzájemné kompatibility mezi různými systémy

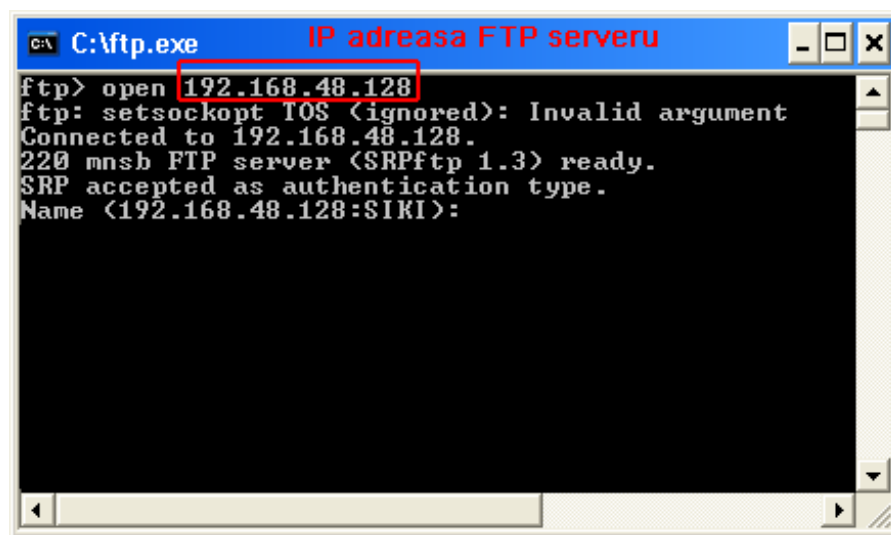
jsem se rozhodl použít FTP klienta pod platformou win32, který je možné stáhnout ze stránek projektu SRP autentizace [13].

Implementace SRP FTP je založena na protokolu „*file transfer protocol*“ (FTP) a je definována v IETF článku pod názvem „FTP Security Extensions“ [31]. Umožňuje kódování a dekódování zpráv se zajištěním bezpečnosti a důvěrnosti pomocí knihovny *libcrypto*. Autentizační část je řízena pomocí příkazů AUTH a ADAT. Po úspěšné autentizaci se používají příkazy MIC (Message Integrity protected Command) pro zajištění důvěrnosti zpráv a ENC (Encoding protected Command) pro kódování. Všechna data jsou kódovány algoritmem *Base64*, který zobrazuje binární data pomocí tisknutelných ASCII znaků.

Nastavení programu Wireshark je shodné jako v kapitole 5.5, pouze je nutné vybrat správné rozhraní (*Interface*) v okně Capture options. V mém případě jde o rozhraní VMware Virtual Ethernet Adapter VMnet8 s IP adresou 192.168.48.1. Stisknutím tlačítka *Start* začne zachytávání paketů. Nyní spustíme FTP klienta, tedy program *ftp.exe*, který se ovládá zadáváním příkazů do příkazové řádky. Zadáním „?“ zobrazíme seznam dostupných příkazů. Pro připojení k FTP serveru potřebujeme znát jeho IP adresu. Tu zjistíme například příkazem *ifconfig* na virtuálním systému. Celý proces vytvoření FTP spojení je možné rozdělit do tří částí:

1. Otevření FTP spojení a připojení k serveru

- klient se připojí k serveru zadáním příkazu: `open <IP>`, kde IP je síťová adresa serveru (viz obr. 6.1). Tato fáze končí výzvou k zadání uživatelského jména klienta.



```
C:\ftp.exe IP adresa FTP serveru
ftp> open 192.168.48.128
ftp: setsockopt TOS (ignored): Invalid argument
Connected to 192.168.48.128.
220 mnsb FTP server (SRPftp 1.3) ready.
SRP accepted as authentication type.
Name (192.168.48.128:SIKI):
```

Obr. 6.1: Připojení klienta k FTP serveru

- na obr. 6.2 jsou data zachycená programem Wireshark, která se vztahují k první části. Mají tento význam (S - server, K - klient):

- paket 1-3: navázání TCP spojení se serverem, pakety mají postupně nastaveny příznaky SYN, SYN+ACK, ACK.
- paket 8: S→K: název serveru a jeho stav (ready = naslouchá).
- paket 9: K→S: typ požadované autentizace.
- paket 11: S→K: potvrzení SRP jako metody autentizace.
- paket 12,13: výměna informace o verzi SRP (4B integer).

No. -	Source	Destination	Protocol	Info
1	192.168.48.1	192.168.48.128	TCP	m-wnn > ftp [SYN] Seq=0 win=65535 Len=0 MSS=1460
2	192.168.48.128	192.168.48.1	TCP	ftp > m-wnn [SYN, ACK] Seq=0 Ack=1 win=5840 Len=0 MSS=1460
3	192.168.48.1	192.168.48.128	TCP	m-wnn > ftp [ACK] Seq=1 Ack=1 win=65535 Len=0
8	192.168.48.128	192.168.48.1	FTP	Response: 220 mnsb FTP server (SRPftp 1.3) ready.
9	192.168.48.1	192.168.48.128	FTP	Request: AUTH SRP
10	192.168.48.128	192.168.48.1	TCP	ftp > m-wnn [ACK] Seq=42 Ack=11 win=5840 Len=0
11	192.168.48.128	192.168.48.1	FTP	Response: 334 Using authentication type SRP; ADAT must follow.
12	192.168.48.1	192.168.48.128	FTP	Request: ADAT AAAABAAAAAE=
13	192.168.48.128	192.168.48.1	FTP	Response: 335 ADAT=AAAABAAAAAE=
14	192.168.48.1	192.168.48.128	TCP	m-wnn > ftp [ACK] Seq=30 Ack=119 win=65417 Len=0

Obr. 6.2: Výpis dat první části vytvoření FTP spojení programu Wireshark

2. Identifikace klienta zadáním jeho uživatelského jména

- jako klient zadám přihlašovací jméno „tester“. Program mě informuje, že probíhá inicializace generátoru náhodných čísel (done = inicializace úspěšná) a vyžádá si heslo. Průběh druhé části FTP klienta je na obr. 6.3.

```

C:\ftp.exe
ftp> open 192.168.48.128
ftp: setsockopt TOS <ignored>: Invalid argument
Connected to 192.168.48.128.
220 mnsb FTP server (SRPftp 1.3) ready.
SRP accepted as authentication type.
Name (192.168.48.128:SRP): tester
Initializing random number generator...done
SRP Password:

```

Obr. 6.3: Zadání uživatelského jména na straně FTP klienta

- ve druhé části zachytil program Wireshark tyto data (obr. 6.4):

- paket 15: K→S: Klient posílá serveru uživatelské jméno (ADAT <username>).
- paket 16: S→K: Server k přijatému už. jménu posílá nazpět hodnoty modulu, generátoru a zasolení (ADAT=<N, g, s>).

- paket 17: K→S: Klient posílá vypočtenou hodnotu svého veřejného klíče serveru („ADAT <wp>“).
- paket 19: S→K: Server posílá vypočtenou hodnotu svého veřejného klíče klientu („ADAT <yp>“).

No. -	Source	Destination	Protocol	Info
14	192.168.48.1	192.168.48.128	TCP	m-wnn > ftp [ACK] Seq=30 Ack=119 win=65417 Len=0
15	192.168.48.1	192.168.48.128	FTP	Request: ADAT AAAABnRlc3Rlcg==
16	192.168.48.128	192.168.48.1	FTP	Response: 335 ADAT=AAAAG06vCrmts43wnDP4CvgPxeHgcmGHdf88C5
17	192.168.48.1	192.168.48.128	FTP	Request: ADAT AAAAGHL5UilLnsh8Snu7nmHfgM09Bs674akfsY1CwaF
18	192.168.48.128	192.168.48.1	TCP	ftp > m-wnn [ACK] Seq=334 Ack=236 win=6432 Len=0
19	192.168.48.128	192.168.48.1	FTP	Response: 335 ADAT=AAAAGEvA/kThumZvk+yHknX3wML+e1NkCoH/F4
20	192.168.48.1	192.168.48.128	TCP	m-wnn > ftp [ACK] Seq=236 Ack=521 win=65015 Len=0

Obr. 6.4: Zachycené pakety druhé části procesu FTP spojení

3. Dokončení autentizace a vytvoření zabezpečeného FTP spojení

- po zadání hesla („test1234“) v klientském programu dojde k finální části autentizace, která probíhá na pozadí a při úspěšném ověření se komunikující strany domluví, které zabezpečovací mechanismy budou používat (typ šifry a hašovací funkce). Program je následně vypíše spolu s výsledkem autentizace (obr. 6.5).

```

C:\ftp.exe
ftp> open 192.168.48.128
ftp: setsockopt TOS (ignored): Invalid argument
Connected to 192.168.48.128.
220 mnsh FTP server (SRPftp 1.3) ready.
SRP accepted as authentication type.
Name (192.168.48.128:SIKI): tester
Initializing random number generator...done
SRP Password:
SRP authentication succeeded.
Using cipher CAST5_CBC and hash function SHA.
200 Protection level set to Private.
232 user tester authorized by SRP.
230 User tester logged in.
ftp>

```

Obr. 6.5: Závěrečná fáze FTP spojení

- v poslední fázi FTP spojení zachytil program Wireshark pakety, jejichž výpis je na obr. 6.6 a mají následující význam:
 - paket 21: K→S: Po výpočtu klíče relace dochází k jeho ověření. Klient posílá žádost o ověření jako zprávu ve tvaru „ADAT <client response>“.
 - paket 23: S→K: Server přichází zprávu ověří a nazpět odešle odpověď ve tvaru „ADAT=<server response>“.
 - paket 24: K→S: Klient rovněž ověří zprávu doručenou od serveru a v kladném případě pošle zprávu „ADAT <0>“. Tím je ověřování dokončeno.

- paket 25: S→K: Server posílá klientu seznam podporovaných šifer a hašovacích funkcí ve tvaru „ADAT=<list of ciphers, list of hashes, seqnum>“.
- paket 26: K→S: Klient zasílá nazpět vybrané možnosti ve tvaru „ADAT <selected cipher, selected hash, seqnum>“.
- paket 27: S→K: Informace o úspěšné autentizaci a zvolené šifře a hašovací funkci.
- pakety 28-35: Výměny zpráv MIC a ENC pro zajištění důvěrnosti a bezpečnosti následné komunikace se již netýkají autentizační části.

No.	Source	Destination	Protocol	Info
20	192.168.48.1	192.168.48.128	TCP	m-wnn > ftp [ACK] Seq=236 Ack=521 win=65015 Len=0
21	192.168.48.1	192.168.48.128	FTP	Request: ADAT AAAAFDurgRTa2sE8ae1eLoyv8Zld0Af7
22	192.168.48.128	192.168.48.1	TCP	ftp > m-wnn [ACK] Seq=521 Ack=275 win=6432 Len=0
23	192.168.48.128	192.168.48.1	FTP	Response: 335 ADAT=AAAAFDVQe3CoudZ7dnOh1fR19Y+OHku
24	192.168.48.1	192.168.48.128	FTP	Request: ADAT AAAAQA=
25	192.168.48.128	192.168.48.1	FTP	Response: 335 ADAT=AAAAEgECaWQFbgcICQoLDA00DXARAAAAAAMBAGAAAAENTWHTW==
26	192.168.48.1	192.168.48.128	FTP	Request: ADAT AAAAQCAAAAABagAAAAATxkPg/
27	192.168.48.128	192.168.48.1	FTP	Response: 235 SRP authentication successful
28	192.168.48.1	192.168.48.128	FTP	Request: MIC AAAAQAAAAEAAAAJUEJTWiAXMDI08ZD4P5xFzqIehrp25b2dzBHERgpec2
29	192.168.48.128	192.168.48.1	FTP	Response: 631 AAAAQAAAAEAAAAANMjAwIFBCU109MTAyNDU8B7c1XFFIFxkjSkxm23dVAf
30	192.168.48.1	192.168.48.128	FTP	Request: MIC AAAAQAAAAEAAAAAGUF3PVCBQ8ZD4QAJw7fyERCfXBPjgs3yZrR85CHH
31	192.168.48.128	192.168.48.1	FTP	Response: 632 AAAAQAAAAIrpUferjYT6B/wx3sfawgbKDztiflXpqvFADb19EbotwEmt8
32	192.168.48.1	192.168.48.128	FTP	Request: ENC AAAAQAAAAJw9w0nmk0edyTT6kiyuUNZ1TOVCQ/NHhLXlffQIWiHw3iyug0
33	192.168.48.128	192.168.48.1	FTP	Response: 632 AAAAQAAAAIRmInk5+GznZ5+ocj3AaBLd2bjZ5EHABvgx0KgmESKMMODf0
34	192.168.48.1	192.168.48.128	FTP	Request: ENC AAAAQAAAAIrFdiBY6qB1OCGUZkuBe9tky0EOEWG3jpu27w1vKI/N2Tq2-
35	192.168.48.128	192.168.48.1	FTP	Response: 632 AAAAQAAAAJ3EqcrYby+Rv5MxYN6ms98GMm1FFwvB3Dox1U1lF3qTvvvFM

Obr. 6.6: Zachycené pakety v závěrečné fázi FTP spojení

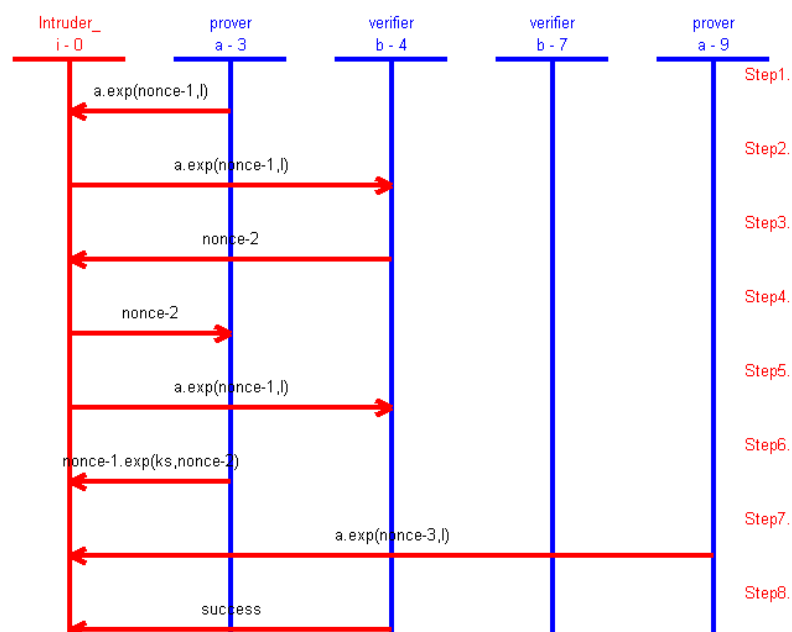
6.4 Bezpečnostní analýza protokolu SRP

K nalezení odpovědi na otázku bezpečnosti protokolu SRP jsem se rozhodl využít nástroje AVISPA (Automated Validation of Internet Security Protocols and Applications), který provádí automatickou validaci bezpečnosti internetových protokolů a aplikací. Jeho hlavní výhodou je možnost použít různé verifikační techniky na stejnou specifikaci daného protokolu. Nástroj AVISPA je možné použít třemi způsoby dostupnými na stránkách projektu [33]:

1. *webové rozhraní nástroje AVISPA*
 - není zapotřebí instalovat žádný software, stačí pouze internetový prohlížeč. Nevýhodou jsou omezené zdroje paměti a procesoru sdílené s ostatními uživateli pouze základní dostupné funkce a nastavení.
2. *nástroj AVISPA verze 1.1*
 - stažení ze stránek až po registraci (vyplněním formuláře), omezeno licencemi. Dostupné pouze na platformu Linux a Mac OS. Nemá grafické rozhraní, spouští se z příkazové řádky.
3. *nástroj SPAN (Security Protocol ANimator for AVISPA)*
 - grafické rozhraní, které umožňuje simulaci běhu protokolu, automatické zobrazení útoků nalezených nástrojem AVISPA či tzv. *intruder mode* - interaktivní vytváření útoků na základě znalosti

potenciálního útočníka. Dostupný je pro všechny platformy (Windows, Linux, Mac OS).

Z nabízených možností jsem zvolil použití nástroje SPAN. Hlavním komunikačním prostředkem je speciální jazyk HLPSL (High-Level Protocol Specification Language), s jehož pomocí se specifikuje konkrétní protokol a definuje, na čem je založena jeho bezpečnost. Jde o formální, modulární jazyk, který při návrhu využívá modelu rolí (obdoba známého modelu „Alice & Bob“, avšak mnohem složitější). Protože sepsání specifikace konkrétního protokolu není zrovna jednoduché a výsledný zdrojový kód je poměrně obtížně čitelný, rozhodli se autoři do nástroje SPAN implementovat tzv. *Message Sequence Charts* (MSC), což je možné představit si jako grafickou realizaci výměny zpráv mezi Alicí a Bobem (viz obr. 6.7). Nejen z uvedeného důvodu náročného sepsání specifikací je na stránkách projektu knihovna obsahující volně dostupné specifikace známých protokolů, mj. např. SIP, RADIUS, Kerberos, TLS a také SRP.

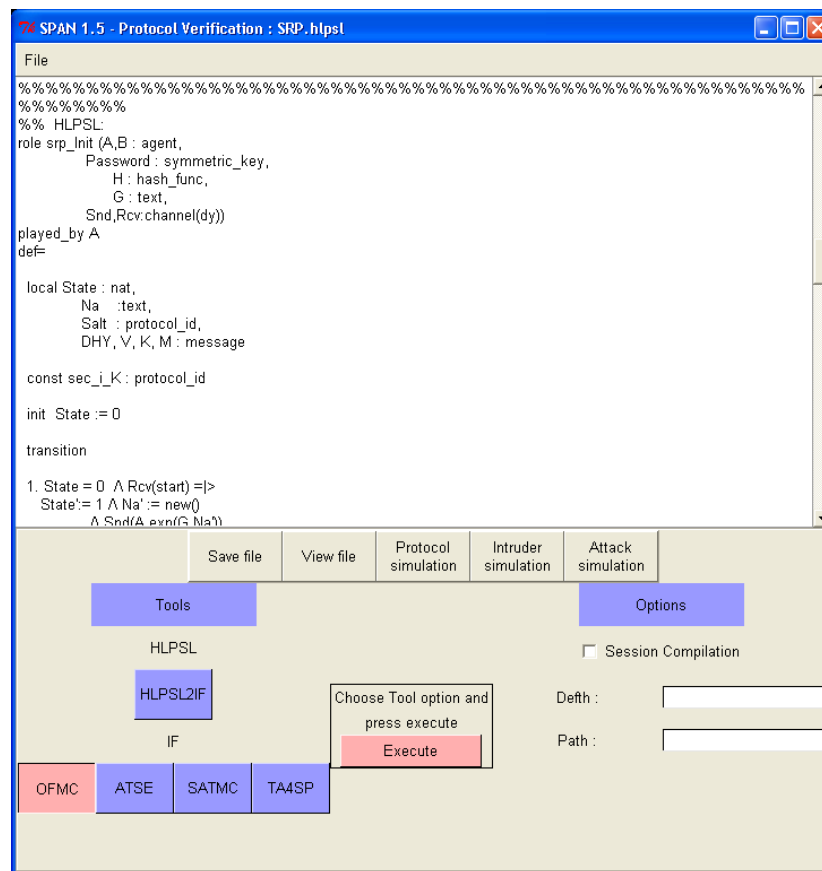


Obr. 6.7: Výměna zpráv v nástroji SPAN pomocí MSC

Finální popis protokolu v jazyce HLPSL je následně převeden do formátu *Intermediate Format* (IF), který používají různé verifikační metody zabudované v nástroji AVISPA. Těmi jsou OFMC (On-the-Fly Model-Checker), CL AtSe (Constraint-Logic-based model-checker Attack Searcher), SATMC (SAT-based Model-Checker) a TA4SP (Tree Automata based Automatic Approximations for the Analysis of Security Protocols). Bližší popis jednotlivých metod není předmětem této práce a je možné se o nich dočíst v dokumentaci nástroje AVISPA dostupné na webových stránkách [33].

Hlavní okno nástroje SPAN (obr. 6.8) se skládá ze čtyř částí: největší část zabírá editor, do kterého se zadává a upravuje specifikace protokolu, kterou je možné buď

přímo psát, nebo nahrát z uloženého souboru pomocí menu *File*→*Open File*. Podporované jsou soubory s příponou „hpsl“, ale prakticky jde o textové soubory, které je možné upravovat v libovolném textovém editoru. Pod oknem editoru je pět tlačítek. Význam prvních dvou je zřejmý podle názvu, zbylé tři spouštějí různé animace: *Protocol simulation* - simulace běhu protokolu bez role útočníka, *Intruder simulation* - v této simulaci můžeme navrhnout vlastní útok na protokol definováním znalosti útočníka a *Attack simulation* - nahraje se útok nalezený metodami nástroje AVISPA. Tyto metody je možné vybrat v levé dolní části okna a spustit tlačítkem *Execute*. Vpravo jsou pak možnosti nastavení, které se liší v závislosti na zvolené metodě.



Obr. 6.8: Hlavní okno nástroje SPAN

6.4.1 Protokol SRP v jazyce HPSL

Jak bylo uvedeno v kap. 6.4, na stránkách projektu AVISPA je možné najít hotové specifikace některých protokolů, mezi kterými je i protokol SRP. Autoři specifikace uvádějí, že jde o mírně zjednodušenou verzi, ve které se ve výpočtech neprovádí operace XOR a modulo n , protože nejsou v jazyce HPSL definovány. Popis výměny zpráv originální verze protokolu je rozebrán v kap. 3.1, proto zde uvedu pouze upravenou verzi přepsanou do jazyka HPSL.

Výpočet klíče relace K a ověřovací zprávy M:

V - verifier, G - generátor, Na, Nb - nonces, náhodné hodnoty generovány na začátku běhu protokolu, operace „.“ (tečka) znamená zřetězení (nahrazení operace sčítání při výpočtu K)

$$K = H(V.(G^{Na})^{Nb})$$

$$M = H(H(G), H(A).Salt.G^{Na}. \{G^{Nb}\}V.K)$$

Průběh výměny zpráv (A - klient, B - server):

A → B : A, G^{Na}

B → A : Salt, {G^{Nb}}V

A → B : M

B → A : H(G^{Na}, M, K)

Ve specifikaci jsou definovány 4 role: srp_Init a srp_Resp jsou role klienta a server. Role session() je složená role ze základních rolí a definuje relaci a komunikační kanál, který je dostupný útočníkovi. Ten z něj může v případě znalosti patřičného klíče brát jakoukoli informaci. V roli environment() je definována znalost útočníka (proměnná intruder_knowledge) a tři souběžné relace. První je běžná komunikace klienta (A) se serverem (B). V další dvou se pak postupně útočník vydává za klienta a za server. Poslední část je goal(), kde se definují cíle bezpečnosti. V případě specifikace protokolu se ověřuje utajení hodnoty klíče relace vypočtené na straně klienta i serveru (secrecy_of sec_i_K, sec_r_K) a autentizace (authentication_on k1, k2) během výměny potvrzovacích zpráv. Ta zajišťuje mj. to, že jsou vypočtené data aktuální v dané relaci a nejsou např. z předešlé relace.

6.4.2 Výsledky hodnocení bezpečnosti nástrojem SPAN

Pro ověření bezpečnosti jsem v nástroji SPAN použil 2 metody: OFMC a CL AtSe. V obou případech byla specifikace protokolu SRP označena za bezpečnou (SAFE), resp. byly splněny podmínky zadané v části goal() a ani jedna metoda nenalezla žádný útok.

7 Zhodnocení bezpečnosti zvolených protokolů

7.1 Autentizační metoda s nulovou znalostí

V kapitole 5 jsem provedl praktický rozbor implementace autentizační metody s nulovou znalostí použité v projektu ZK-SSH. Protože je tato metoda formálně prokázána za bezpečnou pomocí důkazů zmíněných v kap. 3.1, zaměřil jsem se hlavně na vlastní realizaci SSH spojení s autentizací pomocí protokolu Ohta-Okamoto a její porovnání s průběhem komunikace mezi klientem a serverem a rovněž se zachycenými daty během procesu připojení k SSH serveru.

Instalace nástroje OpenSSH s podporou metody s nulovou znalostí a následná konfigurace klienta a serveru proběhla podle postupu uvedeného v kap. 5.2 bez problémů a z pohledu správce sítě nezabere provedení několika příkazů mnoho času. V případě velkého počtu uživatelů se již zajištění podpory této metody značně prodlouží, neboť je potřeba každému uživateli vygenerovat nové soukromé a veřejné klíče a uložit je v jejich domovském adresáři. Důležité je tedy zajištění bezpečné distribuce a uložení klíčů. Popsaná autentizační metoda je navíc podporovaná pouze pod systémem Linux a není mi známa žádná implementace SSH klienta, která by tuto metodu podporovala na jiných operačních systémech. Při realizaci SSH spojení např. z platformy Windows se při volbě autentizace použije běžná metoda ověření pomocí hesla (pokud není nastaveno jinak v konfiguračním souboru serveru `sshd_config`).

Implementace protokolu Ohta-Okamoto v autentizaci ZK-SSH do programu OpenSSH je popsána v kap. 6.3. Průběh komunikace klienta a serveru v části autentizace podle výpisu *debug* zpráv přesně odpovídá zdrojovému kódu souborů, týkajících se nové metody autentizace. V této práci jsem uvedl pouze nejdůležitější části zdrojových kódů, které se přímo týkají autentizace a popisují přenášená data. Přesný význam všech dat zachycených programem Wireshark je obtížné určit vzhledem k faktu, že od ustanovení klíče relace protokolem Diffie-Helman probíhá komunikace šifrovaně, což zvyšuje bezpečnost následné autentizace. Nicméně ze znalosti implementace a fungování metody je možné určit pakety, které se týkají autentizace nulovou znalostí a podle jejich opakující se velikosti říci, jakou informaci by měly obsahovat. Protože jsou ale data šifrována, musely by se na dešifrování aplikovat určité techniky kryptoanalýzy. Po úspěšném dešifrování by však potenciální útočník musel prolomit samotnou metodu nulové znalosti, která je ale formálně prokázána za bezpečnou.

Pro ověření protokolu Ohta-Okamoto v nástroji AVISPA jsem se pokusil o sepsání jeho specifikace v jazyce HLPSL, ale přesto, že po formální stránce se mi podařilo specifikaci úspěšně odladit, nebyl protokol v nástroji SPAN a použitých metodách OFMC a CL AtSe označen za bezpečný. Hlavním důvodem je fakt, že se mi nepovedlo implementovat cyklus s opakováním běhu protokolu a taktéž jazyk HLPSL nepodporuje aritmetickou operaci modulo n a právě na těchto dvou principech je založena bezpečnost protokolu. Moji realizaci specifikace jsem se přesto rozhodl přidat na přiložené médium v případě zájmu o další úpravu a použití.

7.2 Autentizace protokolem SRP

K rozboru bezpečnostní analýzy protokolu SRP jsem vybral jeho implementaci v projektu The Stanford SRP Authentication Project, které jsem se věnoval v kapitole 6. Autor projektu v něm poskytuje zabezpečené verze programů *ftp* a *telnet*, které podporují autentizaci protokolem SRP. Tu je možné nastavit jako jedinou možnou a znemožnit tak přihlásit se nezabezpečeně. Příprava systému se odlišuje podle toho, zda jsme uživatel nebo správce. Ten musí zajistit převod systému uložení hesel z běžného (hesla uložena v souboru */etc/passwd*) do tzv. *EPS* systému (hesla uložena souboru */etc/tpasswd*). Správné nastavení systému zabere trochu víc úsilí než u předchozí metody, pokud se ale správci podaří zajistit, aby si každý uživatel vygeneroval nové heslo, ušetří si tím mnoho času. Instalace na straně klienta je jednoduchá, vystačí si s hotovými zkompilevanými programy a musí si pouze vygenerovat nové heslo. Výhodou je také podpora více operačních systémů. Na platformu Windows jsou ale dostupné pouze klientské aplikace klientů (Win32 SRP FTP, Win32 SRP Telnet).

V kapitole 6.2 jsem se zaměřil na rozbor zdrojových kódů. Ty jsou psány hierarchickou formou, kde na nejvyšší úrovni jsou funkce v souborech programů *ftp* a *telnet* (ve složce */ftp*, resp. */telnet*), které odkazují na funkce hlavního souboru SRP protokolu - *spr.c* - v knihovně *libsrp* (ve stejné pojmenované složce). Tam se podle použité verze protokolu odvolává na příslušné soubory pro danou verzi. Na nejnižší úrovni jsou pak funkce z kryptografických knihoven, které si je možné zvolit při instalaci. Protože by byl popis všech takto vnořených funkcí daleko za hranicemi rozsahu této práce, uvedl jsem pouze funkce hlavního souboru s detailním rozбором nejdůležitější části - výpočtu klíče relace.

Z rozboru zachycených dat a posloupnosti vykonávaných operací (příkazů) je v porovnání s analýzou zdrojového kódu implementace SRP autentizace patrná jedna změna. Generování a výměna veřejných klíčů mezi klientem a serverem se u FTP spojení provádí ihned po odeslání uživatelského jména klienta, tedy předtím, než klient zadá své heslo. V popisu implementace SRP autentizace dochází k výměně veřejných klíčů a následnému výpočtu tajného klíče relace až po zadání hesla klientem. Toto prohození pořadí provedených příkazů by však nemělo mít vliv na správnost funkce protokolu, jelikož při výpočtu veřejných klíčů se heslo nepoužívá a je použito až při výpočtu klíče relace (na straně klienta). Kromě této změny odpovídají zachycené data a jejich pořadí teoretické analýze implementace.

K ověření bezpečnosti jsem použil nástroj AVISPA implementován do grafického prostředí nástroje SPAN. Oba se dnes již běžně používají k verifikaci kryptografických protokolů, o čemž svědčí i poměrně dlouhý seznam knihovny protokolů na stránkách projektu, které byly přepsány do jazyka HLPSL a následně ověřeny nástrojem AVISPA. Je mezi nimi i protokol SRP. Jeho specifikaci jsem popsal v kapitole 6.4 a provedl ověření dvěma metodami (OFMC a CL AtSe). Ani jedna z nich nenalezla žádný možný útok, a protože byly splněny podmínky na utajení klíče relace a procesu autentizace, byl označen za bezpečný. Autoři specifikace však uvádějí, že jde o mírně zjednodušenou verzi a nemám možnost nijak ověřit správnost této specifikace. Přesto bych si při výběru mezi popisovanými metodami autentizace vybral právě tuto, protože nabízí zdokonalení autentizace heslem, v současné době stále ještě nejčastější autentizační metody a navíc zajišťuje silnou autentizaci použitím snadno zapamatovatelného hesla.

8 Závěr

Předmětem této diplomové práce byl popis současně používaných protokolů pro ověření identity uživatelů pomocí hesla a vyhledání a popsání nových metod a trendů v této oblasti. Mezi vybranými metodami pak zvolit nejvhodnější pro praktickou implementaci, kterou následně ověřit v praxi a popsat podrobně všechna přenášená data. V závěru pak provést bezpečnostní analýzu použitých protokolů.

V teoretické části jsem postupně popsal stručně protokoly Kerberos, RADIUS a Diameter. Všechny tyto protokoly jsou založeny na přítomnosti příslušného autentizačního serveru, který udržuje veškeré informace týkající se zabezpečení. Takový server se pak stává cílem útoku a vyžaduje se vysoká úroveň jeho zabezpečení. Další nevýhodou je, že tyto protokoly, resp. hlavně Kerberos, nebyly konstruovány na požadavky dnešních sítí s důrazem na mobilní komunikace. Mezi novými protokoly a metodami jsem vybral protokoly s nulovou znalostí a jejich zástupce v podobě protokolu Fiat-Shamir a Ohta-Okamoto. Druhou skupinu tvoří password-based protokoly se zástupcem SRP založeném na principu protokolu Diffie-Helman a protokol SNAPi, který se opírá o bezpečnost RSA.

V první kapitole praktické části jsem provedl podrobný rozbor implementace metody s nulovou znalostí, založené na protokolu Ohta-Okamoto a realizované v projektu ZK-SSH. Hlavní část je věnovaná porovnání samotné realizace s teoretickým návrhem protokolu uvedeným v kap. 2.3 a s přenášenými daty zachycenými programem Wireshark. V další kapitole jsem se zaměřil na protokol SRP a jeho implementaci v programech *ftp* a *telnet*. Stejně jako v prvním případě jsem provedl porovnání teoretického rozboru s praktickou realizací a navíc pomocí nástroje AVISPA ověřil bezpečnost protokolu. U protokolu Ohta-Okamoto založeném na nulové znalosti jsou matematicky doloženy důkazy o bezpečnosti a rozbor implementace a komunikace odpovídal zachyceným datům mezi klientem a serverem. Protokol SRP sice nemá formálně doložené důkazy o bezpečnosti, přesto nástroj AVISPA, který se používá k validaci internetových protokolů a jejich bezpečnosti nenalezl žádný útok a označil jej za bezpečný. Z mého pohledu se vzhledem k praktickému použití a implementaci v programech komunikujících v prostředí nezabezpečeného internetu jeví lépe protokol SRP. A mimo jiné také proto, že vylepšuje autentizaci heslem, která patří stále mezi nejčastější typy autentizace.

Literatura

- [1] SMITH, Richard E. *Authentication: From Passwords to Public Keys*. [s.l.] : Addison-Wesley, 2002. 549 s. ISBN 0-201-61599-1.
- [2] FIAT, Amos, SHAMIR, Adi. *Authentication: From Passwords to Public Keys*. [s.l.] : Springer-Verlag, 1987. 186-194 s. Proceedings of the CRYPTO'86. ISBN 0-387-18047-8.
- [3] BOYD, Colin, MATHURIA, Anish. *Protocols for authentication and key establishment*. Berlin : Springer-Verlag, 2003. s.91-93, s.247-285. ISBN 3-540-43107-1.
- [4] QUISQUATER, Jean-Jacques , GUILLOU, Louis C. , BERSON, Thomas A. How to Explain Zero-Knowledge Protocols to Your Children.. In *Advances in Cryptology - CRYPTO '89*. [s.l.] : [s.n.], 1990. s. 628-631. Dostupný z WWW: <<http://pages.cs.wisc.edu/~mkowalczyk/628.pdf>>.
- [5] OSTROVSKY, Rafail, VENKATESAN, Ramarathnam, YUNG, Moti. Interactive hashing simplifies zero-knowledge protocol design. *Aaa*. 1111, no. 1, s. 1. Dostupný z WWW: <<http://dsns.csie.nctu.edu.tw/research/crypto/HTML/PDF/E93/267.PDF>>.
- [6] HAJNÝ, Jan. Úvod do Zero-Knowledge protokolů. *Elektrorevue* [online]. 2008 [cit. 2008-11-18], s. 8. ISSN 1213-1539. Dostupný z WWW: <<http://www.elektrorevue.cz/cz/download/uvod-do-zero-knowledge-protokolu/>>.
- [7] CORON, Jean-Sébastien, NACCACHE, David. Cryptanalysis of a Zero-Knowledge Identification Protocol of Eurocrypt '95 . *LNCS*. 2004, no. 2964/2004, s. 1996. Dostupný z WWW: <<http://www.springerlink.com/content/9crftkgt842uwqbr/fulltext.pdf>>.
- [8] *ZK-SSH - A Zero Knowledge Implementation for OpenSSH* [online]. 2005 [cit. 2008-11-19]. Dostupný z WWW: <<http://zk-ssh.cms.ac/>>.
- [9] GOLDREICH, Oded. Zero-knowledge twenty years after its invention. 2004, s. 35. Dostupný z WWW: <<http://www.wisdom.weizmann.ac.il/~oded/PS/zk-tut02v4.ps>>.
- [10] KLÍMA, Vlastimil, ROSA, Tomáš. Nové výzkumy: vypočtěte si kvalitu hesla. *Kryptologie pro praxi* [online]. 2006 [cit. 2008-12-01]. Dostupný z WWW: <http://crypto-world.info/klima/2006/ST_2006_10_13_13.pdf>.
- [11] KLÍMA, Vlastimil. *Personal page: Vlastimil Klima, Dr.* [online]. [cit. 2008-12-01]. Dostupný z WWW: <<http://cryptography.hyperlink.cz/>>.
- [12] WU, Thomas. The Secure Remote Password Protocol. [online]. 1997 [cit. 2008-12-02], s. 17. Dostupný z WWW: <<http://srp.stanford.edu/ndss.html>>.
- [13] WU, Thomas. *The Stanford SRP Authentication Project* [online]. [cit. 2008-12-02]. Dostupný z WWW: <<http://srp.stanford.edu/>>.

- [14] WU, Thomas. SRP-6: Improvements and Refinements to the Secure Remote Password Protocol. *Ertrdfg* [online]. 2002 [cit. 2008-12-02], s. 05. Dostupný z WWW: <<http://srp.stanford.edu/srp6.ps>>.
- [15] MIT krb5 Security Advisory 2007-001. 2007. Dostupný z WWW: <<http://web.mit.edu/kerberos/advisories/MITKRB5-SA-2007-001-telnetd.txt>>.
- [16] MACKENZIE, Philip, SWAMINATHAN, Ram. Secure Network Authentication with Password Identification. *Dfgd* [online]. 1999 [cit. 2008-12-03], s. 11. Dostupný z WWW: <<http://grouper.ieee.org/groups/1363/passwdPK/contributions/snapi.ps>>.
- [17] WANG, X., FENG, D., LAI X., YU, H. Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD, rump session, CRYPTO 2004, Cryptology ePrint Archive, Report 2004/199. Dostupný z WWW: <<http://eprint.iacr.org/2004/199>>
- [18] KLÍMA, Vlastimil. Hašovací funkce, principy, příklady a kolize. *Kryptologie pro praxi* [online]. 2005 [cit. 2008-12-03]. Dostupný z WWW: <http://crypto-world.info/klima/2005/cryptofest_2005.htm>.
- [19] KLÍMA, Vlastimil. Stránka věnovaná kolizím hašovacích funkcí. [online]. [cit. 2008-12-03]. Dostupný z WWW: <http://cryptography.hyperlink.cz/2004/kolize_hash.htm>
- [20] The Keyed-Hash Message Authentication Code (HMAC). [online]. 2002 [cit. 2008-12-03]. Dostupný z WWW: <<http://csrc.nist.gov/publications/fips/fips198/fips-198a.pdf>>.
- [21] KLÍMA, Vlastimil. Nový koncept hašovacích funkcí SNMAC s využitím speciální blokované šifry a konstrukcí NMAC/HMAC. [online]. 2006 [cit. 2008-12-03], s. 22. Dostupný z WWW: <http://cryptography.hyperlink.cz/SNMAC/SNMAC_CZ.pdf>.
- [22] KLÍMA, Vlastimil. Rodina speciálních blokových šifer DN a hašovacích funkcí nové generace HDN typu SNMAC. [online]. 2007 [cit. 2008-12-05]. Dostupný z WWW: <http://cryptography.hyperlink.cz/SNMAC/DN_HDN_CZ.pdf>.
- [23] KOHL, J., NEUMAN, C. RFC 1510: The Kerberos Network Authentication Service (V5). 1993, s. 112. Dostupný z WWW: <<http://www.rfc-archive.org/getrfc.php?rfc=1510>>.
- [24] RIGNEY, C., WILLENS, S., RUBENS, A., SIMPSON W. RFC 2865: Remote Authentication Dial In User Service (RADIUS). 2000, s. 76. Dostupný z WWW: <<http://tools.ietf.org/html/rfc2865>>.
- [25] HILL, Joshua. An Analysis of the RADIUS Authentication Protocol. 2001. Dostupný z WWW: <<http://www.untruth.org/~josh/security/radius/radius-auth.html>>.
- [26] *OpenSSH* [online]. c1999-2009 [cit. 2009-04-14]. Dostupný z WWW: <<http://www.openssh.com/>>.

- [27] CALHOUN, P., LOUGHNEY, J., GUTTMAN, E., ZORN, G., ARKKO, J. RFC 3588: Diameter Base Protocol. 2003, s. 147. Dostupný z WWW: <<http://tools.ietf.org/html/rfc3588>>.
- [28] *Debian "lenny" Release Information* [online]. c1997-2009 [cit. 2009-04-20]. Dostupný z WWW: <<http://www.debian.org/releases/stable/index.en.html>>.
- [29] *VMware Player : Run Virtual Machines on Linux or Windows PCs for Free* [online]. c2009 [cit. 2009-04-18]. Dostupný z WWW: <<http://www.vmware.com/products/player/>>.
- [30] *Wireshark : Go deep* [online]. [2009] [cit. 2009-04-25]. Dostupný z WWW: <<http://www.wireshark.org/>>.
- [31] HOROWITZ, M. FTP Security Extensions. *IETF* [online]. 1997 [cit. 2009-05-10]. Dostupný z WWW: <<http://www.ietf.org/rfc/rfc2228.txt>>.
- [32] *OpenSSL* [online]. c1999-2005 [cit. 2009-04-16]. Dostupný z WWW: <<http://www.openssl.org/>>.
- [33] *The AVISPA project* [online]. [2001-] [cit. 2009-05-05]. Dostupný z WWW: <<http://avispa-project.org/>>.
- [34] Çetin K., KOC, Tolga, ACAR. Montgomery Multiplication in $GF(2^k)$. *Designs, Codes and Cryptography* [online]. 1998, vol. 14, no. 1 [cit. 2009-05-13], s. 57-69. Dostupný z WWW: <<http://www.csd.uwo.ca/~eschost/Exam/Koc.pdf>>.

Seznam zkratek

AES	Advanced Encryption Standard
AKE	Asymmetric Key Exchange
AVISPA	Automated Validation of Internet Security Protocols and Applications
CBD	Což Bylo Dokázáno
DES	Data Encryption Standard
DSA	Digital Signature Algorithm
EEPROM	Electrically Erasable Programmable Read-Only Memory
EKE	Encrypted Key Exchange
ENC	Encoding protected Command
EPS	Exponential Password Suite
FTP	File Transfer Protocol
HDN	Hash Double Net
HLPSL	High-Level Protocol Specification Language
HMAC	Keyed-hash Message Authentication Code
IETF	Internet Engineering Task Force
MD5	Message-Digest algorithm
MIC	Message Integrity protected Command
MSC	Message Sequence Charts
NSD	Největší společný dělitel
OKE	Open Key Exchange
OS	Operační Systém
PAM	Pluggable Authentication Module
PGP	Pretty Good Privacy
RADIUS	Remote Authentication Dial In User Service
RIPEMD	RACE Integrity Primitives Evaluation Message Digest
RSA	Ron Rivest, Adi Shamir, Len Adleman (1977)
SBŠ	Speciální Blokovaná Šifra
SCTP	Stream Control Transmission Protocol
SHA-0,1,2	Secure Hash Algorithm
SNAPI	Secure Network Authentication with Password Identification
SNMAC	Special Nested Message Authentication Codes
SPAN	Security Protocol ANimator for AVISPA
SPEKE	Simple Password Exponential Key Exchange
SRP	Secure Remote Password
SSH	Secure SHell
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
ZK-SSH	A Zero Knowledge Implementation for OpenSSH

A Obsah přiloženého média

Na přiloženém médiu lze kromě elektronické verze této práce najít všechny instalační programy a balíčky, které jsem použil při praktickém rozboru. Navíc jsou přiloženy konfigurační soubory SSH klienta a serveru, výpisy jejich vzájemné komunikace a kompletní popis zachycený dat programem Wireshark. Soubory jsem rozdělil podle náležitosti ke konkrétním metodám do následující struktury:

Adresář SRP:

- složka *linux_pkgs* - obsahuje instalační balíčky pro platformu Linux/i386
- složka *win_ftp* - FTP klient s podporou SRP autentizace pro Windows
- složka *wireshark* - výpisy paketů zachycených programem Wireshark v pdf souborech (*expanded* - podrobný výpis - 37 str., *collapsed* - stručný výpis 5 str.)

Adresář ZK-SSH:

- složka *config* - obsahuje konfigurační soubory pro SSH klienta s server
- složka *debug* - obsahuje výpisy tzv. *debug* zpráv při komunikaci mezi klientem a serverem
- složka *source* - obsahuje balíček OpenSSH a *patch* soubor k rozšíření o metodu autentizace s nulovou znalostí
- složka *wireshark* - výpisy paketů zachycených programem Wireshark v pdf souborech (*expanded* - podrobný výpis - 86 str., *collapsed* - stručný výpis 20 str.)

Adresář AVISPA:

- složka *source* - obsahuje instalační soubory
- složka *hlpsl* - obsahuje specifikace protokolů Ohta-Okamoto a SRP v jazyce HLPSL