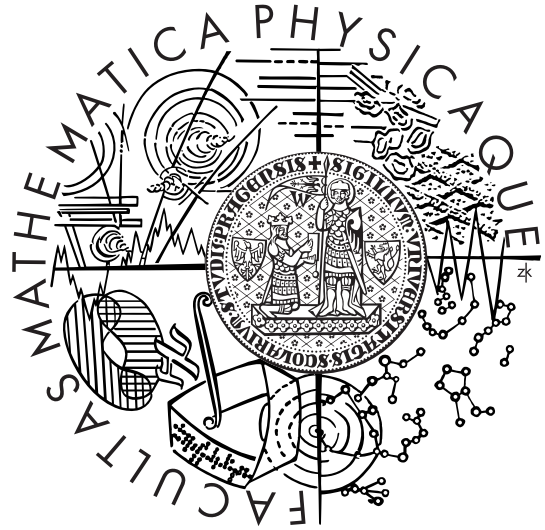
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



Vítězslav Fabian

Kryptografická podpora pro aplikační server SAP

Katedra softwarového inženýrství
Vedoucí diplomové práce: RNDr. Antonín Beneš, Ph.D.
Studijní program: Informatika

Chtěl bych na tomto místě poděkovat všem, kteří mi při psaní této diplomové práce pomáhali. Zejména patří můj dík rodičům, kteří mě po celou dobu studia podporovali. Dále bych chtěl poděkovat vedoucímu diplomové práce RNDr. Antonínu Benešovi, Ph.D. za jeho pomoc a cenné podněty.

Prohlašuji, že jsem svou diplomovou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne 6.8.2007 Vítězslav Fabian

Obsah

Název práce: Kryptografická podpora pro aplikační server SAP
Autor: Vítězslav Fabian
Katedra: Katedra softwarového inženýrství
Vedoucí diplomové práce: RNDr. Antonín Beneš, Ph.D.
E-mail vedoucího: antonin.benes@sap.com

Abstrakt:

Aplikační server SAP používá pro zajištění potřebných kryptografických funkcí externí knihovny. Tyto knihovny jsou volány prostřednictvím standardního rozhraní GSS-API.

Hlavním výsledkem této práce je implementace knihovny podporující zmíněné rozhraní. To představuje implementaci dynamické knihovny v programovacím jazyce C splňující standard GSS-API verze 2. Jako autentizační mechanismus byla zvolena autentizace X.509 certifikáty a pro zajištění kryptografických operací jsme sáhli po knihovně OpenSSL.

Knihovna a všechny její části jsou přenositelné a šiřitelné s otevřeným zdrojovým kódem.
Klíčová slova:

GSS-API, OpenSSL, PKI, otevřený zdrojový kód

Title: Cryptographical support of SAP application server
Author: Vítězslav Fabian
Department: Department of Software Engineering
Supervisor: RNDr. Antonín Beneš, Ph.D.
Supervisor's e-mail address: antonin.benes@sap.com

Abstract:

SAP application server uses external library to provide requiered cryptographical functions. Those libraries are called through standard GSS-API interface.

Main result of this work is practical implementation of specific library that supports above mentioned interface. This is done by C language written dynamic library implementation that supports standard GSS-API v.2 interface. For authentisation X.509 certificate was chosen and for coded cryptographic operations OpenSSL library was chosen.

Library and all of its components is portable and open source.

Keywords:

GSS-API, OpenSSL, PKI, open source

Kapitola 1

Úvod

Internet je fenoménem dneška. Ne nadarmo byl za osobnost roku 2006 časopisu LIFE vyhlášen každý aktivní uživatel tohoto media. Bohužel jako každý úspěch, i prudký rozvoj internetu přilákal pozornost negativních živlů, které jej chtějí zneužívat. Existuje tak nutnost tyto moderní trendy komunikace chránit. Jednotlivé aplikace se mohou pokusit bezpečnost zajistit samy svým proprietárním řešením nebo sáhnout po obecném osvědčeném standardním řešení třetích stran.

Jedním takovým osvědčeným řešením, které používají i aplikační servery firmy SAP, je nasazení bezpečnostní knihovny splňující požadavky standardního rozhraní GSS-API[1].

Na trhu existuje řada komerčních GSS-API knihoven. Jejich kvalita není vždy zjevná, díky uzavřenosti kódu. Na druhé straně jsou k dispozici prověřené a otevřené bezpečnostní knihovny, které však neposkytují standardní GSS-API rozhraní. Naším úkolem bylo spojit tyto dvě věci dohromady – standardní bezpečností rozhraní GSS-API a prověřenou otevřenou bezpečnostní knihovnu.

1.1 Cíl práce

Cílem této diplomové práce bylo implementovat GSS-API knihovnu spolupracující s aplikačními servery SAP. To představovalo implementaci dynamické knihovny v programovacím jazyce C splňující standard GSS-API verze 2.

Dále byl kladen požadavek, aby knihovna a všechny její části byly šiřitelné s otevřeným zdrojovým kódem. Rovněž bylo vyžadováno co nejjednodušší nasazení a správa vytvořeného systému.

[1]GSS-API = General Security Service Aplication Program Interface

Jako autentizační mechanismus byla zvolena autentizace X.509 certifikáty [11]. Pro zajištění kryptografických operací jsme použili knihovnu OpenSSL [24].

Spojení GSS-API rozhraní, autentizace na základě standardu X.509 [8] a otevřené kryptografické knihovny OpenSSL by měl být hlavní přínos této diplomové práce.

Kapitola 2

Architektura GSS-API knihoven

V této kapitole se pokusíme osvětlit fungování GSS-API knihoven. V první část je zaměřena na GSS-API knihovnu z vnějšku, z pohledu aplikace. V druhá část se zabývá vnitřní strukturou knihovny.

2.1 Vnější podhled - GSS-API

General Security Service Aplication Program Interface poskytuje rozhraní k volání bezpečnostních služeb aplikacím. Umožňuje komunikujícím aplikacím autentizovat se, delegovat práva jiným aplikacím a užívat bezpečnostní služby na úrovni jednotlivých zpráv.

Rozhraní je definováno pěti typy volání:

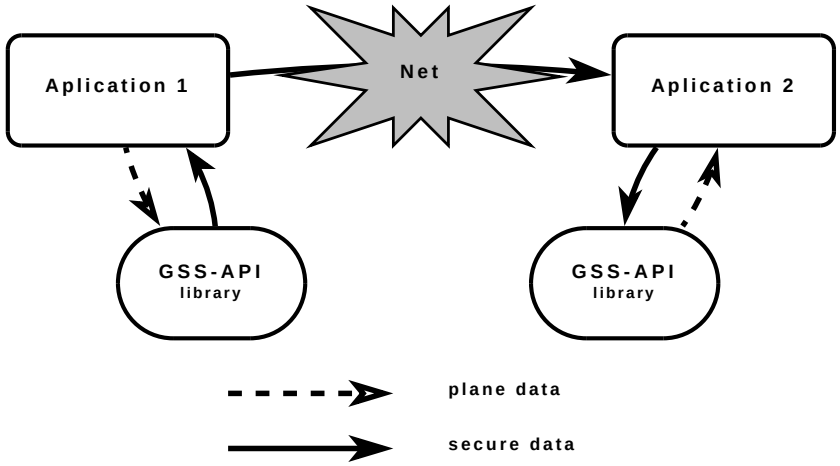
- management jmen
- management credenciálů
- management kontextů
- zabezpečení zpráv
- pomocná volání

Jmény se rozumí identifikace entit, které se účastní komunikace. Jednou ze služeb GSS-API knihovny je ověření vazby mezi jménem a entitou, která se jím představuje.

Credenciály jsou data potřebná k autentizaci entit při zřizování kontextů.

Kontexty jsou data nutná k samotné komunikaci (např. jména komunikujících entit, dohodnutá kryptografie, společný kontextový klíč, doba platnosti kontextu).

Zabezpečení jednotlivých zpráv poskytuje integritu a utajení dat, případně i ochrany detekce opakování (replay detection) a uspořádání (sequencing).



Obrázek 2.1: Komunikace pomocí GSS-API knihoven.

Obě komunikující strany získají voláním GSS-API knihovny GSS-API jméno své entity a jemu příslušné credenciály. Pak navážou nezabezpečený komunikační kanál (Způsob komunikace není předmětem GSS-API.) Jedna ze stran iniciuje proces zřizovaní kontextu příslušným voláním GSS-API knihovny. Výsledkem je kontext (částečně nebo zcela zřízený v závislosti na zvoleném postupu) a datový token, který obsahuje data potřebná ke zřízení kontextu. Tento token je přenesen komunikačním kanálem protistraně. Tam jsou data opět předána GSS-API knihovně a výstupem volání je částečně nebo zcela zřízený kontex a případně i další token, který je opět přenesen druhé straně. Takto obě strany pokračují v handshaku dokud nedojde k jejich autentizaci (je-li vyžadována) a ukončení zřizování kontextu.

Zabezpečení komunikace pak probíhá dle obrázku 2.1. Aplikace volá GSS-API knihovnu a předává jí otevřená data. Knihovna provede příslušné operace a v návratových hodnotách vrací token, který obsahuje zabezpečená data, zpět aplikaci. Aplikace doručí tento token svému protějšku. Ten rovněž volá GSS-API knihovnu a vrací se mu původní otevřená data.

V okamžiku kdy už nepotřebují aplikace spolu komunikovat pak obě strany přeruší komunikační kanál a (opět voláním GSS-API knihovny) uvolní kontext.

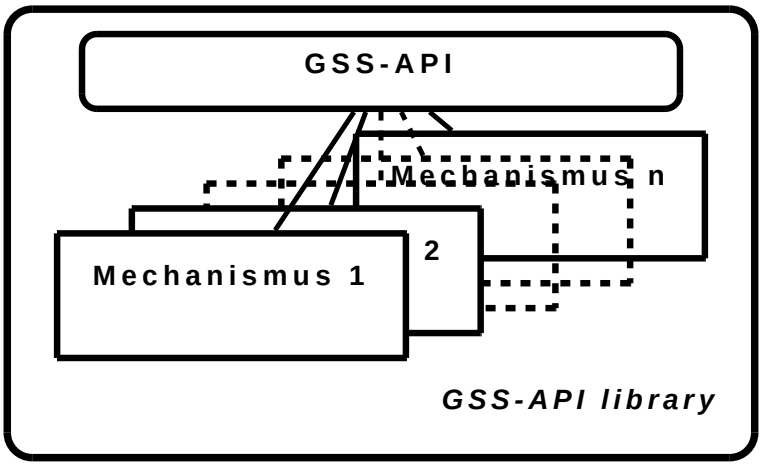
Nejnovějším standardem je GSS-API verze 2, update 1 vydaným jako [1]. Znalost tohoto standardu je doporučována a následující text ji předpokládá. Z obecného standardu pak vycházejí jednotlivé standardy pro programovací jazyky, C-bindings [2] a Java-bindings [3].

2.2 Vnitřní podhled - mechanismy

Výše zmíněné standardy však nijak nepředepisují, jakou formou mají být bezpečnostní funkce knihovny zajišťovány. Tato starost leží na bedrech jednotlivých mechanismů.

Mechanismus by měl definovat:

- způsob autentizace
- formát tokenů jednotlivých zpráv
- použitou kryptografii a QoP[1]
- způsob dohody na kontextovém klíči a generování sub-klíčů



Obrázek 2.2: Vnitřní uspořádání GSS-API knihovny.

[1]QoP = Quality of Protection (vysvětleno v části 4.3)

Každá GSS-API knihovna implementuje jeden nebo více mechanismů (obrázek 2.2). Každý z těchto mechanismů může definovat svůj prostor jmen. Struktura obsahující jméno je pak složena z více částí, každé příslušné některému mechanismu. Podobně i credenciály mohou byt tvořeny různými částmi pro různé mechanismy.

Požadovaný mechanismus je zvolen při volání knihovny volbou příslušného identifikátoru mechanismu.

Kapitola 3

Diskuze existujících mechanismů

V minulé kapitole byla ukázána důležitost mechanismů v architektuře GSS-API knihoven. V této kapitole budou představeny nejznámější mechanismy. Dále budou vyjmenovány i příklady knihoven, které je implementují.

Obecně je možno mechanismy rozdělit na definované standardem a na proprietární. Velkou výhodou standardizovaných mechanizmů je vzájemná interoperabilita knihoven různých dodavatelů.

3.1 Kerberos

Kerberos je populární autentizační nástroj (najdeme ho i v počítačových laboratořích MFF UK[1]), který dospěl již do své páté generace [4]. Kerberos také existuje jako standard pro GSS-API [5].

Zjednodušeně autentizace systémem Kerberos dle následujícího popisu: Klient zašle autentizačnímu serveru (AS) žádost o credenciály k příslušnému serveru. AS odpovídá těmito credenciály zašifrovanými klientovým klíčem. Credenciály se skládají z ticketu pro zvolený server a dočasného šifrovacího klíče (často nazývaný "session key"). Klient zašle ticket serveru (obsahuje identifikaci klienta a kopii dočasného klíče, to vše zašifrované klíčem serveru). V tomto okamžiku mají obě komunikující strany dočasný klíč. Ten je využit k autentizaci klienta (případně i serveru). Dále slouží k výměně sub-session klíčů. Každý systém obsahuje aspoň jeden AS, který drží seznam všech entit (uživatelů a serverů) a jejich klíčů.

[1]MFF UK = Matematicko-fyzikální fakulta Univerzity Karlovy

3.1.1 Výhody a nevýhody

Velkou výhodou je dnes již široké rozšíření knihovny Kerberos, např. i SSPI od Microsoftu umí zpracovávat jeho komunikační tokeny. Výhodou bývala i nižší náročnost na systémové prostředky způsobená použitím pouze symetrické kryptografie. Tato výhoda však rozvojem výkonu počítačů upadá.

Na druhou stranu nevýhodou je absence QoP v GSS-API návrhu. Dalším problémem případného nasazení muže být přítomnost centralizovaných prvků - autentizačních serverů - a tím vhodnost spíše k lokálnímu nasazení.

3.1.2 Příklady

- MIT Kerberos 5
- CyberSafe TrustBroker
- Heimdal Kerberos 5
- Shishi (GNU projekt)

3.2 SPKM

Simple Public-Key GSS-API Mechanism [6] navazuje na úspěch knihovny Kerberos, jen místo infrastruktury symetrických klíčů implementuje klíče asymetrické. Mechanismus se snaží o co největší kompatibilitu se standardy X.509 [8] a PEM [7].

Dokument popisuje dva mechanismy. SPKM-1 používá z důvodu ochrany replay-detection[2] při inicializaci kontextů náhodná čísla, SPKM-2 pak časových razítek, jak vyžaduje X.509 část 10. Autentizace se zajišťuje ověřením elektronického podpisu příslušného veřejného klíče. Asymetrická kryptografie je použita i k výměně kontextového klíče, který pak slouží ke generování klíčů jednotlivých zpráv.

3.2.1 Výhody a nevýhody

Výhodou je možnost užití klasických kryptografických podpisových schémat ve funkcích `gss_getMIC` a `gss_wrap`. Dále velkým pozitivem je možnost využít již existující PKI[3] bez nutnosti složitých zásahů.

[2]Ochrana replay detection brání útočníkovi opakovaně zneužívat již jednou přijaté zprávy.

[3]PKI = Public Key Infrastucture

Velkou nevýhodou je zastaralost tohoto standardu. Tam, kde Kerberos po vydání GSS-API verze 2 již prošel několika updaty, SPKM na své stále čeká. Stejně tak i PKI se od doby vydání tohoto standardu posunulo mílovými kroky kupředu, a tak praxe, kdy k ověření certifikátu se kromě certifikačního řetězce přibalují i příslušná CRL[4], je zde nenaplnitelná.

Další podstatnou nevýhodou je způsob generování sub-klíčů jednotlivých zpráv. Ten obsahuje zásadní chybu, neboť všechny sub-klíče pro daný algoritmus a danou službu jsou vždy stejné. A právě opakování stejného sub-klíče může útočníkovi napomoci k jeho prolomení, a tím i odhalení celé komunikace.

3.2.2 Příklady

- Entrust Authority GSS-API Toolkit for C
- SECUrity Developed Enviroment, SECUDE International AG

3.3 Proprietární mechanismy

Při našem zkoumání a výběru mechanismu nebyla proprietárním řešením dávána váha, neboť absence standardu znemožňuje jejich interoperabilitu. Tyto mechanismy jsou většinou součástí kompletního řešení poskytovaného danou firmou a k jejich vnitřnímu uspořádání není přístup. Proto je zde uvádíme jen na doplnění.

3.3.1 Příklady

- AM-DCE, Bull
- safelayer, Safelayer Secure Communications S.A.
- NEC Secureware, NEC
- LISSIE-SNC, Lissi
- eSecure, Kobil GmbH

[4]CRL = Certificate Revocation List

Kapitola 4

Volba mechanismu

Pro návrh nové knihovny je nejdůležitější volba jejích mechanismů. Ty nejvíce determinují budoucí chování knihovny a tím i možnost nasazení.

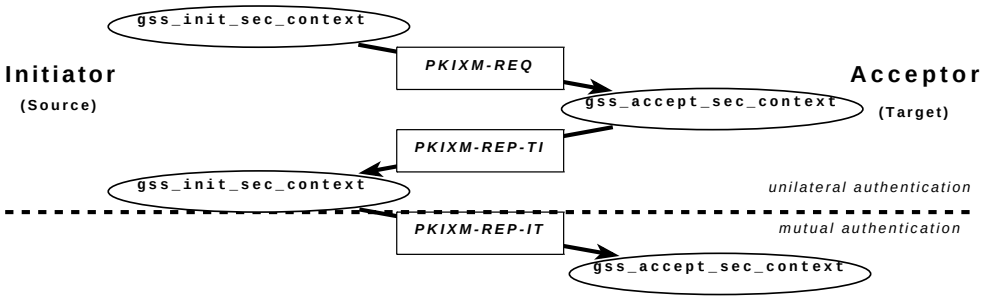
Při volbě mechanismu jsme chtěli použít jedno ze standardizovaných řešení. Bohužel žádný ze známých mechanismů nevyhovoval našemu záměru. Od Kerberos nás odradila nutnost existence centrálního autentizačního serveru. U SPKM by X.509 certifikáty vyhovovaly, ale neakceptovatelná byla zastaralost dokumentu a hlavně chyba ve způsobu generování sub-klíčů. **Nakonec jsme byli nuceni vytvořit vlastní návrh mechanismu - X.509 Public Key Infastructure Mechanism (PKIXM)**.

Náš PKIXM mechanismus, přesto že je v tuto chvíli proprietárním řešením, byl v mnoha ohledech navrhován šířeji, než vyžadovala naše implementace, a to právě s výhledem do budoucna k možné standardizaci a následnému širšímu užití.

Dále v této části rozebereme důležité vlastnosti PKIXM mechanismu. Kompletní popis PKIXM mechanismu včetně ASN.1 definic jednotlivých tokenů zpráv je přiložen jako dodatek A.

4.1 Autentizace

Pro autentizaci bylo rozhodnuto použít asymetrickou kryptografii a X.509 certifikáty. Certifikáty a s nimi spojené PKI spolu s důvody, které nás vedly k jejich zvolení, jsou popsány v části 5.1. PKIXM mechanismus umožňuje jednostrannou i oboustrannou autentizaci (viz obrázek 4.1). U jednostranné autentizace se používá dvojfázový handshake, u oboustranné pak třífázový handshake. Proti replay útokům se autentizace zabezpečuje ochranou náhodnými čísly jak je popsáno v dokumentu [8] části 10.



Obrázek 4.1: Diagram autentizace a zřizování kontextu v PKIXM. Přerušovaná čára nám určuje hranici mezi jednostrannou a oboustrannou autentizací.

4.2 Použitá kryptografie

V PKIXM jsou definovány čtyři množiny algoritmů: S-APG, K-ALG, Sym-ALG a MD-ALG.

Množina S-ALG je množina kryptografických podpisových algoritmů. Kryptografický podpisový algoritmus je používán v procesu autentizace k svázání hodnoty integritního součtu autentizačních tokenů s autentizovanou osobou.

Množina K-ALG udává množinu algoritmů, které mohou být použity při procesu výměny kontextového klíče.

Sym-ALG je množina symetrických blokových šifrovacích algoritmů a MD-ALG je množina hešovacích algoritmů. Ty se podílejí na zajištění utajení a integrity jednotlivých zpráv vytvářených GSS-API voláními `gss_getMIC()`a `gss_wrap()`. Prvky množin daného kontextu si obě strany volí při jeho zřizování.

Algoritmus z množiny S-ALG je determinován použitým X.509 certifikátem. Na algoritmu z K-ALG a na množinách Sym-ALG a MD-ALG se obě strany komunikace domlouvají v průběhu zřizovaní kontextu. Jedna ze stran si zvolí prvky množin a druhá si z nich pak vybere ty, které jí vyhovují. Dohodnuté množiny Sym-ALG a MD-ALG jsou pak součástí kontextu.

4.3 Quality of Protection

Parametr QoP je jedním ze vstupů GSS-API volání `gss_getMIC()` a `gss_wrap()` a dovoluje volajícímu si zvolit úroveň zabezpečení, ať už kontroly integrity zpráv nebo jejich utajení.

V PKIXM mechanismu je 32 bitů QoP definováno tak, že horních 16 bitů udává pořadí algoritmu symetrické šifry v množině Sym-ALG daného kontextu a dolních 16 bitů udává pořadí hešovacího algoritmu v množině MD-ALG. Implicitní hodnota 0, jak v horní tak v dolní části parametru, znamená první algoritmus z dané skupiny a naopak hodnota přesahující počet algoritmů v některé ze skupin znamená poslední algoritmus.

4.4 Dohoda na kontextovém klíči

Kontextový klíč je užíván ke generování sub-klíčů vstupujících do jednotlivých symetrických šifrovacích algoritmů ze Sym-ALG. Jeho délka by měla být zdola omezena nejdelším klíčem z Sym-ALG a zároveň shora omezena maximální velikostí dat použitelných s algoritmem z K-ALG. Dohodnutý algoritmus z K-ALG pak určuje i způsob dohody klíče. Vzhledem k tomu, že dohoda na klíči se provádí během X.509 autentizace, je i takto vzniklý sdílený kontextový klíč autentizován.

4.5 Generování sub-klíčů a inicializačních vektorů

V tomto jsme se rozhodli poučit z chyby při návrhu SPKM, kde pro daný kontext a daný algoritmus jsou sub-klíče symetrických šifer vždy stejné. Tento problém jsme vyřešili přidáním čítače pořadového čísla zprávy do procesu generování sub-klíčů. Tím je zajištěno, že se sub-klíče s velkou periodou obměňují.

Podobně předcházíme i potencionálním útokům proti schématům blokových šifer a jejich paddingu generováním vždy "nového" inicializačního vektoru symetrické šifry.

Sub-klíč nebo inicializační vektor délky k-bitů pak vznikne jako nejlevějších k-bitů zřetězení řetězců S_1 až S_n, kde

$S_i = OWF(context\_key|type|i|seq\_num|context\_key)$

- OWF znamená jeden z algoritmů z množiny MD-ALG
- *context_key* je kontextový klíč
- *type* je typ dat ("INTEGRITY_KEY", "INTEGRITY_IV", "CONFIDENTALITY_KEY", "CONFIDENTALITY_IV")
- i je číslo řetězce

- *seq_num* je sekvenční číslo dané zprávy
- | znamená zřetězení

4.6 Komunikační tokeny

K tomu, aby se zajistila vzájemná interoperabilita různých implementaci stejného mechanismu, slouží definice komunikačních tokenů. Ty jsou v podstatě použity jako interface GSS-API zcela nezávislý na platformě či programovacím jazyku.

V PKIXM používáme pro definici svých tokenů standard ASN.1[1]. Tokeny jsou pak kódovány DER[2] pravidly.

Mechanismus definuje tři typy tokenů:

- tokeny na zřizování kontextu
- tokeny jednotlivých zpráv
- tokeny exportující kontext

Tokeny na zřizování kontextu slouží k navázání komunikace dvou stran. Jejich prostřednictvím se jedna nebo obě strany autentizují (obrázek 4.1), dohodnou se na používaných algoritmech v množinách K-ALG, Sym-ALG a MD-ALG, vytvoří si společný kontextový klíč a všechny ostatní parametry kontextu.

Tokeny `PKIXM-REQ`, `PKIXM-REP-TI` a `PKIXM-REP-IT` jsou používány voláními `gss_init_sec_context()` a `gss_accept_sec_context()` (viz. obrázek 4.1).

Tokeny jednotlivých zpráv slouží k zabezpečení komunikace. Ty jsou hlavním nositelem funkcionality GSS-API.

Tokeny `PKIXM-MIC` jsou výstupem volání `gss_getMIC()` a vstupem volání `gss_verifyMIC()`. Jejich obsahem je mimo jiné integritní součet chráněných dat, data však nikoliv. Oproti tomu tokeny `PKIXM-WRAP` (volání `gss_wrap()`/`gss_unwrap()`), zajišťují integritu, a případně i utajení, dat v nich přenášených.

Tokeny exportující kontext se od těch ostatních liší. Neslouží ke komunikaci dvou protistran, ale jejich úkolem je naplnit možnost GSS-API exportovat své kontexty a přenášet je tak mezi různými procesy za účelem balancování výkonu.

[1] ASN.1 = Abastract Syntax Notation One
[2] DER = Distinguished Encoding Rules

Tokeny `PKIXM-EXPORT` jsou obsluhovány dvojicí GSS-API volání `gss_export_sec_context()` a `gss_import_sec_context()`.

Kapitola 5

Zvolené standardy a technologie

V této části se podíváme na námi zvolené standardy a technologie. Při výběru jsme se snažili brát v úvahu rozšířenost užití, která naznačuje jejich ověření praxí. Dále jsme upřednostňovali dobrou podporu a dokumentaci. Důležitým kritériem byla i otevřenost standardu či technologie.

5.1 PKI (X.509)

Public Key Infrastucture (infrastruktura veřejných klíčů) je systém, který spojuje veřejný klíč a příslušnou uživatelovu identitu za pomocí certifikační autority (CA). Touto vazbou je X.509 certifikát [11].

Aranžmá PKI umožňuje uživateli autentizaci, ověření podpisu nebo obecně jakoukoli kryptografii s veřejným klíčem protějšku komunikace, bez nutnosti znát cokoliv o něm. K tomu stačí ověřit důvěryhodnost jeho certifikátu. Důvěryhodnost garantuje podpis CA, která certifikát vystavila. K podpisu byl však použit soukromý klíč patřící k certifikátu CA. Ten byl vystaven, a tím i ověřen, jinou nadřazenou CA. Takto vytváříme řetěz důvěry od ověřovaného certifikátu až ke kořenovému certifikátu, kterému důvěřujeme.

PKI vzniklo koncem 80-tých let 20. století na základě rodiny protokolů X.500. Z nich byl pro PKI nejdůležitější ITU-T Recommendation X.509 — ISO/IEC 9594-8 [8], který rozvinul až do své současné podoby [9]. Systém X.500 nebyl nikdy plně implementován a tak v roce 1995 vznikl v rámci IETF[1] pracovní skupina PKIX[2][10], která má za úkol přizpůsobit standard

[1]IETF = Internet Engineering Task Force
[2]PKIX = Public-Key Infrastructure (X.509)

X.509 použití v prostředí internetu. V současnosti je běžné považovat pojem "X.509 certifikát" jako odkaz na dokument pracovní skupiny PKIX RFC 3280 [11].

Struktura X.509 certifikátu verze 3 vypadá takto:

- Certificate
  - Version – verze certifikátu
  - Serial Number – jedinečné číslo v rámci CA
  - Algorithm ID – totožný s Certificate Signature Algorithm
  - Issuer – identifikace vystavitele certifikátu (Distinguished Name)
  - Validity – platnost certifikátu
    - Not Before – od
    - Not After – do
  - Subject – identifikace držitele certifikátu (Distinguished Name)
  - Subject Public Key Info
    - Public Key Algorithm – identifikátor algoritmu veřejného klíče
    - Subject Public Key – veřejný klíč držitele
  - Issuer Unique Identifier (možnost) – v internetovém prostředí se nedoporučuje používat
  - Subject Unique Identifier (možnost) – v internetovém prostředí se nedoporučuje používat
  - Extensions (možnost) – pro naše potřeby nejdůležitější rozšíření jsou:
    - Key Usage – možnosti použití klíče
    - CRL Distribution Point – místo, odkud lze získat CRL
- Certificate Signature Algorithm – algoritmus, kterým byl certifikát podepsán
- Certificate Signature – podpis vystavitele

Unique Identifiery znamenají použití certifikátu verze 2 nebo 3, Extensions pak verzi 3.

Certifikát může přestat platit, i před vypršením doby platnosti, tzv. zneplatněním. CA (respektive její část RA[3]) proto vydává seznam zneplatněných

[3]RA = Revocation Authority

certifikátů - CRL[4]. Při ověřování certifikátů se kromě podpisu vystavitele a platnosti ostatních parametrů kontroluje jeho zneplatnění a přítomnost v příslušném CRL.

V implementaci jsme PKI zvolili proto, že nám přináší velkou úroveň škálovatelnosti. Dovoluje naši knihovnu nasadit v malých systémech, kde si vystačí s vlastní malou CA o jednoúrovňovém řetězu důvěry a jednom CRL, ale také v rozsáhlých systémech, které již mají implementováno PKI (ať už vlastní nebo třetích stran).

5.2 ASN.1

V telekomunikacích a počítačových sítích je Abastract Syntax Notation One standardem a flexibilní notací, která popisuje datové struktury pro reprezentaci, enkódování, přenos a dekódování dat. ASN.1 poskytuje množinu formálních pravidel, popisujících strukturu objektů, která jsou nezávislá na enkódování specifickém pro daný stroj a jsou precizní formální notací odstraňující dvojznačnosti.

Dnešní ASN.1 standard vznikl spojením ISO[5] a ITU-I[6] standardu. Poprvé se ASN.1 objevilo v roce 1984 jako součást standardu X.409. Díky jeho širokému rozšíření se v roce 1988 osamostatnil do ITU-T Recommendation X.208 a X.209 jako součást OSI[7] modelu. Dále následoval bouřlivý vývoj až k dnešním sériím standardů ITU-T Rec. X.680 — ISO/IEC 8824 [13 – 16] a ITU-T Rec. X.690 — ISO/IEC 8825 [17 – 21]. Přehledně je průběh standardizace popsán na stránkách ASN.1 Konsorcia [12].

ASN.1 definuje abstraktní syntaxi informace, ale nijak neomezuje, jak bude informace enkódována. Toto je úkolem enkodovacích pravidel. Různá ASN.1 enkódovací pravidla poskytují přenosovou syntaxi (konkrétní reprezentaci) dat, která je popsána abstraktní syntaxí v ASN.1.

Nejznámější jsou tyto standardy ASN.1 enkódovacích pravidel:

- Basic Encoding Rules (BER) [17]
- Canonical Encoding Rules (CER) [17]
- Distinguished Encoding Rules (DER) [17]

[4]CRL = Certificate Revocation List

[5]ISO = International Organisation for Standardization

[6]ITU-T = International Telecomunication Union, Telecomunication Standardization Sector

[7]OSI = Open System Interconnection

- Packed Encoding Rules (PER) [18]
- XML encoding rules (XER) [20]
- mapping XML Schema Definition (XSD mapping) [21]
- Generic String Encoding Rules (GSER) [22]

Pro nás jsou nejpodstatnější enkódovací pravidla BER a DER.

BER představuje sebe popisující a sebe oddělující formát enkódování ASN.1 datových struktur. Každý datový element je enkódován jako trojice: identifikátor typu, délka a datový element. Proto je nazýván tzv. TLV[8] enkódováním. Takto koncipované formátovaní umožňuje dekódovat ASN.1 informace i z nekompletního streamu bez nutnosti předem znát velikost, obsah nebo sémantiku přenášených dat.

DER je pak definovanou podmnožinou BER. BER totiž umožňuje sémanticky totožná data enkódovat syntakticky různým způsobem. Typickým příkladem je možnost enkódovat "TRUE" hodnotu boolean v BER až 255 možnostmi. Toto je nepřípustné u kryptograficky podepisovaných dat, kde je vyžadováno, aby data byla interpretována na obou stranách komunikace stejně. A právě tuto nejednoznačnost odstraňují DER pravidla.

Pro naši implementaci se stal standard ASN.1 a jeho DER formátování jednoznačným řešením. Zapříčinilo to použití standardu v definici formátu tokenu nezávislého na mechanismu v GSS-API standardu [1] i v definicích X.509 certifikátu a CRL [11].

Pro práci s ASN.1 byl zvolen `asn1c` kompilátor [23]. Tento open source projekt vyvíjí midleware, který z ASN.1 definic datových struktur vytváří skeletony struktůr v programovacím jazyce C. Z a do těchto skeletonu se pak provádí enkódovaní/dekódování přenášených dat. Podporovány jsou formáty BER, DER, PER, XER a další se vyvíjejí.

Pro nás podstatnou vlastností asn1c kompilátoru je podpora různých verzí ASN.1 standardu. Jelikož jsou ASN.1 definice tokenů z GSS-API, X.509 certifikátů i CRL navrženy podle standardu ASN.1 1988, i my tento standard přebíráme do našich ASN.1 definic. Vystupující enkódovaná data jsou však s nejnovějšími standardy kompatibilní.

5.3 OpenSSL

OpenSSL [24] je open source projekt, jehož cílem je vytvořit otevřenou bezpečnostní knihovnu. Projekt prochází aktivním vývojem a již několik let

[8] TLV = Type Length Value

je hojně užíván celou řadou jiných open source aplikací. Asi nejznámějším příkladem užití je SSL[9] podpora webovému serveru Apache.

OpenSSL projekt se skládá ze tří hlavních částí:

`crypto` – knihovna, která implementuje velké množství kryptografických algoritmů

`openssl` – command-lineovou aplikací, která dovoluje používat funkce OpenSSL `crypto` knihovny z shellu

`ssl` – knihovna obsahující API, které implementuje SSL v2/v3 a TLS[10] v1

Z knihovny `crypto` implementace využívá tuto funkcionalitu:

- práce s OBJECT IDENTIFIERy [11]
- ověřování X.509 certifikátů a CRL
- přímé asymetrické šifrování
- symetrické šifrování
- hešovací funkce
- vytváření a ověřování podpisů
- generovaní pseudonáhodných dat

Knihovna `crypto` obsahuje dva typy volání kryptografických funkcí - abstraktní volaní a nízko-úrovňová volání. Abstraktní volání umožňuje parametrizovat daný typ kryptografické operace a tím ovlivnit, která nízkoúrovňová funkce bude volána. Parametrem je OBJECT IDENTIFIET daného algoritmu. Toto zajišťuje možnost konfigurovat množiny používaných algoritmů a nastavit QoP[12]. Užití abstraktních volání je tvůrci knihovny doporučeno.

Bohužel ne všechny nízkoúrovňové funkce lze volat abstraktně. Proto pro asymetrickou kryptografii (podepisování i šifrování) lze používat jen algoritmus RSA.

Aplikace `openssl` je užitečná při stavbě malého PKI k vygenerování CA a uživatelských certifikátů, soukromých klíčů a CRL.

OpenSSL je nutné používat od verze 0.9.8, neboť až od této verze je v procesu verifikace certifikátu dovoleno používat jiné CRL, než jaká jsou uložena v lokálním uložišti.

[9] SSL = Secure Socket Layer
[10] TLS = Transport Layer Security
[11] OBJECT IDENTIFIER je speciální ASN.1 datový typ
[12] QoP = Quality of Protection viz sekce 4.3

Kapitola 6

Návrh řešení

V této kapitole se seznámíme s návrhem implementace GSS-API knihovny. V první části se budeme zabývat vlastnostmi, které má nová knihovna. Ve druhé části se podíváme na některé komplikace, které bylo třeba během implementace řešit. Ve třetí části budou představeny aplikace, které pomáhají knihovnu spravovat.

6.1 Vlastnosti knihovny

V této sekci budou popsány vlastnosti námi vytvořené GSS-API knihovny.

6.1.1 Mechanismus

Knihovna používá jen jeden mechanismus. Tím je nově navržený PKIXM.

Volba kryptografických algoritmů a s ní související Qualita of Protection je úzce spojená s volbou konkrétního kryptografického API. V našem případě se pak jedná o knihovnu `crypto` z OpenSSL. Ta umožňuje abstraktně volat v asymetrických šifrovacích a podpisových schématech jen algoritmus RSA. Jako symetrické blokové šifry nebo hešovací funkce si lze vybrat libovolný algoritmus podporovaný v OpenSSL. Ze symetrických blokových šifer jsou doporučovány varianty AES v CBC módu a z hešovacích funkcí některá z rodiny SHA. Uživatel svou volbu provede v konfiguračním souboru.

Vzhledem k předpokládanému nasazení i obecným doporučením je použita oboustranná autentizace. Implementace užívá jednoho X.509 certifikátu [11] obsahujícího RSA veřejný klíč pro autentizaci i k výměně kontextového klíče. Certifikát by měl mít v rozšíření KeyUsage flagy `digitalSignature` a `keyEncipherment`, a naopak je nevhodné užití flagu `nonRepudiation` (nejedná se tedy o kvalifikované certifikáty).

Tokeny zpráv jsou kódovány pomocí ASN.1 DER pravidel.

6.1.2 Jméno

GSS-API obecně umožňuje, aby struktura obsahující jméno, byla složena z více částí, každé příslušné některému z prostoru jmen. V implementaci používáme jen jeden prostor jmen - prostor Distinguished Names (DN).

DN používáme proto, neboť jako vazbu mezi credenciálem a jménem jsme zvolili Subject X.509 certifikátu, který je definován jako DN. Ve všech případech, kde se předpokládá jméno v podobě řetězce (volání `gss_dispaly_name`, `gss_export_name`, `gss_import_name` a v konfigiraci) se používá LDAP v3 UTF-8 řetězcová reprezentace DN[25].

Implicitní jméno, je-li požadováno, může být nastaveno v konfiguračním souboru. Naopak jméno pro anonymní entitu neexistuje, neboť je to neslučitelné s filozofií standardu X.509.

6.1.3 Credenciály

Credenciály jsou data, která příslušné mechanismy potřebují k autentizaci a zřízení kontextu. V případě našeho PKIXM jsou těmito daty X.509 certifikát a jemu příslušný soukromý klíč.

Implicitní credenciály odpovídají implicitnímu jménu. Není-li implicitní jméno definováno v konfiguračním souboru, implicitní credenciály neexistují.

6.1.4 Kontext

Kontext jsou data, která potřebují obě strany k úspěšné vzájemné komunikaci. V našem případě jsou to hlavně množiny symetrických šifrovacích algoritmů a hešovacích algoritmů, kontextový klíč, GSS-API flagy a pořadová čísla. Podrobněji je tato problematika vysvětlena v popisu PKIXM výše nebo v jeho definici v dodatku A.

6.1.5 Sequencing a Replay detection

Další bezpečnostní opatření, které GSS-API může poskytovat jsou ochrana sequencingu a replay detection.

Sequencing se snaží aplikaci informovat, jsou-li zprávy doručovány ve správném pořadí

Reply detection má na starost informovat o duplikátních zprávách

Obě tyto ochrany jsou naší implementací použity.

6.1.6 Channel binding

Channel binding umožňuje aplikacím svázat komunikační kanál s kontextem. Naše knihovna k tomu užívá hodnoty vypočítané jako SHA1 heš ze zřetězených hodnot struktury channel binding v "net orderu".

6.2 Problémy a jejich řešení

Během implementace vyvstaly některé podstatné otázky. V této části se je pokusíme diskutovat a navrhnout jejich řešení.

6.2.1 Inicializace a konfigurace knihovny

GSS-API ve svých voláních neobsahuje žádné volání, které by umožňovalo inicializaci a konfiguraci nebo finalizaci naší knihovny. Ta však vyžaduje inicializaci (a s ní i konfiguraci) před prvním voláním GSS-API funkcí, aby bylo možno inicializovat generátor pseudonáhodných čísel a načíst potřebná data do vnitřních struktur knihovny. Tento problém řešíme použitím primitiv jazyka C

```
void __attribute__ ((constructor)) init(void)
```

a

```
void __attribute__ ((destructor)) fini(void),
```

které se volají při načítání nebo uvolňování dynamické knihovny.

Podobným problémem bylo, jak knihovně předávat konfigurační data. Konfigurace samotná se provádí v konfiguračním souboru. Jeho umístění je předáno knihovně v proměnné prostředí `PKIXM_CONF_FILE`.

6.2.2 Chybové hlášení

GSS-API volání používají pro předávaní informace o chybách dvě návratové hodnoty: major status code a minor status code. Major status code vrací hodnoty definované přímo GSS-API standardem[1], zatímco minor status code obsahuje hodnoty specifické pro daný mechanismus. Obě tyto hodnoty tak mohou být voláním `gss_display_status` převedeny do textové podoby.

Zabývali jsme se otázkou, jak podrobně referovat vnitřní chyby v kódech našeho mechanismu. Může totiž nastat situace, kdy díky špatné implementaci aplikace, která naši knihovnu používá, bude naše velká detailnost zneužita k postrannímu útoku. Proto jsme preventivně konzervativní a v místech, kde by útok mohl hrozit, jsme v chybových hlášeních spíše strozí. To je bohužel vykoupeno menším uživatelským komfortem při odhalovaní chyb.

6.2.3 Reprezentace Object Identifieru

V naší implementaci GSS-API knihovny se objevují čtyři různé reprezentace ASN.1 struktury OBJECT IDENTIFIER:

- struktury `gss_oid_desc` a `gss_oid` ve standardu C-bindings k GSS-API [2]
- struktury `OBJECT_IDENTIFIER` vygenerované `asn1c` kompilátorem
- struktury `ASN1_OBJECT` a její pořadí `nid` v OpenSSL
- lidsky čitelná textová reprezentace tečkovou notací v konfiguraci

Jako interní reprezentaci byl `nid` z OpenSSL, neboť tuto reprezentaci nejčastěji používáme při parametrizaci kryptografických volání a tato knihovna poskytuje i dobrou správu těchto struktur. Z a do ostatních reprezentací Object Identifieru převádíme jen v jim příslušných částech implementace.

6.2.4 Stahování CRL

Jak bylo objasněno v části 5.1, k ověřování X.509 certifikátů jsou zapotřebí certifikáty CA a příslušné CRL. U certifikátu se doba platnosti počítá v měsících či letech, takže jejich management je otázka uživatelské konfigurace, u CRL se setkáváme i s platností v řádu hodin. Zde již je zapotřebí jistá automatizace.

Knihovna OpenSSL tuto stahování CRL neřeší. Pro ověření certifikátu předpokládá přítomnost dat k tomu potřebných ve svých uložištích. Byli jsme tedy donuceni tuto funkcionalitu doimplementovat sami.

Informace o adrese nového CRL získáme z rozšíření CRL Distribution Point, které obsahuje adresu ve formátu URI[1]. CRL pak z této adresy stáhneme a uložíme do našeho uložiště.

Pokud uživatelé mají ke svým certifikátů rychle expirující CRL, měli by zajistit, aby jejich certifikáty obsahovaly toto rozšíření. Podporovány jsou přenosové protokoly http[2], ftp[3] a ldap[4].

Uvědomujeme si, že existují i další možnosti jak získat informace o revokovaných certifikátech jako je OCSP[5]. Zvolená možnost je základní a tudíž by měla být podporována všemi komerčními CA.

[1] URI = Uniform Resource Identifier
[2] HTTP = Hypertext Transfer Protocol
[3] FTP = File Transfer Protocol
[4] LDAP = Lightweight Directory Access Protocol
[5] OCSP = Online Certificate Status Protocol

Konfigurace umožňuje toto stahování vypnout a hledat nové CRL jen v lokálním uložišti. Tato vlastnost je implementována s ohledem na nasazení naší knihovny do prostředí, kde se nové CRL získávají jinou cestou (např. jinou službou).

6.3 Pomocné aplikace

Pro snadnější management knihovny a k ní příslušných dat jsme vytvořili dvě aplikace `snc_test_config` a `snc_gen_cred`.

6.3.1 Aplikace `snc_test_config`

Funkcionalita chybových hlášení je spjata až s GSS-API voláními. Tudíž během inicializace knihovny i načítání konfiguračního souboru je neaktivní. Aplikace `snc_test_config` má za úkol otestovat správnost konfiguračního souboru a předejít pádu knihovny z důvodu konfigurace, kterou by nebylo možno detekovat.

6.3.2 Aplikace `snc_gen_cred`

Aplikace `snc_gen_cred` umožňuje uživateli uložit certifikáty, CRL a soukromé klíče do uložišť definovaných v konfiguračním souboru.

V případě uživatelských certifikátů ukazuje i korektní textový formát Distinguished Name položky Subject (viz sekce 6.1.2).

Kapitola 7

Testy a srovnání

V této kapitole bude prokázána korektnost implementace naší GSS-API knihovny. S testy se začalo již v průběhu práce, kdy se kontrolovala funkčnost jednotlivých implementovaných komponent.

V prvním testu byla použita k ověření naší knihovny testovací utilita `gsstest` navržená Martinem Rexem [28]. Program byl vytvořen právě ke kontrole korektnosti GSS-API knihoven.

Ve druhé testu byla kontrolována naše knihovnu přímo při zabezpečování komunikace. V době dokončování naší implementace již bohužel nebyl k dispozici funkční SAP server. Jeho chování po stránce komunikace bylo simulováno `saproutery`. `Saproutery` byly dobrou náhradou, neboť podporují stejné komunikační funkce včetně zabezpečení komunikace.

Třetí test probíhal podle stejného scénáře jako test druhý. Tento test byl zaměřen na stabilitu knihovny pod déletrvající velkou zátěží.

Čtvrtou částí je srovnání výkonu naší knihovny a komerční knihovny firmy SECUDE.

Pro účely těchto testů byla aplikací `openssl` vytvořena vlastní CA:

- selfsigned CA X.509 certifikát a jemu příslušný RSA soukromý klíč
- 2 uživatelské X.509 certifikáty a 2 RSA soukromé klíče
- CRL

CA certifikát, oba uživatelské certifikáty a jejich soukromé klíče a CRL byly aplikace `snc_gen_cred` zapsány do odpovídajících uložišť knihovny.

7.1 Test – `gsstest`

7.1.1 Cíl

Účelem tohoto testu bylo prokázat dodržení GSS-API standardu námi implementovanou knihovnou.

7.1.2 Podmínky

Test probíhal tak, že utilita `gsstest` dynamicky načetla kontrolovanou GSS-API knihovnu a pak na ní interně prováděla sérii testů.

Během testů byly využívány dvě entity. Identifikátor jedné byl načten z příkazové řádky jako jeden z parametrů, druhou entitou byla implicitní entita. Její identifikátor byl definován v konfiguračním souboru.

7.1.3 Výsledky

Během testu bylo úspěšně provedeno:

- 690x import jmen
- 920x export jmen
- 152x vytvoření credenciálů
- 52x zřízen kontext
- 303x exportován a re-importován kontext
- 3079x použita ochrana zpráv
  - 1048x `gss_getMic`/`gss_verifyMic`
  - 883x `gss_wrap`/`gss_unwrap` (pouze integrita)
  - 1148x `gss_wrap`/`gss_unwrap` (integrita i utajení)

Všechny dílčí testy byly úspěšně provedeny s konečným výsledkem **"Passing all API result tests."**.

7.1.4 Závěr

Testem bylo prokázáno, že naše knihovna dodržuje všechny požadavky, které si klade GSS-API standard.

Tento test v sobě rovněž vyvolává i možné chyby a kontroluje, jak na ně testované knihovny reagují. Můžeme konstatovat, že i po této stránce naše knihovna reaguje korektně.

7.2 Krátkodobý test – saproutery

Aplikace saprouter slouží k propojování klientských sítí se sítěmi, v nichž má klient SAP server umístěn. V testu byla využívána schopnost sapserverů mezi sebou komunikaci zabezpečit použitím GSS-API knihoven.

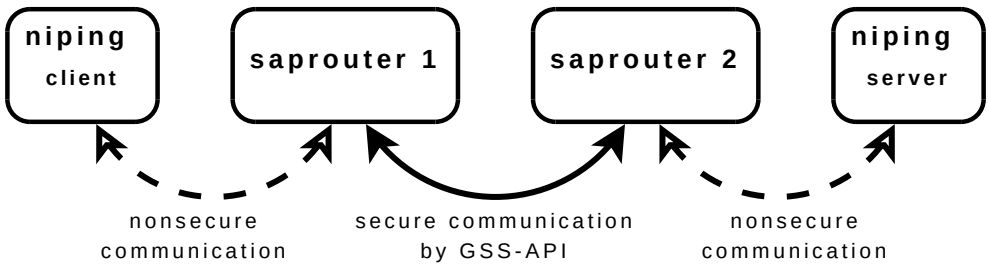
Aplikace niping je obdobou běžně používaného pingu a slouží jako test komunikace sapserverů.

7.2.1 Cíl

Cílem tohoto testu bylo prokázat možnost aplikačního nasazení námi implementované GSS-API knihovny.

7.2.2 Podmínky

V testu byly použity dva saproutery, niping klient a niping server. Oba sapservery byly nastaveny tak, aby komunikace mezi nimi byla zabezpečena naší GSS-API knihovnou. K tomu každý z nich používal jednu z námi připravených entit. Niping klient pak komunikoval s niping serverem pravě přes saproutery. Toto názorně zobrazuje obrázek 7.1.



Obrázek 7.1: Schéma testu s použitím dvou sapserveru, niping klienta a niping serveru.

Saproutery spolupracují jen s určitou množinou knihoven. Naše knihovna však zatím podporována není a tak k provedení testu byla upravena, aby se představovala jako jedna z podporovaných knihoven.

7.2.3 Výsledky

Výsledkem testu je konstatování, že všechny pingy, které byly niping klientem odeslány, námi předepsanou trasou k niping serveru dorazily a odpovědi na ně se vrátily korektně zpět.

7.2.4 Závěr

Test prokázal způsobilost naší knihovny k nasazení v praxi. Přesto, že nebylo možno otestovat knihovnu přímo při komunikaci s aplikačním serverem SAP, provedený test na `saproutrech`, které využívají podobně služeb knihovny, dává implikuje použitelnost i se servery SAP.

7.3 Dlouhodobý test – `saproutery`

7.3.1 Cíl

Cílem testu bylo ověřit korektnost chování knihovny pod zátěží v delším časovém období.

7.3.2 Podmínky

Test probíhal obdobně jako v druhém testu za pomocí dvou `saprouterů`, `niping klienta` a `niping serveru`. Spouštěcí parametry `niping klienta` byly nastaveny na generování dávek o tisíci pingů, každý o velikosti 1000 B. Těchto dávek bylo vygenerováno tisíc.

Během testu byly zkoumány propustnost komunikační cesty a správa paměti jednotlivých procesů.

7.3.3 Výsledky

V průběhu testu se hodnoty průměrné propustnosti dávek pingů nijak významně nelišily. Rovněž čas zpracování jedné dávky zůstával stejný. Oba `saproutery` si na začátku naalokovaly potřebnou paměť a dále již setrvaly na stejné úrovni.

7.3.4 Závěr

Testem byla prokázána dlouhodobá stabilita naší knihovny. Dále bylo otestováno korektní nakládání s pamětí.

7.4 Srovnání s knihovnou firmy SECUDE

7.4.1 Cíl

Cílem bylo výkonnostní srovnání naší knihovny a komerční knihovny.

7.4.2 Podmínky

Pro srovnáni byla zvolena knihovna firmy SECUDE International AG, která je dlouholetým partnerem firmy SAP v oblasti bezpečnosti. Jejich knihovna využívá vlastní proprietární mechanismus postavený na X.509 certifikátech, obdobně jako naše knihovna.

Test probíhal opět za pomocí dvou `saprouterů`, `niping klienta` a `niping serveru`. Bylo generováno 25 dávek o tisíci pingů. Velikost pingů byla nastavena na 10 kB, aby se více projevila síla kryptografických motorů obou knihoven.

Obě knihovny byly nasazeny za schodných podmínek a sledovala se datová propustnost a doba zpracování jednotlivých dávek pingů.

7.4.3 Výsledky

Naše knihovna dosahovala o polovinu větší datovou propustnost a o třetinu kratší časy než konkurenční knihovna.

7.4.4 Závěr

Výsledky nijak nepřeceňujeme, neboť mohou být ovlivněny prostředím, ve kterém se test odehrával. Ale určitě vypovídají o tom, že zvolení OpenSSL jako kryptografického motory bylo dobrou volbou.

Kapitola 8

Pokračování práce

Uvědomujeme si, že práce na IT projektech nikdy nekončí. Náš projekt má tři zdroje inspirace k dalšímu vývoji:

- standardizace
- technologie
- ostatní

8.1 Standardizace

Velkým námětem budoucí práce mohou být snahy vyhovět všem standardům a doporučením.

V první řade je nutno sledovat, zda se neobjeví nová verze mechanismu SPKM [6] nebo některý jiný mechanismus založený na PKI. To by nás donutilo tento nový mechanismus přidat do naší knihovny, neboť je větší šance na interoperabilitu naší knihovny implementací mechanismu podporovaného známými osobnostmi internetu a velkými firmami, než šance, že by se podařilo výrazně prosadit náš návrh mechanismu.

S předchozím odstavcem pak může souviset i nutnost implementovat pseudo-mechanismus SPNEGO [26]. Tento standard se snaží zvýšit možnost vzájemné kooperace různých knihoven přidáním možnosti volby používaného mechanismu do procesu zřizování kontextu.

Dále je třeba sledovat i diskuze nad otázkami, které které byly řešeny i při naší implementaci, a být připraven akceptovat a zapracovat z nich vzniklá doporučení. Jednou z nich je debata nad vazbou mezi jmény a credenciály (v našem případě X.509 certifikáty), kterou se od prosince 2006 pokouší rozpoutat informační dokument S. Hartmana [27].

V neposlední řadě nutnost změn přinese i případné vydaní GSS-API verze 3 a všech bindingů programovacích jazyků.

8.2 Technologie

Změnami v této části rozumíme změny spojené s vývojem v použitých cizích knihovnách. I u těchto zásahů bude jejich prapůvod souvislost s vývojem standardu nebo standardizačních doporučení, ale pro nás to bude znamenat zásah v oblasti implementovaných technologii.

Použití abstraktních volání kryptografických funkcí z knihovny OpenSSL by mohlo knihovnu ochránit před případnými změnami v kryptografických algoritmech (např. hešovací funkce jsou na prahu revoluce zapříčiněné útoky, které na ně byly v posledních letech objeveny). Jednoduše se použije aktuální verze OpenSSL, která bude obsahovat aktuální algoritmy. Ty se pak nastaví v konfiguraci naší knihovny. Pokud by však došlo k prolomení bezpečnosti RSA (a kvantové počítače tím hrozí) a neexistovala-li by náhrada v asymetrickém algoritmu, který by byl použitelný jak v šifrovacích tak v podpisových schématech, bylo by třeba používat na tyto dvě kryptografické operace dvou různých algoritmů a tím i dvou různých X.509 certifikátů. Návrh PKIXM mechanismu sice s touto variantou počítá, ale implementace pro jednodušší nasazení a správu používá certifikát jeden. Tato úprava by nebyla náročná a zabrala by jen několik málo hodin.

Další možný zásah v oblasti OpenSSL by vyvolal požadavek na použití speciálního kryptografického hardwaru. Vznikla by nutnost přizpůsobit tomuto požadavku i naši implementaci.

Podobně může nastat problém v ASN.1. Momentálně používáme ASN.1 definice podle standardu roku 1988, abychom zachovali kompatibilitu s ASN.1 definicemi z jiných standardů. Pokud u nich dojde v tomto směru k aktualizaci, budeme muset tento krok následovat a ASN.1 definice PKIXM změnit. Ale tím se změní i struktury vygenerované `asn1c` kompilátorem.

8.3 Ostatní

Dále by se mohla ze strany budoucích uživatelů objevit žádost o další pomocné aplikace, které by pomohly při správě systému. Rovněž je možno vytvořit ovládání těchto aplikací pomocí přívětivějšího uživatelského GUI[1].

[1] GUI = Grafic User Interface

Kapitola 9

Závěr

Hlavní cíl diplomové práce tj. vytvořit GSS-API knihovnu postavenou na základech PKI s využitím OpenSSL byl splněn. Rovněž byla prokázána použitelnost naší knihovny v aplikacích firmy SAP. Během práce musel být vytvořen a implementován vlastní GSS-API mechanismus, neboť se nepodařilo najít žádný, který by vyhovoval našim požadavkům.

Práce na knihovně nás dostala do blízkosti standardů a technologii, které se hojně používají v dnešním světě IT. Z nich byly vybrány ty, které zaručují otevřenost a přenositelnost implementace.

Při vývoji se narazilo na dva protipóly současného programování. Na jedné straně byly uzavřené firmy jako např. SAP, kde se díky nedostatku informací práce dostávala až na hranice reverzního inženýrství. Na straně druhé to byly open source projekty a jejich překotný vývoj takřka pod rukama, občas vyvolaný i námi objevenými chybami. S oběma rozdílnými póly se nám podařilo úspěšně vyrovnat a práce byla dovedena ke zdárnému konci.

Dodatek A

PKIXM

X.509 Public Key Infastructure Mechanism

A.1 Abstract

Tato specifikace definuje protokol a konvence potřebné k implementaci GSS-API [1] za použití Internet X.509 Public Key Infastructure Mechanismu.

A.2 Úvod

PKIXM je jednou z instancí dokumentů, které souhrně můžeme nazývat "GSS-API mechanismus". Tento mechanismus poskytuje autentizaci, dohodu na klíči, integritu dat a utajení dat v distribuovaných aplikacích za použití PKI. Protože splňuje interface definovaný v [1], lze PKIXM použít jako náhradu v jakékoli aplikaci využívající bezpečnostní služby pomocí GSS-API volání.

A.3 Algoritmy

V PKIXM může být použita velká škála různých algoritmů. Tento dokument nepředepisuje žádný POVINNÝ algoritmus, protože se domníváme, že vzhledem k rychlému vývoji by toto bylo kontraproduktivní. Je na samotných implementátorech, jakou množinu algoritmů zvolí, s přihlédnutím k jejich potřebám nebo k zajištění interoperability s ostatními knihovnami.

A.3.1 Symetrický šifrovací algoritmus (Sym-ALG)

Symetrický šifrovací algoritmus slouží k zašifrování/odšifrování dat ve funkcích `gss_wrap` / `gss_unwrap` nebo ve spolupráci s heší k zajištění integrity zprávy ve funkcích `gss_getMIC` / `gss_verifyMIC`.

A.3.2 Hešovací algoritmus (MD-ALG)

Hešovací algoritmy se používají jednak při zajišťování integrity, jednak v roli OWF(one-way function) při generovaní sub-klíčů a inicializačních vektorů.

A.3.3 Podpisový algoritmus (S-ALG)

Kryptografický podpisový algoritmus je používán v procesu autentizace k svázání hodnoty integritního součtu autentizačních tokenů s autentizovanou osobou.

A.3.4 Algoritmus k dohodnutí klíče (K-ALG)

Tento algoritmus slouží k dohodě na klíči, který pak iniciátor i akceptor používají jako součást dohodnutého kontextu. Tento kontextový klíč je užíván ke generování sub-klíčů vstupujících do jednotlivých symetrických šifrovacích algoritmů (Sym-ALG), ať už při zajišťování integrity nebo utajení. Vzhledem k tomu, že dohoda na klíči se provádí během X.509 autentizačního handshaku, je i takto vzniklý sdílený kontextový klíč autentizován.

A.3.5 Vyjednávání algoritmů

Během zřizování kontextu v PKIXM se iniciátor i akceptor musí dohodnout na množinách používaných algoritmů.

U MD-ALG a Sym-ALG to probíhá tak, že iniciátor navrhne jemu vyhovující množinu příslušných algoritmů. Akceptor z nich pak vybere ty, které vyhovují jemu. U těchto množin je důležité pořadí, neboť uspořádání má vliv při volbě Quality of Protection (QoP). Pokud k dohodě nedojde, tedy výsledná množina je prázdná, pak kontext nebude zřízen.

Podobně i u K-ALG iniciátor zašle množinu použitelných algoritmů. Pokud to má smysl, zašle i případná data k vygenerovaní kontextového klíče odpovídající prvním z K-ALGů. Akceptor, buď první K-ALG příjme a pokračuje na dokončení kontextového klíče, nebo si zvolí jiný z nabízených algoritmů a začne inicializaci klíče sám.

A.4 Tajná data

A.4.1 Kontextový klíč

Jak už bylo řečeno výše, kontextový klíč vzniká při inicializaci kontextu. Jeho délka by měla být zdola omezena nejdelším klíčem Sym-ALG a zároveň shora omezena maximální velikostí dat použitelných s K-ALG (např. u RSA to znamená délku modulu mínus minimální délka paddingu).

A.4.2 Sub-klíče a inicializační vektory

Sub-klíč a inicializační vektor jsou generovány znovu pro každé použití Sym-ALG v každé přenášené zprávě.

Data délky k-bitů pak vzniknou jako nejlevějších k-bitů zřetězení řetězců S_1 až S_n, kde

$S_i = OWF(context\_key|type|i|seq\_num|context\_key)$

- OWF znamená jeden z algoritmů z množiny MD-ALG
- *context_key* je kontextový klíč
- *type* je typ dat ("INTEGRITY_KEY", "INTEGRITY_IV", "CONFIDENTALITY_KEY", "CONFIDENTALITY_IV")
- i je číslo řetězce
- *seq_num* je sekvenční číslo dané zprávy
- | znamená zřetězení

A.5 Formát tokenů

V této kapitole definujeme viditelné části protokolu PKIXM, které nám zajistí jeho interoperabilitu a nezávislost na samotných programovacích jazycích jak žádá [1].

PKIXM GSS-API mechanismus identifikuje Object Identifier .

Tokeny používané oběma stranami GSS-API komunikace jsou zakódovány pomocí ASN.1 DER pravidel splňujících následující definice (importy z ostatních dokumentů jsou uvedeny až v přehledu na konci kapitoly):

A.5.1 Tokeny na zřizování kontextu

V této sekci jsou definovány dvě třídy tokenů: "iniciátorské" tokeny, které vystupují z volání `gss_init_sec_context()` a vstupují do volání `gss_accept_sec_context()`, a "cílové" tokeny, které vystupují z volání `gss_accept_sec_context()` a do volání `gss_init_sec_context()` vstupují.

Podle [1] části 3.1, má inicializační token při zřizování contextu splňovat následující kostru:

```
InitialContextToken ::= [APPLICATION 0] IMPLICIT SEQUENCE {
   thisMech              MechType,
   innerContextToken     ANY DEFINED BY thisMech
}

MechType ::= OBJECT IDENTIFIER
```

Pak `thisMech` je PKIXM `OBJECT IDENTIFIER` a `innerContextToken` je definovan:

```
PKIXMInnerContextToken ::= CHOICE {
   req    [0] PKIXM-REQ,
   rep-ti [1] PKIXM-REP-TI,
   rep-it [2] PKIXM-REP-IT,
   mic    [3] PKIXM-MIC,
   wrap   [4] PKIXM-WRAP,
   exp    [5] PKIXM-EXPORT
}
```

Tato GSS-API kostra je pak použita u všech tokenů tvořených PKIXM GSS-API mechanismem a nejen u inicializačního. Přesto že to není standardem vyžadováno, nám to umožňuje přesnější kontrolu případných chyb. Hodnota tagu poskytnutá `PKIXMInnerContextToken` (`[0]` do `[5]`) specifikuje každý token. Podobně také informace uložená v položkách tok-id nám identifikuje příslušný token. Mohlo by se tedy zdát, že položky tok-id jsou nadbytečné, avšak není tomu tak. Tagy slouží k tomu, aby tokeny mohly být korektně dekódovány, kdežto tok-id má za cíl kryptograficky svázat daný typ tokenu s daty v něm obsaženými.

Položka `innerContexToken` při zřizování kontextu u PKIXM GSS-API mechanismu obsahuje jednu z těchto zpráv: `PKIXM-REQ`, `PKIXM-REP-TI` nebo `PKIXM-REP-IT`.

PKIXM tokeny sloužící ke zřizování kontextu jsou definovány v souladu s X.509 [8], použití náhodných čísel, a to části 10.3 - dvojfázová autentizace -

pro jednostrannou autentizaci cíle vůči iniciátorovi nebo části 10.4 - třífázová autentizace - pro oboustrannou autentizaci.

Typicky se používá oboustranná autentizace. Přesto, že mechanismus dovoluje i jednostrannou autentizaci, obecně se nedoporučuje.

Tokeny na zřizování kontextu - INICIÁTOR (první token)

Během inicializace kontextu je nezbytné, aby obě zúčastněné strany měly přístup k certifikátu s veřejným klíčem protistrany a dalším datům, sloužícím k jejich ověření. Ty mohou být zaslány v tokenech nebo mohou být získávány lokálně. Je již na samotném implementátorovi PKIXM mechanismu jakým způsobem toto zařídí. Toto není obsahem tohoto dokumentu.

```
PKIXM-REQ ::= SEQUENCE {
        requestToken      REQ-TOKEN,
        certif-data       CertificationData
}

CertificationData ::= SEQUENCE {
        certificationPath               CertificationPath,
        certificateRevocationLists [0]  CertificateLists
                                                OPTIONAL
}

CertificationPath ::= SEQUENCE {
        userCertif            Certificate,
        theCACertificates [0] Certificates OPTIONAL,
        userVerifCertif [1]   Certificate OPTIONAL,
        theCAVerifCertificates [2] Certificates OPTIONAL
}

Certificates ::= SEQUENCE OF Certificate

CertificateLists ::= SEQUENCE OF CertificateList
```

- `userCertif` - certifikát, který je použit k výměně kontextového klíče
- `theCACertificates` - řetězec certifikátů sloužících k ověření `userCertif`
- `userVerifCertif` - certifikát, kterým je podepsán token
- `theCAVerifCertificates` - řetězec certifikátů k `userVerifCertif`

Pokud `userVerifCertif` (a jeho ověřovací řetězec) není přítomen, je i k podpisu použit `userCertif`.

```
REQ-TOKEN ::= SEQUENCE {
        req-contents     Req-contents,
        algId            AlgorithmIdentifier,
        req-integrity    Integrity
}

Integrity ::= BIT STRING
```

- `algId` - identifikátor algoritmu zajišťujícího integritu tokenu
- `req-integrity` - výsledná hodnota, která vznikla užitím algoritmu `algId` na DER-enkódovanou hodnotu položky `req-contents`

Položka Integrity v tokenech `PKIXM-REQ`, `PKIXM-REP-TI` a `PKIXM-REP-IT` bude obsahovat kryptografický podpis za použití zvoleného algoritmu S-ALG.

```
Req-contents ::= SEQUENCE {
        tok-id           INTEGER (256),  -- 0100 (hex)
        context-id       Random-Integer,
        pvno             BIT STRING,
        randSrc          Random-Integer,
        targ-name        Name,
        src-name         Name,
        req-data         Context-Data,
        validity [0]     Valid OPTIONAL,
        key-estb-set     Key-Estb-Algs
}

Random-Integer ::= BIT STRING

Context-Data ::= SEQUENCE {
        seq-number      INTEGER OPTIONAL,
        options         Options,
        md-algs         MD-Algs,
        sym-algs        Sym-Algs,
        channelId       ChannelId OPTIONAL
}

ChannelId ::= BIT STRING
```

```
Options ::= BIT STRING {
        delegation-state (0),
        mutual-state (1),
        replay-det-state (2),
        sequence-state (3),
        conf-avail (4),
        integ-avail (5)
}

MD-Algs ::= SEQUENCE OF AlgorithmIdentifier

Sym-Algs ::= SEQUENCE OF AlgorithmIdentifier

Key-Estb-Algs ::= SEQUENCE OF AlgorithmIdentifier

Valid ::= INTEGER
```

- `tok-id` - identifikátor typu tokenu (hodnota 0100 v hexa)
- `context-id` - identifikátor vytvářeného kontextu
- `pvno` - udává podporované verze protokolu; příslušný bit je nastaven na 1 (v tuto chvíli pouze 0-tý bit)
- `randSrc` - náhodné číslo sloužící k autentizaci
- `targ-name` - jméno cíle
- `src-name` - jméno iniciátora
- `req-data` - iniciátorem navrhovaná data:
  - `seq-number` - počáteční sekvenční číslo
  - `options` - GSS-API flagy
  - `md-algs` - množina MD-ALG
  - `sym-algs` - množina Sym-ALG
  - `channelId` - GSS-API channel binding
- `validity` - doba platnosti contextu
- `key-estb-set` - množina navrhovaných K-ALG algoritmů

O tom, zda autentizace proběhne jako jednostranná nebo oboustranná rozhoduje nastavení položky mutual-state. Ostatní bity v položce `Options` a položka `ChannelId` jsou popsány v základním standardu [1].

Tokeny na zřizování kontextu - Cíl

Většina položek je téměř shodná. Budeme popisovat jen odlišnosti.

```
PKIXM-REP-TI ::= SEQUENCE {
        responseToken    REP-TI-TOKEN,
        certif-data      CertificationData
}

REP-TI-TOKEN ::= SEQUENCE {
        rep-ti-contents  Rep-ti-contents,
        algId            AlgorithmIdentifier,
        rep-ti-integ     Integrity
}

Rep-ti-contents ::= SEQUENCE {
        tok-id           INTEGER (512),   -- 0200 (hex)
        context-id       Random-Integer,
        pvno             BIT STRING,
        src-name         Name,
        targ-name        Name,
        randTarg         Random-Integer,
        randSrc          Random-Integer,
        rep-data         Context-Data,
        validity [0]     Valid  OPTIONAL,
        key-estb-id      AlgorithmIdentifier,
        key-estb-str     BIT STRING
}
```

- `randTarg` - náhodné číslo pro autentizaci zvolené cílem
- `key-estb-id` - zvolený K-ALG algoritmus
- `key-estb-str` - řetězec obsahující data pro získání kontextového klíče v závislosti na zvoleném K-ALG algoritmu

Cíl nastaví odpovídající položky v `Context-Data` a vybere jednu z nabízených verzí protokolu. Pokud by mu některá z množin `key-estb-set`, `md-algs` nebo `sym-algs` nevyhovovala, zřizování kontextu bude přerušeno.

Tokeny na zřizování kontextu - INICIÁTOR (druhý token)

```
PKIXM-REP-IT ::= SEQUENCE {
        responseToken    REP-IT-TOKEN,
        algId            AlgorithmIdentifier,
        rep-it-integ     Integrity
}

REP-IT-TOKEN ::= SEQUENCE {
        tok-id           INTEGER (768),  -- 0300 (hex)
        context-id       Random-Integer,
        randSrc          Random-Integer,
        randTarg         Random-Integer,
        targ-name        Name,
        src-name         Name,
        key-estb-rep     BIT STRING OPTIONAL
}
```

- `key-estb-rep` - řetězec obsahující data pro získání kontextového klíče, pokud byl jako K-ALG zvolen dvoufázový algoritmus

A.5.2 Tokeny jednotlivých zpráv

V této části definujeme dva tokeny: "MIC" token, který vystupuje z volání `gss_getMIC()` a vstupuje do volání `gss_verifyMIC()`, a "Wrap" token, který zpracovává volání `gss_wrap()` a `gss_unwrap()`.

Tokeny jednotlivých zpráv - MIC

Užití volání `gss_getMIC()` vytváří token, oddělený od samotných dat, který může být použit k ověření integrity obdržených dat. Token a data mohou být přijata odděleně, a je tedy na samotné aplikaci, aby zajistila jejich korektní spárování.

```
PKIXM-MIC ::= SEQUENCE {
        mic-header       Mic-Header,
        int-cksum        BIT STRING
}

Mic-Header ::= SEQUENCE {
        tok-id           INTEGER (257), -- 0101 (hex)
        context-id       Random-Integer,
```

```
        md-alg  [0]      AlgorithmIdentifier OPTIONAL,
        sym-alg [1]      AlgorithmIdentifier OPTIONAL,
        snd-seq          SeqNum
}

SeqNum ::= SEQUENCE {
        num              INTEGER,
        dir-ind          BOOLEAN
}
```

- `int-cksum` - kontrolní součet hlavičky a dat počítaný algoritmy uvedenými v položkách `md-alg` a `sym-alg`
- `md-alg` - identifikátor hešovacího algoritmu; pokud není přítomen, bere se defaultní hodnota
- `sym-alg` - identifikátor symetrického šifrovacího algoritmu; pokud není přítomen, bere se defaultní hodnota
- `snd-seq` - pořadí zprávy a její směr

Kontrolní součet je počítán takto: Data sřetězíme s DER-enkódovanou hlavičkou. Toto zahešujeme md-alg algoritmem a vzniklou heš ochráníme symetrickou šifrou sym-alg.

Předpokládá se, že transportní vrstva, kterou aplikace používá, zajišťuje komunikační spolehlivost. Položky snd-seq jsou zde jen k zajištění bezpečnosti (viz sequencing a replay detection v [1]).

Tokeny jednotlivých zpráv - Wrap

Užití volání `gss_wrap()` vytváří tokeny, které obsahují vstupní uživatelova data (případně i zašifrovaná) spolu s jejich kontrolním součtem.

```
PKIXM-WRAP ::= SEQUENCE {
        wrap-header       Wrap-Header,
        wrap-body         Wrap-Body
}

Wrap-Header ::= SEQUENCE {
        tok-id           INTEGER (513), -- 0201 (hex)
        context-id       Random-Integer,
        md-alg  [0]      AlgorithmIdentifier OPTIONAL,
        sym-alg [1]      AlgorithmIdentifier OPTIONAL,
```

```
        snd-seq          SeqNum,
        conf-state       BOOLEAN
}

Wrap-Body ::= SEQUENCE {
        int-cksum        BIT STRING,
        data             BIT STRING
}
```

- `conf-state` - udává, je-li použita ochrana dat šifrováním

Kontrolní součet se počítá stejně jako u MIC tokenů, jen vstupní data jsou před výpočtem šifrována, pokud je ochrana dat vyžadována a kontextem podporována.

A.5.3 Tokeny exportující kontext

Exportovací tokeny se od těch ostatních liší. Neslouží ke komunikaci dvou protistran, ale jejich úkolem je naplnit možnost GSS-API exportovat své kontexty. Token vzniká ve volání `gss_expotr_sec_contex()` a je konzumován voláním `gss_import_sec_contex()`.

```
PKIXM-EXPORT ::= SEQUENCE {
        tok-id           INTEGER (258), -- 0102 (hex)
        context-id       Random-Integer,
        context-key      Random-Integer,
        pvno             BIT STRING,
        targ-name        Name,
        src-name         Name,
        validity         Valid,
        seq-number-send  INTEGER,
        seq-number-rcv   INTEGER,
        options          Options,
        md-algs          MD-Algs,
        sym-algs         Sym-Algs,
        lokal-init       BOOLEAN
}
```

- `seq-number-send` - čítač odeslaných zpráv
- `seq-number-rcv` - čítač přijatých zpráv

- `lokal-init` - udává, je-li daný kontext kontextem iniciátora komunikace

Exportovací tokeny slouží k přenášení již zřízených kontextů. Částečně zřízené kontexty jimi exportovat nelze.

A.5.4 Import definic

V tokenech se používají struktury `Certificate`, `CertificateList`, `Name` a `AlgorithmIdentifier` definované v [11].

A.6 Quality of Protection (QoP)

Parametr QoP je jedním ze vstupů GSS-API volání `gss_getMIC()` a `gss_wrap()` a dovoluje volajícímu si zvolit úroveň zabezpečení - ať už kontroly integrity zpráv nebo jejich utajení.

V PKIXM mechanismu je 32 bitů QoP definováno tak, že horních 16 bitů udává pořadí algoritmu symetrické šifry v daném kontextu a dolních 16 bitů udává pořadí hešovacího algoritmu v kontextu.

Defaultní hodnota 0, jak v horní tak v dolní části parametru, znamená první algoritmus z dané skupiny a naopak hodnota, přesahující počet algoritmů v některé ze skupin, znamená poslední algoritmus.

Literatura

[1] J. Linn *RFC 2743 - Generic Security Service Application Program Interface Version 2, Update 1*, 01/2000, http://www.ietf.org/rfc/rfc2743.txt.

[2] J. Wray, I. Associates *RFC 2744 - Generic Security Service API Version 2 : C-bindings*, 01/2000, http://www.ietf.org/rfc/rfc2744.txt.

[3] J. Kabat, M. Upadhyay *RFC 2853 - Generic Security Service API Version 2 : Java Bindings*, 06/2000, http://www.ietf.org/rfc/rfc2853.txt

[4] C. Neuman, T. Yu, S. Hartman, K. Raeburn *RFC 4120 - The Kerberos Network Authentication Service (V5)*, 07/2005, http://www.ietf.org/rfc/rfc4120.txt

[5] L. Zhu, K. Jaganathan, S. Hartman *RFC 4121 - The Kerberos Version 5 Generic Security Service Application Program Interface (GSS-API) Mechanism: Version 2*, 07/2005, http://www.ietf.org/rfc/rfc4121.txt

[6] C. Adams *RFC 2025 - The Simple Public-Key GSS-API Mechanism (SPKM)*, 10/1996, http://www.ietf.org/rfc/rfc2025.txt

[7] S. Kent *RFC 1422 - Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management*, 02/1993, http://www.ietf.org/rfc/rfc1422.txt

[8] *ITU-T Recommendation X.509 — ISO/IEC 9594-8 Information technology - Open Systems Interconnection - The directory: authentication framework*, 11/1993

[9] *ITU-T Recommendation X.509 — ISO/IEC 9594-8 Information technology - Open Systems Interconnection - The Directory: Public-key and attribute certificate frameworks*, 08/2005

[10] *Public-Key Infrastructure (X.509) Working Group*, http://www.ietf.org/html.charters/pkix-charter.html

[11] R. Housley, W. Polk, W. Ford, D. Solo *RFC 3280 - Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*, 04/2002, http://www.ietf.org/rfc/rfc3280.txt.

[12] *ASN.1 Consortium*, http://www.asn1.org.

[13] *ITU-T Recommendation X.680 — ISO/IEC 8824-1 Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation*, 07/2002.

[14] *ITU-T Recommendation X.681 — ISO/IEC 8824-2 Information technology - Abstract Syntax Notation One (ASN.1): Information object specification*, 07/2002.

[15] *ITU-T Recommendation X.682 — ISO/IEC 8824-3 Information technology - Abstract Syntax Notation One (ASN.1): Constraint specification*, 07/2002.

[16] *ITU-T Recommendation X.683 — ISO/IEC 8824-4 Information technology - Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 specifications*, 07/2002.

[17] *ITU-T Recommendation X.690 — ISO/IEC 8825-1 Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)*, 07/2002.

[18] *ITU-T Recommendation X.691 — ISO/IEC 8825-2 Information technology - ASN.1 encoding rules: Specification of Packed Encoding Rules (PER)*, 07/2002.

[19] *ITU-T Recommendation X.692 — ISO/IEC 8825-3 Information technology - ASN.1 encoding rules - Specification of encoding control notation (ECN)*, 03/2002.

[20] *ITU-T Recommendation X.693 — ISO/IEC 8825-4 Information technology - ASN.1 encoding rules: XML encoding rules (XER)*, 12/2001,

[21] *ITU-T Recommendation X.694 — ISO/IEC 8825-5 IInformation technology - ASN.1 encoding rules: Mapping W3C XML schema definitions into ASN.1*, 01/2004.

[22] S. Legg *RFC 3641 - Generic String Encoding Rules (GSER) for ASN.1 Types*, 10/2003, http://www.ietf.org/rfc/rfc3641.txt

[23] *Open Source ASN.1 Compiler*, http://lionet.info/asn1c/.

[24] *OpenSSL*, http://www.openssl.org.

[25] M. Wahl, S. Kille, T. Howes *Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names*, 12/1997

[26] L. Zhu, P. Leach, K. Jaganathan, W. Ingersoll *RFC 4178 - The Simple and Protected Generic Security Service Application Program Interface (GSS-API) Negotiation Mechanism*, 10/2005, http://www.ietf.org/rfc/rfc4178.txt

[27] S. Hartman *RFC 4768 - Desired Enhancements to Generic Security Services Application Program Interface (GSS-API) Version 3 Naming*, 12/2006, http://www.ietf.org/rfc/rfc4768.txt

[28] Martin Rex *GSSTEST v 1.26*, 10/2002, ftp://ftp.sap.com/pub/ietf-work/gssapi/gsstest/