VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

AKCELERACE ŠIFROVÁNÍ PŘENOSU SÍŤOVÝCH DAT

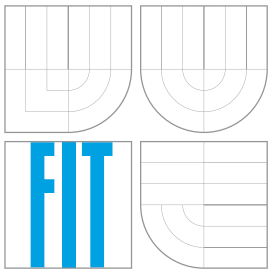
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. KAREL KORANDA

BRNO 2013

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

AKCELERACE ŠIFROVÁNÍ PŘENOSU SÍŤOVÝCH DAT

ACCELERATION OF NETWORK TRAFFIC ENCRYPTION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. KAREL KORANDA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. LIBOR POLČÁK

BRNO 2013

Abstrakt

Tato práce se zabývá tvorbou hardwarové jednotky urychlující proces zabezpečení přenosu síťových dat z vestavěného zařízení, které je součástí systému pro zákonné odposlechy vyvíjeného v rámci projektu Sec6Net. Součástí práce je analýza dostupných bezpečnostních mechanismů pro zabezpečení přenosu dat počítačovou sítí, na jejímž základě je jako nejvhodnější pro cílový systém vybrán protokol SSH. Práce se dále zabývá rozborem možných variant akcelerační jednotky pro protokol SSH a podrobným návrhem a implementací varianty jednotky založené na algoritmu AES-GCM, který zajišťuje důvěrnost, integritu a autentizaci přenášených dat. Implementovaná akcelerační jednotka dosahuje propustnosti 2,4 Gb/s.

Abstract

This thesis deals with the design of hardware unit used for acceleration of the process of securing network traffic within the Lawful Interception System developed as a part of Sec6Net project. First aim of the thesis is the analysis of available security mechanisms commonly used for securing network traffic. Based on this analysis, SSH protocol is chosen as the most suitable mechanism for the target system. Next, the thesis aims at introduction of possible variations of acceleration unit for SSH protocol. In addition, the thesis presents a detailed design description and implementation of the unit variation based on AES-GCM algorithm, which provides confidentiality, integrity and authentication of transmitted data. The implemented acceleration unit reaches maximum throughput of 2,4 Gbps.

Klíčová slova

Zákonné odposlechy, zabezpečení přenosu dat, SSH, AES-GCM, hardwarová akcelerace.

Keywords

Lawful Interception, securing network traffic, SSH, AES-GCM, hardware acceleration.

Citace

Koranda, K. *Akcelerace šifrování přenosu síťových dat.* Brno: FIT VUT v Brně, 2013. Diplomová práce.

Akcelerace šifrování přenosu síťových dat

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Libora Polčáka. Uvedl jsem všechny literární zdroje, které jsem při tvorbě práce použil. Další informace mi poskytli kolegové z projektu Sec6Net.

.....
Karel Koranda
19. května 2013

Poděkování

Na tomto místě bych chtěl poděkovat vedoucímu své diplomové práce Ing. Liboru Polčákovi za cenné připomínky a odborné rady, kterými přispěl k vypracování této práce. Zároveň bych rád poděkoval svým kolegům z projektu Sec6Net za poskytnutí dalších informací.

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Mezi současné trendy dnešní doby patří mimo jiné i stále se zvyšující a neustálá potřeba připojení uživatelů do sítě Internet. Ve vyspělých zemích světa už prakticky každý využívá služeb této počítačové sítě a se zvyšující se rychlostí dochází k neustálému nárůstu objemu přenášených dat. Protože data, která jsou přenášena počítačovou sítí, mohou být často citlivá, narůstá i počítačová kriminalita. Cílem útočníka mohou být nezabezpečená a tím snadno dostupná data. Mnoho organizací zabývajících se síťovou komunikací, mezi nimi například *Internet Engineering Task Force – IETF*, se snaží navrhovat a standardizovat prostředky sloužící k zabezpečení přenášených dat a chránění citlivých informací před potenciálními útočníky.

S rozvíjející se potřebou rychlé komunikace a požadavku na vyšší výpočetní výkon se také uplatňují různé principy neustálého zrychlování přenosu dat počítačovou sítí a jejich zpracování. Mezi oblíbené metody patří i technika HW&SW Codesign, jejímž hlavním cílem je vytvoření pomocných hardwarových výpočetních jednotek, které urychlují výpočetně náročné operace prováděné procesorem počítače.

Jedním ze systémů, který je zdrojem přenosu obzvláště citlivých dat a u kterého je vyžadováno, aby zpracovával data na relativně vysokých rychlostech, je systém pro zákonné odposlechy [16]. Tento systém umožňuje oprávněným orgánům (například Policii ČR) provádět odposlech síťové komunikace osoby podezřelé z páchání trestné činnosti. Nasbíraná data tímto systémem jsou pak poskytována pro účely případného důkazního řízení. Ačkoli je odposlech nařízen oprávněným orgánem, je potřeba zachovat soukromí odposlouchávaných osob. Dále je potřebné, aby data získaná z legálně nařízeného odposlechu nemohla být u soudu zpochybněna, a proto musí být záznam odposlouchávané komunikace úplný. Z těchto důvodů je nezbytné zajistit, aby tato data byla chráněna jak proti jejich neoprávněnému prozrazení, tak proti jejich neoprávněné modifikaci. Jinými slovy je potřeba zajistit spolehlivost jejich přenosu, jejich důvěrnost a integritu.

V rámci projektu *Moderní prostředky pro boj s kybernetickou kriminalitou na Internetu nové generace – Sec6Net*, řešeného na Fakultě informačních technologií Vysokého učení technického v Brně, je vytvářen systém pro realizaci zákonných odposlechů, jehož součástí je též hardwarová sonda provádějící nařízený odposlech. Tato diplomová práce se zabývá analýzou současně dostupných mechanismů pro zabezpečení přenosu dat počítačovou sítí, výběrem vhodného implementovaného mechanismu použitelného pro účely přenosu dat z vyvíjené sondy, návrhem jeho hardwarové akcelerace s využitím techniky HW&SW Codesign a implementací příslušné hardwarové akcelerační jednotky.

1.1 Členění práce

Kapitola 2 stručně čtenáře seznamuje s architekturou systému pro zákonné odposlechy a současným stavem realizovaného systému v rámci projektu Sec6Net. Kapitola 3 se zabývá analýzou běžně používaných prostředků pro zabezpečení dat přenášených počítačovou sítí a dokumentuje výběr vhodného mechanismu použitelného na cílovém systému. Uvedení detailů použití bezpečnostních prostředků poskytovaných zvoleným mechanismem je popsáno v kapitole 4. V kapitole 5 jsou představené dostupné implementace zvoleného bezpečnostního mechanismu a kapitola 6 prezentuje návrh hardwarové akcelerace mechanismu s podrobnějším zaměřením na možné varianty akcelerační jednotky. Podrobný návrh nejvhodnější navržené varianty je pak popsán v kapitole 7. Dosažené výsledky a možnosti pokračování práce jsou prezentovány v kapitole 8. Práce je nakonec zhodnocena v kapitole 9.

Kapitola 2

Systém pro zákonné odposlechy

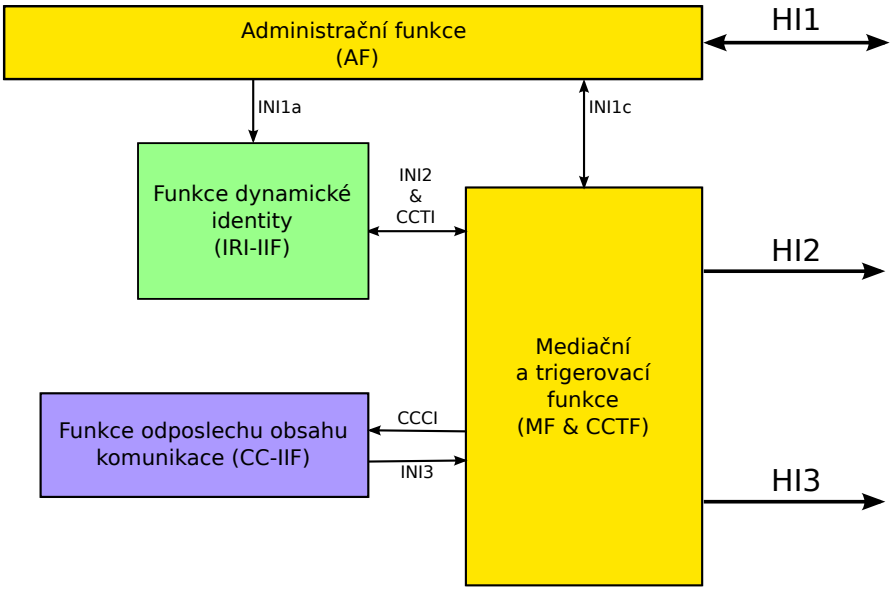
Systém pro zákonné odposlechy je odezvou na stále se zvyšující a zdokonalující kybernetickou kriminalitu, která se rozvíjí především díky rozmachu sítě Internet. V důsledku toho byl navržen systém pro zákonné odposlechy, který umožňuje orgánům činným v trestním řízení (*Law Enforcement Agency – LEA*) vydat příkaz na sledování podezřelých osob na telefonní či počítačové síti za účelem poskytnutí důkazního materiálu pro případné soudní řízení. Způsob, jakým by měl být systém pro zákonné odposlechy realizován, je definován normami *ETSI* [16], které specifikují, jaké moduly má systém obsahovat, jaké je mezi nimi rozhraní a případně co dalšího systém a jeho části musí (nebo by měl, norma v některých částech není striktní) splňovat.

Systém pro zákonné odposlechy je rozdělen na dvě části - část, která se nachází u poskytovatele telekomunikačních služeb (*ISP*), a část, která se nachází u oprávněného orgánu (policie). První část slouží především pro příjem požadavků a jejich autorizaci od oprávněného orgánu, sledováním aktivity podezřelého uživatele (filtrace) a předáním získaných dat druhé části systému. Ta zajišťuje sběr dat, jejich archivaci a vyhodnocení. Systém musí zajistit, že kopie všech odposlechnutých dat bude doručena až k orgánům činným v trestním řízení.

2.1 Architektura systému

V rámci projektu *Moderní prostředky pro boj s kybernetickou kriminalitou na Internetu nové generace* (*Sec6Net*), realizovaného na Fakultě informačních technologií VUT v Brně, je vytvářen prototyp systému pro sběr dat pro zákonné odposlechy [47] (*Lawful Interception System – LIS*), který reprezentuje část systému pro zákonné odposlechy provádějící filtraci a sběr dat sledovaného cíle. Shromážděná data budou následně poskytnuta pro rekonstrukci a jejich klasifikaci. Součástí realizace systému je také vývoj vysokorychlostní sondy a mikrosondy, které slouží jako zařízení připojená do sítě poskytovatele provádějící požadovaný sběr dat. Cílem projektu Sec6Net je vytvořit prototyp zjednodušeného systému, který by měl sloužit především pro výzkum a vývoj nových technik dynamické identifikace uživatele, vývoj nových metod rekonstrukce datových toků a jako testovací prostředí pro zdokonalování hardwarových sond.

Zjednodušený návrh systému v rámci projektu *Sec6Net* je znázorněn na obrázku 2.1. Návrh vychází z norem ETSI a existujícího řešení společnosti Cisco [9].



Obrázek 2.1: Architektura prototypu systému pro zákonné odposlechy [47]

2.1.1 Rozhraní mezi ISP a LEA

Norma ETSI [16] definuje komunikační rozhraní mezi částí systému nacházející se u poskytovatele služeb a mezi částí systému patřící orgánům činným v trestním řízení. Toto rozhraní (označené jako *Hand-Over Interface – HI*) se skládá ze tří částí:

- rozhraní *HI1* je určeno pro předávání požadavku na odposlech ze strany oprávněných orgánů a pro zjištění informací o průběhu aktuálně probíhajícího odposlechu,
- rozhraní *HI2* slouží pro předávání průběžných metainformací o aktivitě sledovaného cíle (např. připojení či odpojení cíle, jeho změna identity apod.) a
- rozhraní *HI3*, určené pro předávání zachyceného obsahu celé komunikace oprávněným orgánům.

2.1.2 Moduly systému a jejich rozhraní

Navržený systém znázorněný na obrázku 2.1 se skládá ze čtyřech částí. *Administrační funkce* (*Administration Function – AF*) zajišťuje kontrolu správnosti požadavku na nařízený odposlech, který je přijímán na rozhraní HI1. Součástí AF je také fronta všech požadovaných odposlechů. U každého požadavku je potřeba zajistit inicializaci odposlechu, konfiguraci ostatních částí systému pro tento odposlech a jeho ukončení tak, aby byla zaznamenána všechna odpovídající data v požadovaném časovém intervalu. Je také nezbytné, aby AF byla schopna provádět archivaci všech proběhnutých odposlechů.

Další částí systému je modul označený jako *Funkce dynamické identity* (*Intercepted Related Information - Internal Interception Function – IRI-IIF*). Tento modul zajišťuje identifikaci uživatelů a detekci zpráv, které patří sledované osobě. Současně provádí zjišťování aktuální identity této osoby, která se v průběhu odposlechu může dynamicky měnit (např. změna IP adresy protokolem DHCP). Tyto změny musí být rovněž promítnuty v dalších částech systému. Tato část systému rovněž provádí detekci začátků a konců odposlouchávaných spojení a tyto informace dále předává přes rozhraní INI2 mediační funkci (viz dále).

Modul *Funkce odposlechu obsahu komunikace* (*Content of Communication - Internal Interception Function – CC-IIF*) realizuje odposlech obsahu komunikace sledované osoby. Provádí filtraci datových toků a data, která odpovídají požadovanému vzoru, kopíruje a předává přes rozhraní INI3 mediační funkci. Tento modul dostává požadavky na započetí a ukončení odspolechu přes rozhraní CCCI. Tato část systému pro zákonné odposlechy je realizována hardwarovou sondou.

Mediační funkce (*Mediation Function – MF*) provádí zpracování dat přijatých na rozhraní INI2 (metainformace o datovém toku) a INI3 (data). Z těchto informací vytváří datové jednotky, které dále přeposílá přes rozhraní HI2 a HI3 oprávněným orgánům. Součástí modulu je také *Trigerovací funkce* (*Content of Communication Trigger Function – CCTF*), která zajišťuje konfiguraci modulů CC-IIF přes rozhraní CCCI.

Požadavky na rozhraní INI3

Rozhraní INI3 slouží pro předávání odposlechnutého obsahu komunikace mediační funkci a dále oprávněným orgánům. Protože blok CC-IIF bude realizován pomocí hardwarové sondy, která bude zapojena do sítě ISP a která musí provádět přenos citlivých dat přes počítačovou síť, norma ETSI [17] předepisuje jisté požadavky, které toto rozhraní musí splňovat (nepředepisuje přímo způsob, pouze uvádí příklad možného provedení). Požadavky jsou:

- spolehlivost přenosu dat (data nesmí být ztracena),
- zabezpečení přenášených dat (data nesmí být dostupná neoprávněným subjektům a musí být zajištěna proti neoprávněné modifikaci).

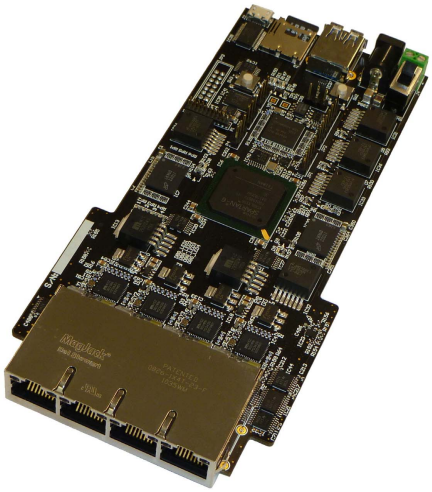
Norma uvádí jako příklad pro přenos dat přes rozhraní INI3 s využitím protokolů UDP a RTP. Ty ale nezajišťují spolehlivou komunikaci, kterou musí systém splnit jiným způsobem, na aplikační úrovni. Příklad na zabezpečení dat pro rozhraní HI (nikoli pro INI3) je uveden v normě ETSI [17], kde doporučeno využití protokolu TLS nebo standardu IPsec. Bezpečnostní model doporučený pro rozhraní INI3 je uveden v další normě ETSI [18]. Zde jsou uvedené příklady mechanismů zabezpečení přenosu dat na úrovni rozhraní, mechanismy omezení přístupu k prvkům systému, audit událostí apod. Zajištění důvěrnosti a integrity přenášených dat jsou ale uvedené bez konkrétních protokolů nebo kryptografických algoritmů a tím jsou ponechány na návrháři systému. Tato diplomová práce se bude zabývat zvolením vhodného mechanismu v kapitole 3.

V rámci projektu Sec6Net byl pro zajištění spolehlivosti přenosu dat zvolen protokol TCP, který nativně poskytuje potřebné mechanismy. Srovnání protokolů UDP a TCP a možností zajištění spolehlivosti přenosu jsou zdokumentovány v technické zprávě [30] a nebudou zde blíže rozebírány.

2.2 Mikrosonda

Mikrosonda (jinak též označovaná jako uSonda) je určena pro nasazení k menším poskytovatelům telekomunikačních služeb na linkách s maximální rychlostí přenosu 1 Gb/s. Sonda má za úkol provádění filtrace dat a kopii dat, která odpovídající požadovanému nastavení, zasílat na mediační zařízení. Analýzou požadavků a návrhem mikrosondy se zabývá technická zpráva [38], výsledek je pak prezentován v [39].

Sonda se musí vypořádat s několika problémy. Je nezbytné, aby při výpadku linky, na které probíhá předávání odchycených dat mediačnímu zařízení, byla všechna data dočasně



Obrázek 2.2: Mikrosonda uG4-150

uložena. Současně je nutné, aby byla odchycena veškerá komunikace sledovaného cíle, protože pokud by tomu tak nebylo, v rámci soudního řízení by mohla být zpochybněna relevance získaného odposlechu. Sonda musí být snadno konfigurovatelná a musí být schopna reagovat na měnící se identitu sledovaného cíle. Tato konfigurace by měla být přístupná pomocí vzdáleného přístupu. Jak bylo zmíněno v sekci 2.1.2, je nutné zajistit, že přenos dat z a do mikrosondy bude spolehlivý a zabezpečený.

2.2.1 HW platforma pro mikrosondu

Aby sonda měla požadovanou propoustnost a výkonnost realizovaného filtru, byl za cílový výpočetní prvek zvolen čip FPGA. Ten umožňuje na vestavěných mikroprocesorech MicroBlaze běh softwarového vybavení, jehož výpočetně náročné části mohou být hardwarově akcelerovány v programovatelné logice FPGA. Zvolení technologie FPGA je výhodné také pro výzkum a vývoj nových či zdokonalování existujících technik pro detekci a filtrování datového provozu a jejich zpracování.

Mikrosonda obsahuje čtyři ethernetová síťová rozhraní. Pro potřebu monitorování síťové komunikace jsou vyhrazená dvě z nich, zbylá dvě jsou určena pro konfiguraci sondy vzdáleným přístupem a pro zasílání dat mediačnímu zařízení.

2.2.2 SW platforma pro mikrosondu

V rámci projektu Sec6Net je pro účely mikrosondy také přizpůsobován operační systém Linux, který zajistí přístup ke komponentám systému a současně umožní, aby na mikrosondě mohly běžet potřebné softwarové aplikace umožňující vzdálený přístup, konfiguraci síťových rozhraní apod.

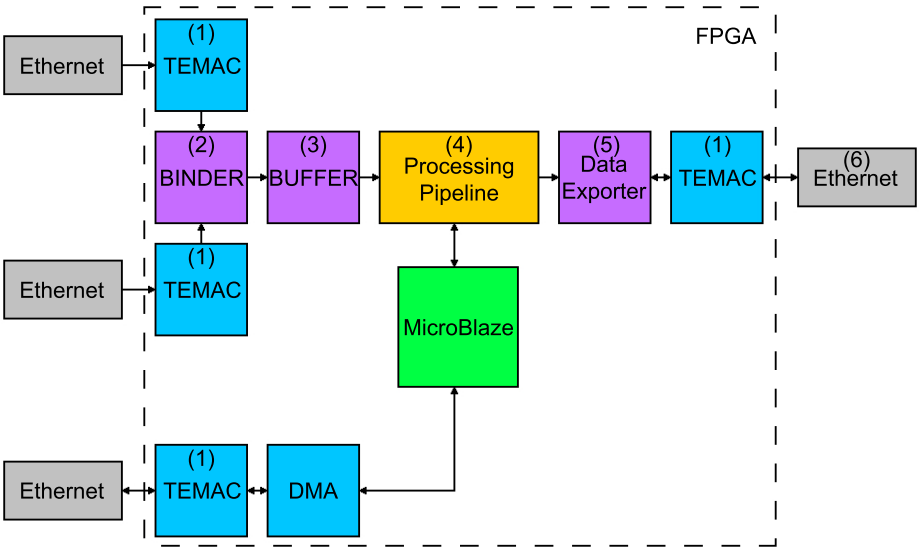
2.2.3 Architektura mikrosondy

Požadavky na architekturu mikrosondy [38] byly definované následovně:

- zpracování datových toků dvou ethernetových rozhraní na rychlosti 1 Gb/s,
- nezbytná podpora až 1 000 filtrovacích pravidel,

- při výpadku spojení mezi sondou a mediačním zařízením nesmí docházet ke ztrátě odposlouchávaných dat,
- dynamická změna pravidel a schopnost jejich aplikace během aktuálně běžícího odposlechu,
- výstupní tok dat musí mít rychlost alespoň 200 Mb/s.

Na základě těchto požadavků byla navržena architektura, která je zjednodušeně znázorněná na obrázku 2.3. Cílem návrhu bylo, aby architektura byla sestavena z existujících komponent pro FPGA a procesoru MicroBlaze, který bude provádět řízení její činnosti. Protože procesor MicroBlaze není příliš výkonný a čistě softwarová implementace by nedokázala splnit požadovanou rychlost filtrování, zpracování a přeposílání datových toků, bylo identifikováno, že je nutné hardwarově akcelerovat mechanismy TCP/IP a současně probíhá vývoj této akcelerace. Procesor MicroBlaze by pak měl zajišťovat pouze konfiguraci jednotlivých bloků architektury mikrosondy.



Obrázek 2.3: Zjednodušené schéma architektury mikrosondy

Procesor MicroBlaze (případně dvě jeho instance) slouží pro zmíněnou konfiguraci mikrosondy a usnadnění filtrace provozu. Pro ethernetová rozhraní byly zvoleny Xilinx IPcore moduly *Tri-Mode Ethernet Media Access Controller – TEMAC* (v obrázku 2.3 vyznačené bloky (1)), za sběrnici, po které bude probíhat přenos a zpracování datových toků, byla vybrána sběrnice AXI4-Stream [52]. Pro komunikaci mezi procesními prvky a procesorem byla zvolena sběrnice AXI4-Lite [52].

Na obrázku 2.3 je vyznačena komponenta označená jako *Binder* (2). Tato komponenta slučuje dvě sběrnice AXI-Stream do jedné. Výstup této komponenty je zapojen do bloku *Buffer* (3), který slouží jako zpožďovací a vyrovnávací komponenta mezi vstupními rozhraními a procesní linkou. Ta je označena jako blok *Processing Pipeline* (4) a slouží pro samotnou klasifikaci a filtraci datového toku. Data, která odpovídají nakonfigurovanému filtru jsou následně zpracována v bloku *Data Exporter* (5). Jednotka *Data Exporter* provádí identifikaci a označení dat odpovídajícímu odposlechu [47]. Součástí této jednotky by měl být také blok zajišťující bezpečnost těchto dat, protože tato data budou přenášena přes

další síťové rozhraní (6) mimo mikrosondu a dále přes počítačovou síť. Toto síťové rozhraní odpovídá rozhraní INI3, které bylo představené v sekci 2.1.2.

Součástí této práce bude návrh a implementace části architektury jednotky *Data Exporter* realizující zabezpečení dat přenášených ze sondy, přičemž navržená jednotka bude provádět akceleraci výpočetně náročných operací zabezpečení a její konfigurace bude prováděna z operačního systému běžícího na procesoru MicroBlaze. První fáze návrhu jednotky budou blíže popsány v kapitole 6.

Kapitola 3

Zajištění bezpečnosti přenosu dat

Jak již bylo zmíněno, v rámci systému pro zákonné odposlechy jsou přenášena citlivá data odposlouchávaného účastníka. Je nezbytné zabránit jejich odhalení, ať už před zaměstnanci poskytovatele, nebo před jinými útočníky nacházejícími se na počítačové síti. Říkáme, že je nutné zajistit důvěrnost (ochranu proti neoprávněnému prozrazení informace) a integritu (ochranu proti neoprávněné modifikaci informace) přenášených dat. Mezi existující a používané prostředky patří například skupina protokolů IPsec, a dále protokoly SSL/TLS a SSH a další. Jmenovanými prostředky se bude stručně zabývat tato kapitola, přičemž budou následně porovnány a nejvhodnější mechanismus bude vybrán pro použití v rámci navrhovaného systému pro zákonné odposlechy. Porovnání mechanismů a výběr nejvhodnějšího je současně publikován v [30].

3.1 IPsec

Standard IPsec [35, 32] je sada protokolů určená pro zajištění autentizace, důvěrnosti a integrity dat přenášených mezi libovolnými dvěma uzly, které spolu komunikují v počítačové síti. Kromě peer-to-peer komunikace poskytuje také prostředky pro zabezpečení komunikace v rámci jedné multicastové skupiny. Mechanismy, které poskytuje, se nachází na třetí (síťové) vrstvě ISO/OSI modelu, čímž zajišťuje zabezpečení protokolu IP a všech protokolů vyšších vrstev. Díky svému umístění IPsec nevyžaduje součinnost s aplikací, jejíž data jsou považována za citlivá a která je nutné zabezpečit. Na druhé straně je však vyžadována podpora v jádře operačního systému.

Standard IPsec je nativní součástí protokolu IPv6, je možné ho ale používat i v rámci sítí založených na síťovém protokolu IPv4.

3.1.1 Security Association

Aby bylo možné zabezpečit komunikaci mezi dvěma komunikujícími uzly, je mezi nimi dle standardu IPsec nejprve sestaveno kryptograficky zabezpečené spojení, terminologicky označované jako *Security Association – SA* [35]. SA obsahuje identifikaci obou komunikujících stran a v rámci jednoho tohoto spojení mají obě dvě strany dohodnuté jedinečné kryptografické klíče relace, integritní klíče, sekvenční čísla vyměňovaných zpráv a další. Protože se předpokládá, že se pohybujeme v sítích, v nichž je podporován duplexní přenos dat, pro jedno logické spojení jsou vytvořené vždy dvě logické SA - pro každý směr komunikace jedna. Aby bylo možné rozlišit více logických spojení mezi uzly, je zaveden takzvaný *Security Parameter Index – SPI*, který jednoznačně určuje každou SA. Tento identifikátor

je při komunikaci uváděn v hlavičkách přenášených datagramů, aby byla určena příslušnost datagramu k dané SA.

Každý z uzlů si udržuje lokální databázi všech aktuálně ustanovených SA, tato databáze se označuje jako *Security Association Database – SAD*. Jednotlivé položky v databázi obsahují kromě SPI například také IP adresy komunikujících uzlů, sekvenční čísla, vybraný autentizační či šifrovací algoritmus, délku života SA a další.

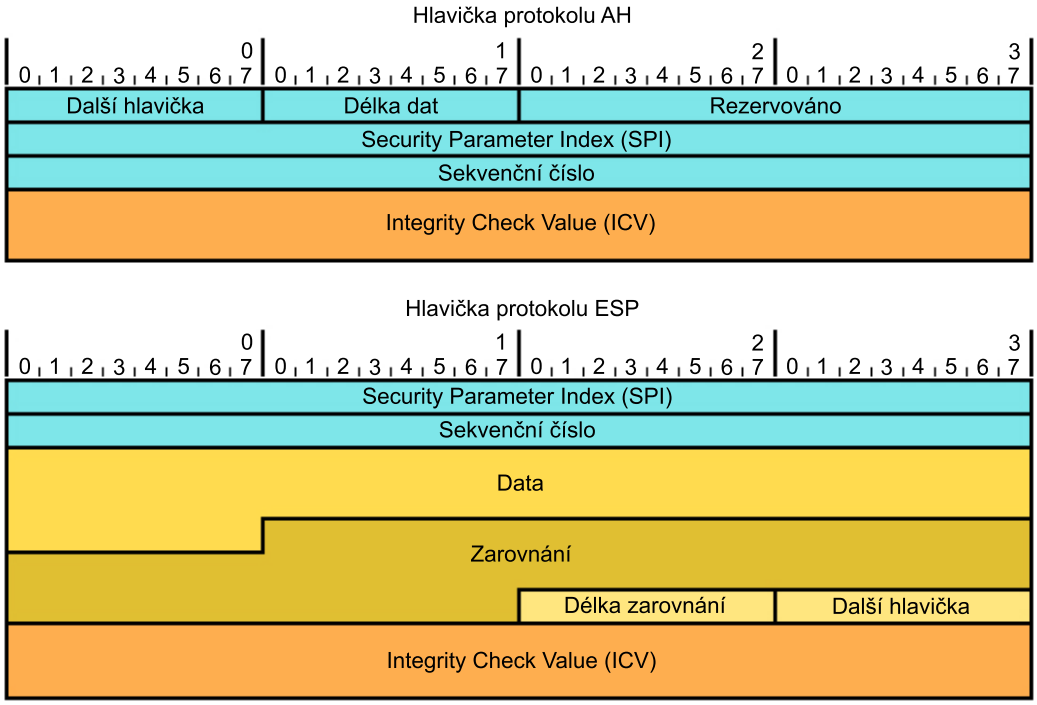
3.1.2 Protokoly IPsec

V rámci standardu IPsec jsou definovány tři základní protokoly - protokoly *Authentication Header – HA* [33], *Encapsulating Security Payload – ESP* [34] a protokol *Internet Key Exchange – IKE* (nyní ve verzi 2, *IKEv2*) [31].

Protokoly AH a ESP

Protokoly AH i ESP zapouzdřují data vyšších vrstev. Protokol AH zajišťuje výhradně integritu dat a autentizaci jejich původce. Díky použitému mechanismu *anti-replay window* současně dokáže zamezit útokům opakovaným zasláním dat (tzv. útoky *replay*). Naproti tomu protokol ESP je schopem mimo integrity zajistit také důvěrnost přenášených dat pomocí šifrování.

Položky hlaviček datagramů těchto protokolů jsou znázorněné na obrázku 3.1. Oranžovou barvou je na obrázku znázorněn integritní kontrolní součet vypočítaný příslušným zvoleným algoritmem z přenášených dat (například klíčovanou hashovací funkcí *SHA-256*). Žlutými odstíny jsou v hlavičce protokolu ESP znázorněné ty části přenášeného datagramu, které jsou zabezpečené šifrováním.

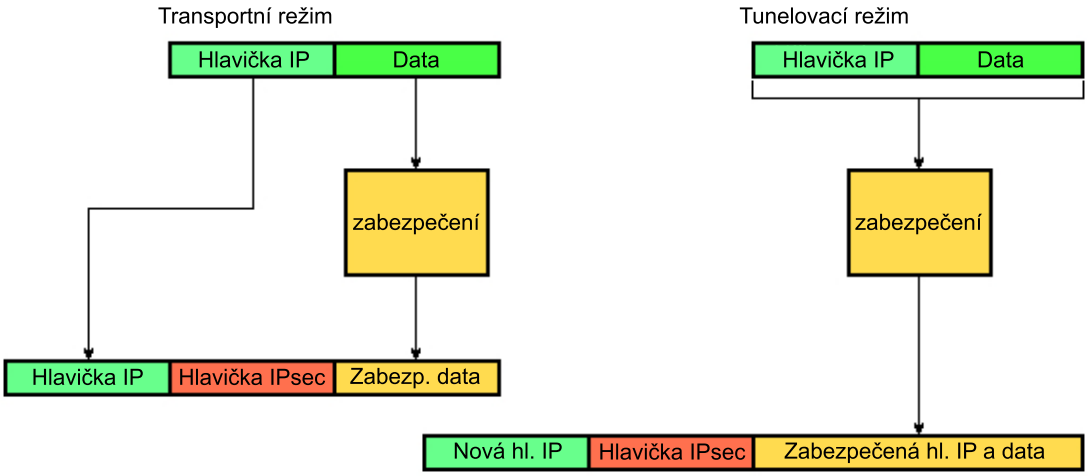


Obrázek 3.1: Hlavičky protokolů AH a ESP standardu IPsec

Protokoly AH a ESP mohou být vzájemně v rámci jedné SA kombinovány a v průběhu přenosu může dojít dokonce k několikanásobnému zapouzdření původního datagramu různými hlavičkami AH i ESP (toto je umožněno díky existenci režimů popsaných v sekci 3.1.2). Může se zdát, že by postačoval pouze protokol ESP, jelikož je schopen zajistit jak integritu, tak důvěrnost. Odpověď na to, proč existuje také protokol AH je jednoduchá. Díky umístění zabezpečení standardu IPsec dochází při šifrování s využitím protokolu ESP také k zašifrování hlaviček protokolů transportní vrstvy (protokolů TCP a UDP). Toto může značným způsobem znesnadnit práci například firewallům a směrovačům, které při své činnosti mohou nahlížet na informace obsažené v těchto hlavičkách. S tímto problémem se protokol AH nepotýká, protože šifrování neprovádí. Díky jeho existenci je tak možné zajistit integritu i v sítích, kde je šifrování protokolem ESP nemožné nebo nežádoucí. V takových případech je pak ale nutné zajistit důvěrnost přenášených aplikačních dat jiným způsobem na vyšších vrstvách ISO/OSI modelu.

Režimy

Standard IPsec definuje dva režimy použití protokolů AH a ESP - transportní režim a tunelovací režim. IPsec pracující v transportním režimu provádí vkládání hlavičky IPsec mezi hlavičku protokolu síťové a transportní vrstvy. To umožňuje, aby původní hlavička protokolu IP zůstala v otevřené podobě. Tunelovací režim zapouzdří celý datagram protokolu IP hlavičkou IPsec a následně provede zapouzdření vzniklého datagramu novou hlavičkou protokolu IP, často s jinými IP adresami, než s těmi, které byly obsažené v původním datagramu. Rozdíl mezi oběma režimy je názorně ukázán na obrázku 3.2.



Obrázek 3.2: Rozdíl mezi transportním a tunelovacím režimem IPsec [32]

Transportní režim je využíván především pro logická spojení mezi dvěma koncovými uzly. Tunelovací režim se pak využívá mezi směrovači, případně mezi směrovačem a koncovým uzlem, kterým je například brána do lokální sítě. Tunelovací režim tímto způsobem často zajišťuje bezpečnost pouze po části cesty datagramu počítačovou sítí. Oba dva režimy je také možné libovolně kombinovat.

Protokol IKEv2

Ve standardu IPsec je pro ustanovení relace mezi dvěma vzdálenými uzly zaveden mechanismus reprezentovaný protokolem IKEv2. Tento protokol definuje, jakým způsobem je ustanoveno logické spojení SA a současně proces výměny náhodného materiálu pro výpočet klíčů relace a jejich průběžnou obnovu. Protokol je založen na kombinaci dříve existujících mechanismů protokolů ISAKMP [42] a Oakley [46].

Protokol pracuje ve dvou fázích. V rámci první fáze je provedena vzájemná autentizace obou komunikujících uzlů (což je podstatný rozdíl oproti protokolům založených na modelu klient-server, kde zpravidla dochází k autentizaci pouze jedné strany, a to serveru) a výměně náhodných dat určených pro výpočet klíčů relace. Tímto je ustanoveno první logické spojení, s jehož využitím se v rámci druhé fáze protokolu IKEv2 provede obdobným způsobem sestavení dalších SA pro zabezpečení konkrétních datových toků.

První fáze protokolu IKEv2 je založena na mechanismu Diffie-Hellman [48], který je v kryptografii hojně využíván pro ustavení klíčů relace mezi dvěma stranami. Tato fáze protokolu IKEv2 může být provedena dvěma způsoby. Prvním způsobem je takzvaný agresivní režim, který provede vzájemnou autentizaci a ustavení relačních klíčů s využitím výměny právě třech zpráv. Druhým způsobem je tzv. hlavní režim, který obohacuje agresivní režim o další možnosti. Kromě výpočtu klíčů relace se také komunikující strany dohodnou na použití konkrétních kryptografických algoritmů pro zajištění integrity a důvěrnosti dat.

Druhá fáze protokolu IKEv2 je označovaná jako *vytvoření dceřinné SA* (v RFC 4306 [31] je označena jako *CREATE_CHILD_SA*). Pracuje v tzv. rychlém režimu. V rámci této fáze je mimo kryptografických klíčů pro vytvářenou SA také dohodnuto, pro který konkrétní datový tok bude vytvářená SA využívána.

3.1.3 Podporovaná kryptografie

Standard IPsec je již od počátku založen na tom, že je navrhován dostatečně univerzálně, aby v průběhu let mohl reagovat na změny bezpečnosti dostupných kryptografických mechanismů. Současné kryptografické algoritmy nemusí být v budoucnosti považovány za bezpečné a ve standardu IPsec bude nutné toto reflektovat. V rámci současně platných norem standardu IPsec [25] je definováno využití algoritmů *MD5*, *SHA-1*, *DES* a *AES* pro zajištění integrity dat, pro zajištění důvěrnosti pak vybírá šifrovací algoritmy *RC4*, *RC5*, *IDEA*, *3IDEA*, *CAST*, *Blowfish*, *DES*, *3DES*, *AES*. V současné době je upřednostňován šifrovací algoritmus AES, jeho použití v rámci standardu IPsec je definováno v [19, 26, 27].

3.1.4 Výhody a nevýhody IPsec

Za hlavní výhodu standardu IPsec je možné považovat jeho vysokou robustnost a nezávislost aplikace na použitém bezpečnostním mechanismu. Současně IPsec umožňuje mezi dvěma komunikujícími uzly zajistit zabezpečení různých datových toků různými kryptografickými mechanismy a relačními klíči. Současně je IPsec transparentní k prvkům pracujícím na síťové vrstvě.

Jednou z nevýhod standardu je nutnost rozsáhlé podpory v jádře OS. Také je obtížné připravit kompletní hardwarovou podporu standardu v rámci síťových prvků (v současné době však probíhá vývoj takové podpory založené například na dynamické částečné rekonfiguraci FPGA [50]). Další nevýhodou může být šifrování transportní vrstvy, což může způsobovat komplikace činnosti firewallům. V sítích založených na protokolu IPv4 také může nastat problém s mechanismem NAT. Díky obtížné konfiguraci také není IPsec tak

široce rozšířen, jak bylo plánováno, a to ani v rámci sítí protokolu IPv6, pro které byl primárně určen.

3.2 Protokoly SSL/TLS

Protokol *Transport Layer Security – TLS* [13] a jemu předcházející protokol *Secure Sockets Layer – SSL* [20] jsou protokoly poskytující zabezpečenou komunikaci na aplikační vrstvě. Umístění protokolů odpovídá šesté (prezentační) vrstvě ISO/OSI síťového modelu. Nezaručuje zabezpečení hlaviček transportní ani síťové vrstvy a úzce spolupracuje s aplikací, jejíž data jsou považována za citlivá. Protokol TLS lze rozdělit do dvou podvrstev. První podvrstvou je vrstva záznamů, která odpovídá hlavičce protokolu. Druhá podvrstva je tvořena třemi pomocnými protokoly protokolu TLS, kterými jsou protokoly *ChangeCipherSpec, Alert, Handshake* a nebo libovolném protokolu aplikační vrstvy ISO/OSI modelu (např. HTTP). Tato podvrstva tak odpovídá vlastně typu přenášené zprávy. Protokoly SSL/TLS zajišťují pro přenášená aplikační data důvěrnost a integritu. Od roku 2012 je definována také podpora zabezpečeného přenosu dat protokolem UDP [49].

V současné době je aktuální verzí protokolu TLS verze 1.2 a protokolu SSL 3.0. Použití starších verzí SSL je zakázáno, protože již nejsou považovány za dostatečně bezpečné [51].

Protože protokol SSL je předchůdcem protokolu TLS, bude následující text hovořit pouze o protokolu TLS.

3.2.1 Pomocné protokoly

Jak bylo zmíněno, pomocné protokoly protokolu TLS jsou tři - Handshake, ChangeCipherSpec a Alert. Tyto protokoly byly zavedeny především pro účely ustanovení spojení, určení jeho vlastností, k jeho ukončení a k signalizaci neočekávaných nebo neobyčejných stavů, ke kterým během spojení mohlo dojít.

Protokol Handshake

Protokol Handshake, jak napovídá jeho název, slouží k ustanovení relace mezi aplikacemi, přičemž komunikace mezi nimi je založena na modelu klient-server. Za iniciaci spojení odpovídá vždy klientská aplikace. Výměna zpráv mezi oběma stranami probíhá dle následujícího zjednodušeného schématu:

1. Klientská aplikace zasílá svůj požadavek na sestavení spojení cílovému serveru. Zaslaná zpráva obsahuje náhodné číslo R_A a současně seznam klientem podporovaných kryptografických mechanismů.
2. Aplikace serveru odpovídá klientské aplikaci zprávou, která obsahuje náhodné číslo R_B, svůj certifikát a volbu nejsilnějšího klientem nabídnutého kryptografického mechanismu.
3. Klientská aplikace zasílá serveru náhodné číslo S označované jako *PreMaster Secret* zašifrované veřejným klíčem serveru, který je součástí zaslaného certifikátu.
4. Klientská i serverová aplikace provedou výpočet tzv. *Master Secret*, ze kterého jsou odvozené relační klíče pro zvolený šifrovací algoritmus a pro mechanimus zajišťující integritu. *Master Secret* je vypočteno z dříve přenesených R_A, R_B a S.

V uvedeném postupu je uvedeno, že server zasílá klientovi svůj certifikát. Klient si díky tomu může ověřit identitu serveru. Je patrné, že autentizace je v tomto případě ale pouze jednosměrná (což například u běžných aplikací využívajících například protokol HTTP dostačuje). Protokol TLS volitelně podporuje také obousměrnou autentizaci. Pokud aplikace toto vyžaduje, je tato autentizace prováděna mezi body 2. a 3. uvedeného schématu tak, že klient zašle serveru svůj certifikát, který umožní serveru ověřit identitu připojeného klienta.

Protokol ChangeCipherSpec

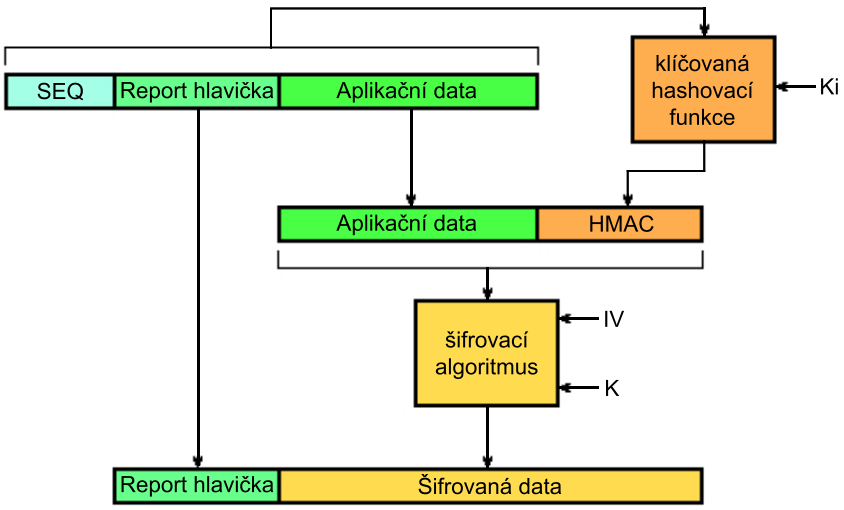
Protokol ChangeCipherSpec reprezentuje zprávu zasílanou po ustanovení relace. Zpráva reprezentuje okamžik, od kterého je všechna následující komunikace zabezpečena způsobem dohodnutým během fáze protokolu Handshake. Je definováno, že zprávu si musí zaslat vzájemně obě dvě komunikující strany.

Protokol Alert

Protokol Alert zajišťuje přenos zpráv, kterými komunikující strana informuje druhou stranu o vzniklé chybě, případně sděluje jiné upozornění. Současně je v rámci tohoto protokolu definována speciální zpráva *closure*, kterou zasílá jedna ze stran ve chvíli, kdy chce indikovat, že již nemá další data k odeslání a označuje touto zprávou požadavek na ukončení spojení.

3.2.2 Vrstva záznamů

Vrstva záznamů je reprezentována protokolem Record, a provádí pouze zapouzdření pomocných protokolů TLS a protokolů aplikační vrstvy. Součástí je definice formátu hlavičky záznamu. Tato hlavička obsahuje především typ přenášené zprávy, tedy označení, zda se přenáší zpráva protokolů Handshake, ChangeCipherSpec, Alert nebo aplikační data. Přenášená aplikační data jsou zabezpečena šifrováním a mechanismy zajišťujícími integritu.



Obrázek 3.3: Schéma zabezpečení dat protokoly SSL/TLS [32]

Součástí definice protokolu Record je také postup zabezpečení aplikačních dat, který je stručně znázorněn na obrázku 3.3. Zabezpečení probíhá následovně:

1. K sestavovanému rámci je určeno jeho sekvenční číslo v rámci ustanovené relace.
2. Ze sekvenčního čísla, budoucí hlavičky záznamu a aplikačních dat je vypočten autentizační kód zprávy *HMAC*[1] s využitím integritního klíče K_i.
3. K aplikačním datům je připojen vytvořený HMAC a tato data jsou zašifrována šifrovacím algoritmem zvoleným při ustanovení relace (parametry šifrování jsou šifrovací klíč K a inicializační vektor IV).
4. Šifrovaná data jsou zapouzdřena hlavičkou záznamu. Vzniklý datagram je připraven k odeslání.

3.2.3 Podporovaná kryptografie

Protokol TLS verze 1.2 podporuje pro účely autentizace serveru a volitelně klienta asymetrický šifrovací algoritmus RSA a varianty mechanismu Diffie-Hellman [48]. Pro účely šifrování jsou pro zajištění integrity dat vybrány algoritmy *MD5*, *SHA-1*, *SHA-256/384* a *AES*. Pro zajištění důvěrnosti dat je pak definováno použití symetrických šifrovacích algoritmů *RC4* (proudová šifra), *IDEA*, *DES*, *3DES* a *AES* (blokové šifry).

3.2.4 Výhody a nevýhody TLS

Mezi hlavní výhody TLS patří především jeho velká rozšířenost a jednoduché používané mechanismy. Protokol nevyžaduje žádnou složitou konfiguraci ani podporu v jádře operačního systému, jako je tomu u protokolů IPsec.

Nevýhodou TLS může být například to, že v současné době v síťových aplikacích umožňuje použití několika rozdílných verzí tohoto protokolu, což může vést při ustanovování relací k volbě slabších a starších kryptografických mechanismů, než těch, které jsou dostupné v nejnovějších standardech. Hlavní slabinou protokolu je však velmi nejasně definovaná forma obnovování klíčů v průběhu ustanovené relace a nejasná podpora tohoto obnovování u existujících implementací. Při příliš dlouhé relaci může neobnovování materiálu sloužícího k výpočtu klíčů nahrávat potenciálnímu útočníkovi, který může komunikaci sledovat, a pokud získá dostatečné množství dat, může se pokusit o provedení statistického útoku na šifrovací algoritmus a následně prolomit zabezpečení. Z tohoto důvodu se protokol TLS zpravidla využívá pouze pro krátká spojení a přenos malého objemu dat.

3.3 Protokol SSH

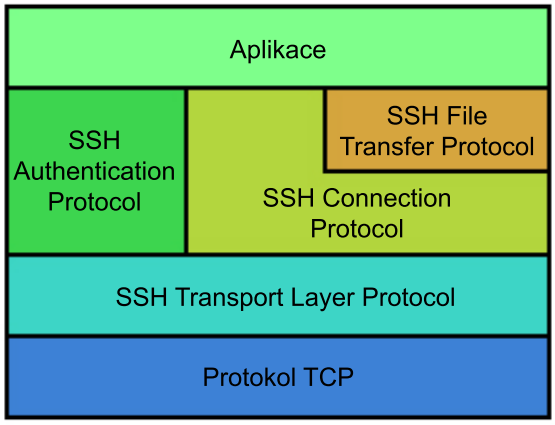
Protokol *Secure Shell – SSH* [40, 55] je jednou z dalších možností zabezpečení dat přenášených počítačovou sítí. Kromě zabezpečení přenosu dat poskytuje také prostředky pro vzdálený přístup. Proto dokáže zajistit nejen autentizaci, důvěrnost a integritu, ale při vzdáleném přístupu také autorizaci vzdáleného uživatele (přidělení oprávnění k přístupu). Protokol se nachází na aplikační vrstvě ISO/OSI modelu a úzce spolupracuje s aplikací, která jeho služeb využívá. Modelem komunikace je obdobně jako u protokolů SSL/TLS model klient-server.

Protokol SSH je v současné době dostupný ve dvou navzájem nekompatibilních verzích SSHv1 a SSHv2, nicméně SSHv1 již není považováno za bezpečné.

[1] *Keyed-hash Message Authentication Code – HMAC* je výsledek kryptografické hašovací funkce aplikované na data, slouží pro zajištění integrity těchto dat. Výpočet je proveden s využitím tajného klíče.

3.3.1 Dílčí protokoly SSH

Protokol SSH se skládá ze třech základních částí, které jsou reprezentovány třemi protokoly - *SSH Transport Layer Protocol* [57], *SSH Authentication Protocol* [56], *SSH Connection Protocol* [58]. Hierarchie těchto protokolů v rámci SSH je znázorněna na obrázku 3.4.



Obrázek 3.4: Architektura protokolu SSH

Kromě uvedených částí existují i další rozšíření protokolu SSH (například protokol *SSH File Transfer Protocol* [21], jehož umístění v rámci hierarchie SSH je taktéž názorně ukázáno na obrázku 3.4). Protokol je záměrně navržen způsobem, kdy není obtížné přidávat další moduly. Jednotlivé části protokolu spolu mohou úzce spolupracovat a navzájem si poskytovat služby. Současně je protokol poměrně rozsáhle konfigurovatelný, což umožňuje vysokou míru interoperability různých aplikací. Podobně jako je tomu u standardu IPsec a protokolu TLS, jsou vlastnosti navázaného spojení dohadovány interaktivně při jeho vytváření.

SSH Transport Layer Protocol

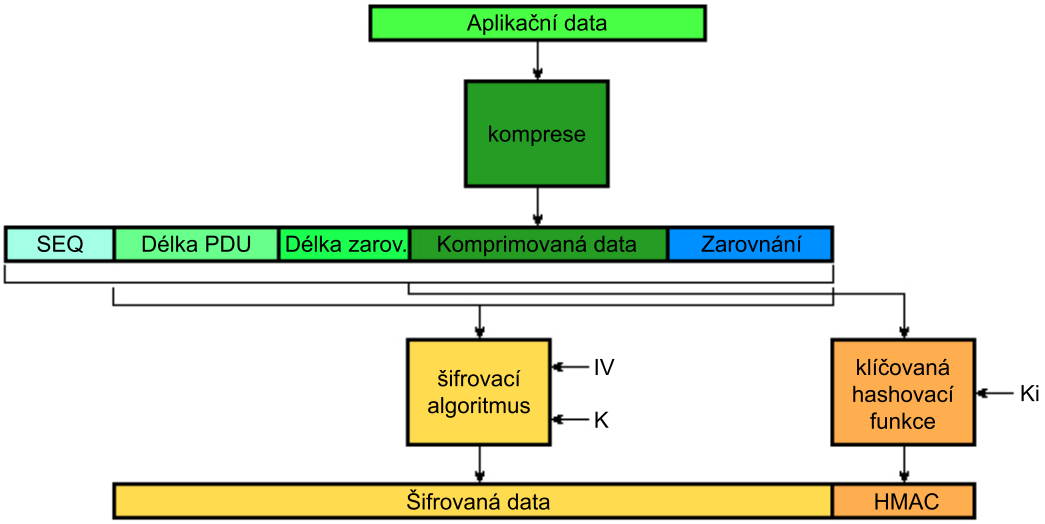
Protokol SSH Transport Layer Protocol slouží pro navázání spojení mezi dvěma aplikacemi a následný přenos dat vyšších vrstev. Během ustanovení spojení dochází k autentizaci serveru i klienta a je provedena výměna šifrovacích klíčů založená na mechanismu Diffie-Hellman.

Ověřování identity serveru je možné dvojím způsobem. Klientská aplikace si může udržovat lokální databázi názvů serverů a jejich veřejných klíčů, případně může využívat ověřování certifikátu serveru, který je podepsán certifikační autoritou. Klient ve druhém případě musí znát kořenový certifikát certifikační autority. Ověření identity klienta se provádí s využitím protokolu SSH Authentication Protocol.

V rámci tohoto protokolu je také definován způsob zabezpečení přenášených dat. Schéma tohoto zabezpečení je znázorněno na obrázku 3.5. Ze schématu je patrné, že je velice podobné schématu zabezpečení protokolem TLS (viz obrázek 3.3). Rozdílem je především to, že protokol SSH provádí kromě šifrování dat také šifrování hlaviček (oproti TLS, který pro přenášené hlavičky zajišťoval pouze integritu).

SSH Authentication Protocol

Protokol SSH Authentication Protocol zajišťuje ověření identity vzdáleného uživatele, aplikace nebo klientské stanice. Pro tento účel nabízí několik možností. Prvním nabízeným



Obrázek 3.5: Schéma zabezpečení dat protokolem SSH

prostředkem je ověření na základě znalosti hesla. Toto heslo je přenášeno po síti v šifrované podobě. Další možností je využití asymetrické kryptografie RSA, systému Kerberos [59], mechanismu S/Key [24] a dalších. Způsob autentizace je dohadován vždy při ustanovování relace mezi komunikujícími uzly. Tento protokol je také po ověření identity schopen přidělit připojovanému klientovi příslušná oprávnění.

SSH Connection Protocol

Činnost protokolu SSH Connection Protocol spočívá v řízení toku dat, multiplexování otevřených kanálů v rámci jednoho spojení, umožňuje přenášet data protokolu X Window, umožňuje vzdáleně spouštět aplikace na serveru a tunelovat výsledky jejich provádění, propagovat signály OS Unix apod. Tyto možnosti jsou připojenému klientovi poskytnuty po ustanovení relace a vzájemné úspěšné autentizaci serveru a klienta.

3.3.2 Podporovaná kryptografie

Protokol SSHv2 podporuje v současné době pro účely autentizace serveru a klienta bohatou paletu mechanismů, z nichž ty používanější byly zmíněny v rámci sekce 3.3.1. Zabezpečení je pak z pohledu integrity dat zajišťováno hašovacími algoritmy *SHA-1* a *MD5*, nově také algoritmus *SHA-2* [11]. Důvěrnost dat pak může být zajištěna skrze symetrické šifrovací algoritmy *RC4, Blowfish, Twofish, 3DES, Serpent, Cast* a *AES*.

3.3.3 Výhody a nevýhody SSH

Protokol SSH nabízí rozsáhlé prostředky pro autentizaci jak serveru, tak klienta, obsahuje bezpečnostní mechanismy, které ošetřují útoky na doménové názvy a zcizení IP adresy. Protokol je také odolný vůči útokům *Man-in-the-Middle* a zcizení probíhajícího spojení. Protokol má jasně definovaný způsob, jakým dochází k průběžné výměně klíčů relace, což případnému útočníkovi znemožňuje provedení statistického útoku. Zároveň protokol zajišťuje šifrování jeho vlastních hlaviček, což umožňuje skrytí více informací o datovém toku,

než jak tomu je například u protokolu TLS. Další výhodou protokolu SSH je jeho srozumitelnost, rozsáhlá konfigurovatelnost a jeho velká rozšířenost (např. nástroje OpenSSH je standardní součástí operačních systémů na bázi Unixu).

3.4 Shrnutí a výběr mechanismu

V rámci této kapitoly byly představeny protokoly, které jsou v současné době běžně dostupné a využívané v rámci přenosu dat počítačovými sítěmi. Z textu je patrné, že prakticky všechny představené protokoly mají mnoho společného a liší se především v jejich umístění v rámci ISO/OSI síťového modelu, a dále z určení jejich použití vyplývají některé další vlastnosti.

Zatímco standard IPsec poskytuje zabezpečení spojení na úrovni síťové vrstvy, což je výhodné především z hlediska transparentnosti vůči aplikaci ale nevýhodné z hlediska náročných požadavků na operační systém, protokoly TLS a SSH pracující na prezentační či aplikační vrstvě ISO/OSI modelu nevyžadují tak rozsáhlou podporu OS na úkor ztráty zabezpečení hlaviček protokolů transportní a síťové vrstvy. Protokoly TLS a SSH jsou hojně využívané a je dostupná řada jejich implementací.

Pokud se na představené vlastnosti jednotlivých protokolů podíváme z pohledu navrhované mikrosondy, je potřeba zhodnotit především nevýhody, které mohou mít vliv na výkonnost mikrosondy. Mikrosonda v současné době neobsahuje výkonný procesor, na kterém bude z tohoto důvodu pracovat pouze odlehčený operační systém na bázi Linuxu. Z tohoto pohledu je hlavní nevýhodou standardu IPsec jeho netriviální konfigurace a nutnost rozsáhlé podpory v jádře operačního systému. Tomuto by se v rámci mikrosondy dalo vyhnout jedině kompletní nebo velmi rozsáhlou hardwarovou realizací standardu IPsec v čipu FPGA, který se na sondě nachází, což není reálné řešení z pohledu dostupných výpočetních zdrojů. V rámci protokolů TLS a SSH není vyžadována podpora OS, tyto protokoly navíc nejsou tak rozsáhlé, jako standard IPsec. Je tedy možná jejich jednodušší hardwarová akcelerace oproti standardu IPsec.

Dále je nutné přihlédnout k faktu, že z mikrosondy se mohou přenášet poměrně rozsáhlé toky dat (nařízený odposlech může trvat i několik dní a po celou dobu je nutné zajistit přenos všech zachycených dat odpovídajících nastavenému filtru). Z tohoto důvodu je nutné, aby byly v rámci zvoleného bezpečnostního mechanismu implementovány prostředky zabraňující útokům typu Man-in-the-Middle, zcizení spojení, podvržení doménových jmen a podobně. Zároveň je naprosto nezbytné, aby zvolený bezpečnostní mechanismus měl prostředky zabraňující statistickému útoku. Tyto vlastnosti splňuje standard IPsec a protokol SSH, které jsou určené pro déle trvající relace. Naproti tomu protokol TLS tento požadavek v současné době není schopen naplnit.

Vzhledem k výše uvedeným faktům bude pro zajištění ochrany dat přenášených z mikrosondy zvolen protokol SSH. V rámci dalšího postupu bude zvolena jeho vhodná implementace a na základě metodiky HW&SW Codesign bude navržena architektura pro prostředí mikrosondy, která umožní hardwarově akcelerovat kritické části softwarové implementace protokolu.

Kapitola 4

Vybrané bezpečnostní mechanismy

V kapitole 3.3.2 bylo vyjmenováno několik kryptografických algoritmů, jejichž použití je pro protokol SSH standardizováno. K jejich použití však nejsou ve zmíněné kapitole uvedené žádné podrobnosti. Tato kapitola se snaží o uvedení souvislostí, které jsou nezbytné pro budoucí návrh hardwarové akcelerační jednotky. Stručně představuje režimy činnosti blokových šifer, které jsou standardizovány pro použití v rámci transportní vrstvy protokolu SSH, a dále vysvětluje princip činnosti hashovacích funkcí v režimu HMAC.

4.1 Vybrané režimy činnosti blokových šifer

Základní princip každého šifrovacího algoritmu je převod vstupního otevřeného textu na text šifrovaný za použití utajeného klíče. U blokových šifer dochází k rozdělení zprávy do bloků fixní velikosti, které jsou algoritmem šifrovány každý jednotlivě. V rámci tohoto textu se bude pod pojmem bloková šifra myslet výhradně symetrický blokový šifrovací algoritmus.

Výše popsaný základní princip je označován jako *Electronic Code Book – ECB*. Šifrování jednotlivých bloků není nijak závislé na ostatních blocích. Toto chování však není příliš bezpečné, například kvůli tomu, že shodný vstup je převeden na shodný výstup, nebo proto, že potenciální útočník může provádět záměnu pořadí jednotlivých bloků či jejich náhradu. Proto jsou zavedené takzvané režimy činnosti blokových šifer, což jsou způsoby použití šifrovacího algoritmu zajišťující dosažení různých bezpečnostních cílů. Tyto režimy mohou z blokové šifry vytvořit například generátor náhodných čísel nebo zajistit autentizaci přenášené zprávy.

U každého kryptografického algoritmu definovaného v odpovídajících standardech protokolu SSH [57, 10, 29, 28] je rovněž uveden režim, ve kterém má být daný algoritmus použit. Příkladem může být například algoritmus AES, který je definován hned v několika režimech. Režimy činnosti *Cipher Block Chaining – CBC*, *Counter Mode – CTR* a *Galois Counter Mode – GCM* budou stručně představené v této části práce.

4.1.1 Režim Cipher Block Chaining

Princip činnosti režimu CBC [14] lze popsat následovně. Mějme otevřený text zprávy P, který lze rozdělit na n bloků datové šířky požadované blokovou šifrou, bloky označme jako $P_0, \ldots, P_{n-1}$. Šifrovací funkce $E_K(P_i) = C_i$ pak provede šifrování bloku P_i na blok C_i s využitím tajného klíče K. Dešifrování je provedeno symetricky, kde vstupním blokem šifrovací funkce je šifrovaný text a jejím výsledkem je otevřený text. V režimu CBC je

šifrování prováděno dle následujících vztahů:

$$C_0 = E_K(P_0 \oplus IV) \tag{4.1}$$
$$C_i = E_K(P_i \oplus C_{i-1}), \qquad \text{pro } i \in \{1, \ldots, n-1\} \tag{4.2}$$

Ve vzorci 4.1 figuruje hodnota označená jako IV – inicializační vektor. Tento blok slouží jako výchozí hodnota pro výpočet prvního šifrovaného bloku zprávy. Inicializační vektor by měl být volen náhodně a pro každou zabezpečovanou zprávu by měl být zvolen jiný. Dešifrování zprávy je symetrické k šifrování:

$$P_0 = E_K(C_0 \oplus IV) \tag{4.3}$$
$$P_i = E_K(C_i \oplus C_{i-1}), \qquad \text{pro } i \in \{1, \ldots, n-1\} \tag{4.4}$$

Z uvedených vztahů je patrné, že při šifrování zprávy dochází k závislosti na předchozích šifrovaných blocích. Toto chování způsobuje, že dva shodné bloky nebudou šifrovány na stejný výstup. Rovněž tento režim zamezuje útočníkovi přeskládat pořadí bloků, či provést náhradu jednoho bloku za jiný.

V rámci protokolu SSH je definováno, že přenášená data jsou považována za právě jednu zprávu. Díky tomuto není nutné měnit inicializační vektor při každé přenášené zprávě, nicméně je doporučeno provést výměnu inicializačních vektorů po přenesení jednoho gigabajtu dat nebo po jedné hodině spojení [57].

Princip tohoto režimu neumožňuje při šifrování provádět paralelizaci kvůli závislostem na předchozích blocích zprávy. Naproti tomu proces dešifrování paralelizovatelný je. Tento režim se často objevuje v softwarových implementacích šifrovacích algoritmů nejen protokolu SSH. Mezi algoritmy, které jsou pro protokol SSH definovány v tomto režimu, patří například AES, 3DES, Blowfish a další.

4.1.2 Režim Counter Mode

Režim CTR, definovaný v [14], přistupuje k problémům režimu ECB odlišným způsobem od CBC. Tento režim prakticky přináší transformaci blokové šifry v šifru proudovou[1] tím, že vstupem šifrovacího algoritmu není otevřený text, nýbrž průběžně inkrementovaná hodnota čítače. Výstupem šifrování je pak pseudonáhodný řetězec, na který je spolu s otevřeným textem aplikována logická operace xor. Po každém šifrovaném bloku je čítač inkrementován. Vztahy pro šifrování a dešifrování jsou následující:

$$C_i = P_i \oplus E_K(IV + i), \qquad \text{pro } i \in 0, \ldots, n-1 \tag{4.5}$$
$$P_i = C_i \oplus E_K(IV + i), \qquad \text{pro } i \in 0, \ldots, n-1 \tag{4.6}$$

Podobně jako u režimu CBC, i u tohoto režimu figuruje inicializační vektor IV, který zde slouží jako počáteční hodnota čítače. Rovněž, jako u režimu CBC, by měl být inicializační vektor volen náhodně a je nezbytné zajistit jeho průběžnou obnovu v rámci ustanovené relace. Definice použití a požadavky na tento režim příslušných algoritmů pro protokol SSH jsou uvedené v [10].

Princip činnosti tohoto režimu umožňuje jeho plnou paralelizovatelnost, případně využití principů zřetězení blokového šifrovacího algoritmu. Díky tomu je tento režim schopen dosahovat vyšších rychlostí zpracování dat, nežli je tomu u režimu CBC. Mezi algoritmy definované v tomto režimu ve standardech protokolu SSH patří AES, 3DES, Blowfish a další.

[1] *Proudová šifra* je typ symetrické šifrovací funkce, který provádí šifrování jednotlivých bitů otevřeného textu jejich kombinací s pseudonáhodnou bitovou posloupností zpravidla s využitím logické operace xor.

4.1.3 Režim Galois/Counter Mode

Režim činnosti GCM, definovaný organizací NIST [43, 15], přináší zcela odlišný přístup od výše zmíněných režimů. Tím je takzvané *autentizované šifrování* a *autentizované dešifrování* (*authenticated encryption*, *authenticated decryption*). Režim totiž současně zajišťuje jak důvěrnost, tak integritu/autentizaci důvěrných dat. K tomu využívá univerzální hashovací funkce definované nad binárním Galoisovým polem $GF(2^{128})$ s generujícím polynomem $x^{128} + x^7 + x^2 + x + 1$. Režim GCM rovněž obsahuje prostředky pro zajištění integrity nešifrovaných dat (například hlaviček některých protokolů), pokud je to vyžadováno.

Tento režim byl navržen tak, aby mohl zajišťovat důvěrnost a integritu na vysokých rychlostech jak v softwarových, tak v hardwarových implementacích.

Šifrování

Šifrování v tomto režimu se prakticky žádným způsobem neliší od šifrování v režimu CTR, který byl popsán v sekci 4.1.2. Počáteční hodnota k-bitového čítače je určena z hodnoty inicializačního vektoru tak, že horních $(k-32)$ bitů (záleží na zvolené blokové šifře) je nastaveno na hodnotu inicializačního vektoru, a dolních 32 bitů je nastaveno na nulu. V průběhu šifrování dochází k inkrementaci pouze spodních 32 bitů.

Hashování

Výpočet integritního součtu v rámci režimu GCM je založen na operaci násobení polynomů v Galoisově poli. Polynomy jsou zde vyjádřeny pomocí bitových bloků, proto dále bude tato operace označována jako násobení bloků. Tuto operaci lze popsat algoritmem 1 [15].

Algoritmus 1 Operace násobení $\bullet$ bloků X, Y v Galoisově poli $GF(2^{128})$ s generujícím polynomem $x^{128} + x^7 + x^2 + x + 1$

```
1:  X, Y                                          ▷ Vstupní 128-bitové bloky X a Y
2:  R ← 11100001 ‖ 0^120                          ▷ 128-bitový řetězec, ‖ značí konkatenaci
3:  X = x_0 x_1 ... x_127
4:  Z_0 ← 0^128
5:  V_0 ← Y
6:  for i ← 0, ..., 127 do
7:      if x_i = 0 then
8:          Z_{i+1} ← Z_i
9:      else
10:         Z_{i+1} ← Z_i ⊕ V_i                   ▷ pro x_i = 1
11:     end if
12:     if LSB(V_i) = 0 then                      ▷ LSB = Least Significant Bit
13:         V_{i+1} ← V_i ≫ 1
14:     else
15:         V_{i+1} ← (V_i ≫ 1) ⊕ R               ▷ pro LSB(V_i) = 1
16:     end if
17: end for
18: return Z_128                                  ▷ 128-bitový výstupní blok
```

Na operaci násobení $\bullet$ je založen výpočet hashovací funkce *GHASH*, která popsaná algoritmem 2 [15]. Z algoritmu na výpočet hashe zprávy X je patrná jistá podobnost k režimu

Algoritmus 2 Operace GHASH nad blokem X s hashovacím klíčem H

1: H ▷ 128-bitový blok H, hashovací klíč
2: X ▷ Vstupní zpráva X, jejíž bitová délka je násobkem 128
3: $X = X_0 \parallel X_1 \parallel \cdots \parallel X_{m-1}$, délka$(X) = 128m$.
4: $Y_0 \leftarrow 0^{128}$ ▷ 128-bitový nulový blok
5: **for** $i \leftarrow 0, \ldots, m-1$ **do**
6: $\quad Y_i \leftarrow (Y_{i-1} \oplus X_i) \bullet H$
7: **end for**
8: **return** Y_m

CBC blokových šifer, přičemž šifrovací algoritmus je zde nahrazen operací násobení $\bullet$. Oproti blokovým algoritmům je však operace $\bullet$ výpočetně nenáročná a existují metody na její optimalizaci jak pro software, tak pro hardware [43].

Je důležité podotknout, že operace $GHASH$ se svou podstatou blíží spíše výpočtu kontrolních součtů (např. CRC32) a nejedná se v žádném případě o kryptografickou hashovací funkci. Proto nesmí být využita mimo režim GCM. Důvody, proč tomu tak je, a také proč je použití této hashovací funkce v rámci režimu GCM dostatečně bezpečené, se zabývá dokument [15].

Celkové schéma režimu GCM

Autentizované šifrování umožňuje souběžně využitím blokové šifry v režimu CTR zajištění důvěrnosti dat, funkce $GHASH$ popsaná algoritmem 2 pak umožňuje výpočet integrity jak šifrovaných dat, tak libovolných dat, která jsou součástí zprávy, ale mají být přenášena v otevřené podobě. Výstupem režimu GCM jsou autentizovaná data přenášená v otevřené podobě, šifrovaná data a vypočtený integritní součet označovaný jako *authentication tag*. Činnost režimu GCM je popsána algoritmem 3 [15].

Algoritmus 3 Princip činnosti režimu GCM

1: Blokový šifrovací algoritmus $E_K(x)$ s velikostí bloku 128 bitů
2: K ▷ Šifrovací klíč K
3: IV ▷ Inicializační vektor
4: P ▷ Text v otevřené podobě k zašifrování
5: A ▷ Autentizovaná data přenášená v otevřené podobě
6: $H \leftarrow E_K(0^{128})$ ▷ Výpočet hashovacího klíče
7: $J_0 = IV \parallel 0^{31} \parallel 1$ ▷ Počáteční obsah čítače
8: $C_i = P_i \oplus E_K(J_0 + i)$, pro $i \in \{0, \ldots, n-1\}$, n počet bloků P ▷ Šifrování
9: $u \leftarrow$ počet bitů zarovnání posledního neúplného autentizovaného bloku
10: $v \leftarrow$ počet bitů zarovnání posledního neúplného šifrovaného bloku
11: $S \leftarrow GHASH_H(A \parallel {}^u \parallel C \parallel 0^v \parallel$ délka A $\parallel$ délka C)
12: $T \leftarrow S \oplus E_K(J_0)$ ▷ Výpočet integritního součtu
13: **return** (A, C, T)

Autentizované dešifrování probíhá symetrickým způsobem, s tím rozdílem, že na závěr výpočtu je u zprávy porovnán vypočtený integritní součet dešifrované zprávy s přijatým integritním součtem. V případě, že se odlišují, je identifikováno, že došlo k chybnému přenosu zprávy nebo jejímu úmyslnému narušení.

Uvedení režimu GCM do protokolu SSH

Režim GCM je pro protokol SSH určen pro použití s jediným algoritmem a tím je AES [29]. Zavedení tohoto režimu mírně upravuje formát datagramu transportní vrstvy protokolu SSH a to tím způsobem, že 32 bitové pole hlavičky obsahující délku zprávy není v tomto režimu šifrované, ale je u něj zajištěna integrita. Proč toto chování nepředstavuje bezpečnostní problém, je popsáno v [29].

V současné době není tento režim implementovaný v dostupných softwarových knihovnách s výjimkou řešení OpenSSH verze 6.2 [6].

4.2 Hashovací funkce v režimu HMAC

Klíčovaná hashovací funkce je způsob použití kryptografických hashovacích funkcí, který byl navržen za účelem zvýšit jejich bezpečnost tím, že data budou hashována s využitím tajného hashovacího klíče. V praxi to znamená, že datové bloky vstupují do hashovací funkce až poté, co je provedena inicializace funkce blokem, který v jisté formě obsahuje hashovací klíč. Tím je zajištěno, že pro shodná data a různé klíče bude mít hashovací funkce vždy jiný výstup. Schéma hashování lze vyjádřit vztahem 4.7.

$$HMAC_K(m) \quad = \quad h((K \oplus opad) \parallel h((K \oplus ipad) \parallel m)) \tag{4.7}$$

V uvedeném vztahu je klíčovaná hashovací funkce označena jako $HMAC_K(m)$, m je zabezpečovaná zpráva. $h(m)$ je pak libovolná kryptografická hashovací funkce (například MD5, SHA-1 a další). Klíč K je xorován s bloky *opad* a *ipad*, které mají velikost právě jeden blok zvolené hashovací funkce (např. pro MD5 je blok velký 512 bitů). Jejich hodnota je pak stanovena v případě *opad* na `0x5c5c...5c`, v případě *ipad* pak na `0x3636...36`.

HMAC nezvyšuje pouze bezpečnost použité kryptografické hashovací funkce, ale zvyšuje výpočetní náročnost hashovací funkce. Inicializační hodnoty lze za předpokladu, že jsou jsou používané opakovaně, předpočítat.

Pro protokol SSH jsou v současné době definovány klíčované hashovací funkce založené na algoritmech MD5, SHA-1 a rodině hashovacích funkcí SHA-2.

Kapitola 5

Srovnání dostupných implementací protokolu SSH

Protože protokol SSH je poměrně rozšířeným a hojně využívaným protokolem poskytujícím prostředky pro zabezpečení přenosu dat, je snahou mnoha vývojářů implementovat požadavky definovaných norem, ať už za účelem komerčního zisku nebo zvýšení bezpečnosti nejen volně distribuovaných operačních systémů. V rámci této kapitoly budou srovnány volně dostupné implementace protokolu SSH za účelem zjištění, které algoritmy definované standardem jsou běžně dostupné v těchto implementacích, což je důležité znát z pohledu kompatibility vytvářeného řešení s ostatními. Dalším cílem je případně vybrat takovou implementaci protokolu SSH, která bude realizovat nejnovější požadavky na tento protokol a mohla by být použita jako základ pro implementaci softwarové části akcelerovaného systému. Pokud má být zvolená implementace hardwarově akcelerovatelná, je vhodné, aby byla jasně a srozumitelně dokumentovaná. Je taktéž nezbytné, aby byla přeložitelná pro operační systém, který bude na mikrosondě dostupný, tedy pro distribuce OS Linux.

5.1 Dostupné implementace protokolu SSH

V rámci této sekce budou stručně představeny dostupné knihovny implementující rozhraní a prostředky protokolu SSH. Kromě knihoven byly prověřené také implementace, které nejsou přímo knihovnami, ale reprezentují již hotové řešení a aplikace postavené na protokolu SSH. Podrobné srovnání dostupných informací je uvedené v příloze A, vynesené do tabulek A.1, A.2, A.3, A.4, A.5 a A.6.

- **Knihovna *libssh*** [2] je multiplatformní, volně dostupnou knihovnou implementující prostředky pro klienta i server protokolu SSH. Knihovna je implementována v programovacím jazyce C. Je průběžně aktualizována vzhledem k měnícím se standardům protokolu SSH.
- **Knihovna *libssh2*** [3] je knihovnou implementující pouze prostředky pro implementaci klienta protokolu SSH. Nabízí velký počet kryptografických mechanismů a snaží se držet definovaných norem pro protokol SSH. Je implementována v jazyce C/C++.
- **Knihovna *FlowSsh*** [1] je knihovna implementovaná v jazyce C/C++ a .NET, určená pro operační systémy Microsoft Windows. Knihovna je dostupná pouze pod komerční licencí.

- **Nástroje *OpenSSH*** [6] jsou standardní součástí operačních systému založených na UNIXu. Jedná se již o implementované aplikace zajišťující služby protokolu SSH. Dokumentace je součástí manuálových stránek operačního systému. Bohužel pro tuto implementaci v současné době neexistuje dostupné rozhraní API.
- **Knihovna *Netsieben SSH*** [4] je implementována v jazyce C/C++, implementuje pouze klientskou část protokolu SSH. Je dostupná pod volnou či komerční licencí v závislosti na tom, zda nad ní implementovaná aplikace nemá nebo má být komerčním produktem. Knihovna nepodporuje kompresi dat. Poslední dostupná verze je z roku 2009 a tedy tato knihovna neplní aktuálně platné standardy.
- **Řešení *NanoSSH*** [44] je kompletní implementací určenou pro vestavěné systémy. Jedná se o komerční produkt s podporou mimo běžně dostupných operačních systémů také pro real-time operační systémy (např. Freescale MQX™ RTOS). V závislosti na požadavcích zákazníka obsahuje licence prostředky pro realizaci klienta, serveru nebo klienta a serveru protokolu SSH. Řešení sleduje aktuálně platné požadavky na protokol SSH, nicméně cena úprav tohoto řešení pro využití na mikrosondě byla vyčíslena na nereálně vysokou.
- **Knihovna *Chilkat SSH*** [12] je další komerční knihovnou, která poskytuje pravděpodobně nejrozsáhlejší nabídku dostupných API (např. pro C#, C/C++, Java, Perl, Python, Ruby a další). Bohužel se k této knihovně nepodařilo zjistit přesné informace o tom, jaké prostředky pro zajištění bezpečnosti skutečně poskytuje.
- **Knihovna *paramiko*** [5] je zajímavou implementací protokolu SSH v jazyce Python (verze 2.3 nebo vyšší). Knihovna je volně dostupná a nabízí poměrně velké množství kryptografických algoritmů, neboť využívá prostředků knihovny *pycrypto2.1+*.

Řešení *OpenSSH* bylo z dalšího porovnání vyřazeno, protože v současné době nemá dostupné rozhraní API. Rovněž hotové komerční řešení *NanoSSH* pro vestavěné systémy bylo z porovnání vyřazeno z důvodu příliš vysoké ceny.

5.2 Srovnání

V rámci porovnávání bylo pohlíženo na realizovanou verzi protokolu SSH, na implementované kryptografické algoritmy pro šifrování, zajištění integrity dat a autentizace komunikujících stran. Současně bylo zjišťováno, zda knihovna podporuje či nepodporuje kompresi přenášených dat, která může být užitečná především pro přenos velkého množství dat. Mezi další kritéria patřila dostupnost knihovny z pohledu licence, podporovaného operačního systému a zvoleného programovacího jazyka. Dále bylo posuzováno, zda je dostupná dokumentace rozhraní knihovny, zda jsou dostupné příklady jejího použití a zda je knihovna aktuální.

Na základě srovnávaných kritérií bylo zjištěno, že implementace, které poskytují API, neimplementují nové RFC [11] definující způsob využití algoritmu SHA-2 pro zajištění integrity přenášených dat. Namísto toho jsou běžně používané hashovací funkce MD5 a SHA-1. Nejčastějšími šifrovacími algoritmy v daných implementacích jsou pak algoritmy Blowfish, 3DES a AES. Z pohledu režimu činnosti je pak nejběžnějším režim CBC, o něco méně pak režim CTR. Žádná ze srovnávaných knihoven neimplementuje RFC [29] definující použití algoritmu AES v režimu GCM (oproti tomu řešení OpenSSH postrádající API v nejnovější verzi 6.2 ano).

Pokud by byla hardwarová akcelerační jednotka založena na běžně dostupných algoritmech, pak se jako nejlepší základ pro softwarovou část systému jeví knihovna *libssh2*, dostupná pod licencí BSD. Knihovna neimplementuje serverovou část protokolu SSH, což ovšem nepředstavuje problém, protože mikrosonda se z principu chová jako klient připojovaný k mediačnímu zařízení, které z tohoto pohledu reprezentuje server (k mediačnímu zařízení může být současně připojeno i více mikrosond).

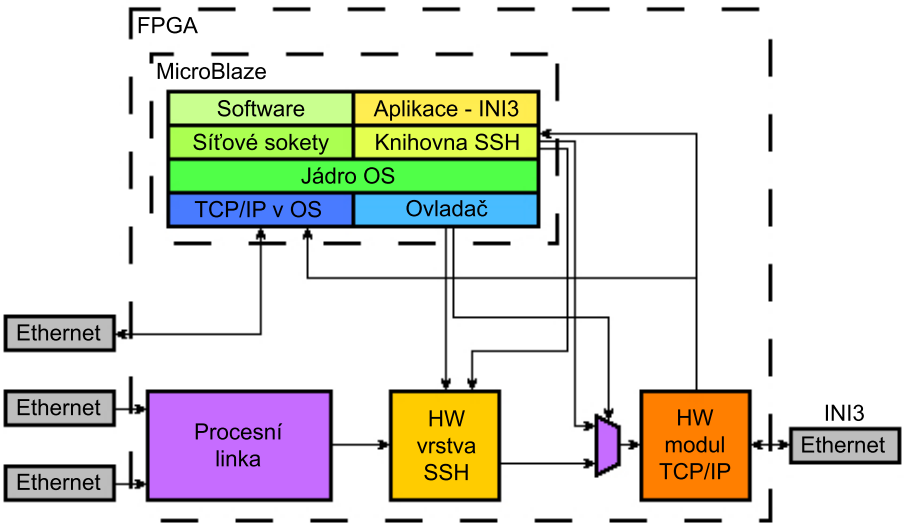
Dalšími zajímavými knihovnami jsou také *libssh* a *paramiko*, jejichž prostředky jsou srovnatelné s knihovnou *libssh2*. Knihovna *libssh* obsahuje také implementaci algoritmu AES v režimu činnosti CTR, což je její výhoda oproti ostatním zmíněným knihovnám.

Kapitola 6

Návrh zabezpečené komunikace

V rámci této kapitoly je představen návrh hardwarové akcelerace zvoleného bezpečnostního mechanismu. Kapitola lehce naznačuje možnosti zasazení hardwarově akcelerované softwarové implementace protokolu SSH do kontextu mikrosondy. Následně se kapitola podrobněji zaměřuje na zevrubný návrh architektury hardwarové akcelerační jednotky a rozbor jejích možných variant s následnou volbou varianty, která je pro použití na mikrosondě nejvhodnější.

Orientační architektura HW&SW Codesignu bezpečnostního mechanismu je znázorněna na obrázku 6.1.



Obrázek 6.1: Zjednodušený návrh hardwarové akcelerace protokolu SSH

Z obrázku 6.1 je patrné, že hardwarově akcelerovaná knihovna implementující protokol SSH bude sloužit výhradně pro potřeby zabezpečení dat vycházejících z procesní linky na mikrosondě (a tím pro realizaci přenosu dat přes rozhraní INI3). Navržené řešení se skládá z upravené knihovny implementující protokol SSH a nad touto knihovnou postavenou aplikací, která bude zajišťovat ustavení relace mezi mikrosondou a mediačním zařízením na rozhraní INI3 a správu této relace. Dále bude zapotřebí vytvořit v rámci jádra operačního systému ovladač, skrze který bude moci knihovna řídit a konfigurovat akcelerační hardwarovou výpočetní jednotku.

Činnost celého systému by měla být následující:

1. Aplikace nad knihovnou SSH provede ustanovení relace mezi mikrosondou a mediačním zařízením.
2. Po ustanovení relace se provede konfigurace hardwarové jednotky (nahrání platného šifrovacího klíče, inicializačního vektoru a podobně), která urychluje výpočetně náročné části knihovny (zajištění integrity a důvěrnosti dat procesní linky). Tímto se zamezí přenosu dat z procesní linky do procesoru MicroBlaze, jejich pomalému softwarovému zpracování. Po konfiguraci je činnost hardwarové jednotky spuštěna.
3. Aplikace během existence relace zajišťuje průběžnou výměnu klíčů relace dle specifikace protokolu SSH a dynamicky pozastavuje, rekonfiguruje a znovuspouští činnost hardwarové akcelerační jednotky.

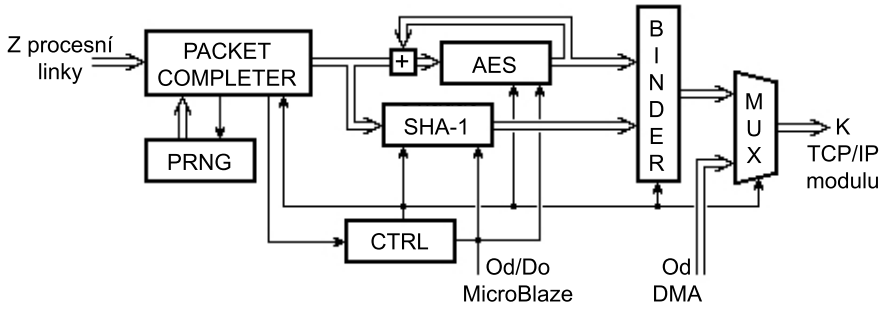
Aplikace provádějící zmíněné řízení přenosu dat přes rozhraní INI3 bude pravděpodobně součástí aplikace komunikující po rozhraní CCCI, a její činnost bude řízena v souladu s požadavky na realizovaný odposlech.

6.1 Varianty hardwarové akcelerační jednotky

V průběhu tvorby této práce byly navržené tři varianty hardwarové akcelerační jednotky, jejichž rozbor je uveden v této části kapitoly. Všechny varianty mají společný výběr šifrovacího algoritmu, kterým je algoritmus AES. Varianty se odlišují především režimem činnosti šifrovacího algoritmu a zvoleným řešením zajištění integrity přenášených dat. U jednotlivých variant jsou uvedená jejich pozitiva a negativa. Na závěr je vybrána varianta vyhovující použití na mikrosondě v rámci systému pro zákonné odposlechy, která byla dále rozpracována a implementována.

6.1.1 Varianta AES-CBC a SHA-1

Na obrázku 6.2 je znázorněna první navržená varianta hardwarové akcelerační jednotky pro zabezpečení dat při přenosu z cílového zařízení. Data vstupující do jednotky jsou nejprve zpracována procesní jednotkou *PACKET COMPLETER*, která má za úkol uvést data do správného formátu, tedy vytvořit korektní zprávu protokolu SSH. Tím je zajištěno, že data určená pro další zpracování jsou zapouzdřena příslušnou hlavičkou a zarovnána na správnou velikost, se kterou umí pracovat další komponenty akcelerační jednotky. Protože dle standardu [57] mají být bajty zarovnání náhodné, je k jednotce *PACKET COMPLETER* připojen generátor pseudonáhodných dat, na obrázku označený jako *PRNG*.



Obrázek 6.2: Architektura akcelerační jednotky, varianta AES-CBC a SHA-1

Dále data pokračují z jednotky *PACKET COMPLETER* do dvou paralelních bloků, bloku šifrovacího algoritmu AES, který zprávu protokolu SSH zašifruje, a bloku hashovací funkce SHA-1, která zajistí výpočet integrity dat. Algoritmus AES je v případě této jednotky zapojen v režimu CBC, popsaném v sekci 4.1.1. Je rovněž důležité podotknout, že funkce SHA-1 zde vystupuje jako klíčovaná hashovací funkce HMAC. Protože výsledek výpočtu blokem SHA-1 musí být připojen na konec šifrovaných dat z bloku AES, je do systému zapojena komponenta *BINDER*, která provede korektní spojení těchto dvou částí. Tímto je zpráva protokolu SSH kompletní a připravena k odeslání.

Protože protokol SSH obsahuje ještě zprávy vyšších vrstev a protože tyto zprávy slouží jako řídicí (např. pro ustanovení relace) a budou v rámci systému přenášené ze softwaru, je v architektuře 6.2 znázorněn taktéž multiplexor označený jako *MUX*, který provádí přepínání mezi přenosem zpráv protokolu SSH ze softwaru a z akcelerační jednotky.

Celá jednotka je řízena řídicím konečným automatem *CTRL*, který má především úlohu synchronizace jednotky se softwarem, její konfiguraci, a synchronizaci paralelních bloků jednotky tak, aby celý proces zabezpečení proběhl korektním způsobem.

Poznámky k variantě

Jednotka v této variantě vychází z analýzy dostupných softwarových implementací v kapitole 5, především z běžnosti implementací uvedených algoritmů AES a SHA-1. Cílem varianty je ukázat variantu řešení, která zachovává co největší kompatibilitu s existujícími implementacemi zabezpečení komunikace protokolem SSH.

Hlavní nevýhodou varianty je její nízká propustnost. Pokud budou uvažovány pouze jednotky AES v režimu CBC a SHA-1 v režimu HMAC při frekvenci hodinového signálu 100 MHz, pak pro zprávy délky 48 bajtů dosahuje blok AES propustnosti 852,5 Mb/s a propustnost bloku SHA-1 je dokonce pouhých 199 Mb/s. Nízká propustnost u bloku AES je způsobena zvoleným režimem činnosti CBC a kvůli nutným zpětným závislostem nemůže být v rámci systému rychlejší. Nízká propustnost u bloku SHA-1 je pak způsobena tím, že na jeden 512 bitový blok připadá 80 taktů výpočtu (což je cca čtyřikrát více, než u bloku AES), přičemž v režimu HMAC se výpočet protahuje o zpracování minimálně dvou dalších bloků (a tím 160 taktů), a to i za předpokladu, že je funkce HMAC pro opakující se bloky předpočítána.

Kvůli nízké propustnosti se tato varianta stává nereálnou pro využití v rámci cílového systému a je proto potřeba se uchýlit k jiné, rychlejší variantě.

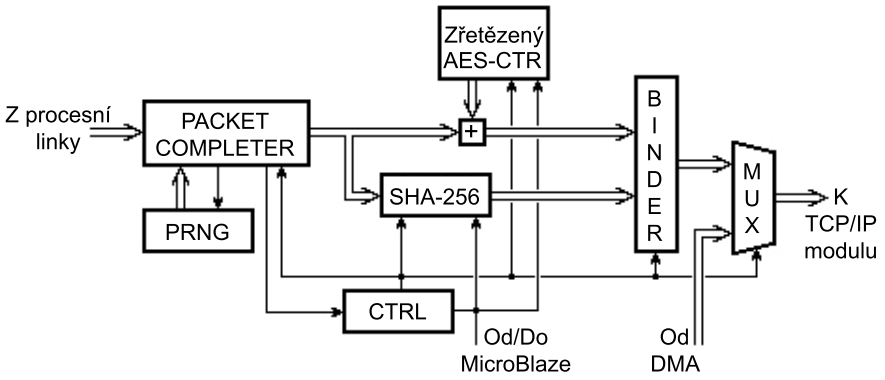
6.1.2 Varianta AES-CTR a SHA-2

Druhá navržená varianta architektury akcelerační jednotky vychází z první varianty. Na obrázku 6.3 mají znázorněné jednotky *PACKET COMPLETER*, *PRNG*, *CTRL*, *BINDER* a *MUX* shodnou funkci, jako u první varianty. Rozdíl v architekturách spočívá ve volbě jiného režimu činnosti algoritmu AES, kterým je režim CTR. Dále je funkce SHA-1 pro výpočet integrity dat nahrazena funkcí SHA-256, která provádí své výpočty méně taktů, než funkce SHA-1.

Poznámky k variantě

Tato varianta je založena na algoritmech, které se stále, i když již ne tak často, vyskytují v existujících implementacích protokolu SSH. Cílem varianty je nahradit pomalé algoritmy

zvolené v předcházející variantě jejich rychlejšími kolegy, i za cenu ztráty kompatibility s valnou většinou softwarových řešení.



Obrázek 6.3: Architektura akcelerační jednotky, varianta AES-CTR a SHA-2

Jak již bylo zmíněno při popisu režimu činnosti blokových šifer CTR v sekci 4.1.2, je režim CTR plně paralelizovatelný a zřetězitelný. Díky tomuto se šifrování na frekvenci hodinového signálu 100 MHz dostává na maximální teoretickou propustnost 12,8 Gb/s, což více než desetinásobně splňuje požadavek cílového systému a dává prostor pro možné optimalizace z hlediska úspory zdrojů. Oproti tomu funkce SHA-256 je stále pomalá a v tomto zapojení dosahuje teoretické propustnosti pouhých 249,5 Mb/s, což není ani řádové zrychlení oproti předchozí variantě.

Ačkoli algoritmus AES v režimu CTR dosahuje pro šifrování dostatečně vysoké propustnosti, funkce SHA-256 v režimu HMAC nedokáže požadavek na propustnost systému stále naplnit a proto je tato varianta nedostatečná. Nedostatek pomalé funkce SHA-256 by se dal řešit například přepínáním mezi více jednotkami SHA-256 a vytvořením transakčního systému s uchováváním šifrovaných transakcí v dostatečně velkých bufferech. Příklad takového zapojení dosahujících vysoké propustnosti je uveden pro funkci SHA-1 například v [41]. Po konzultacích v rámci projektu Sec6Net se tento směr ukázal jako nerealizovatelný především z pohledu dostupnosti zdrojů na cílovém čipu.

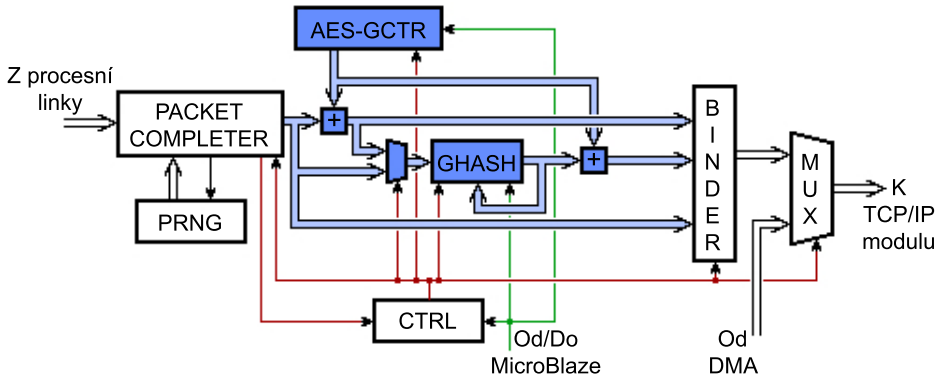
6.1.3 Varianta AES-GCM

Protože ani předchozí varianta nebyla dostačující a rovněž cílový systém nedisponuje prostředky pro komplikovanější architekturu, je potřeba zvolit jiné řešení. Jako toto řešení se nabízí režim činnosti blokových šifer GCM, který je blíže popsaný v sekci 4.1.3. Tento režim zajišťuje autentizované šifrování a dosahuje vysokých propustností [22, 15]. Režim je pro protokol SSH standardizován v [29].

Náhled na architekturu s využitím režimu GCM je znázorněn na obrázku 6.4. Nově zvolený režim vyžaduje drobné odlišnosti i v ostatních modulech jednotky. Jednotka *PACKET COMPLETER* musí zajistit označení části zprávy protokolu SSH, která se nebude šifrovat. Komponenta *BINDER* pak musí svázat položky zprávy do správného pořadí tak, aby otevřené položky hlavičky byly na prvním místě, následované šifrovanými daty a integritním součtem.

Šifrování v režimu GCM je znázorněno blokem *AES-GCTR*. Tento blok reprezentuje algoritmus AES v režimu CTR, generující pseudonáhodný řetězec, na který je spolu se šifrovaným textem aplikována operace xor. Výpočet integritního součtu v režimu GCM je

zde reprezentován blokem *GHASH*, který vykonává stejnojmennou funkci dříve popsanou algoritmem 2. Modře vyznačené části na obrázku 6.4 pak odpovídají realizaci algoritmu 3.



Obrázek 6.4: Architektura akcelerační jednotky, varianta AES-GCM

Na řídicí automat *CTRL* je v tomto režimu kladen mnohem větší důraz v ohledu synchronizace jednotek. Konfigurace jednotky ze softwaru (nahrání šifrovacího klíče a inicializačního vektoru) je prakticky totožná s předchozími variantami.

Poznámky k variantě

Varianta je založena velice rychlém režimu činnosti GCM, který dosahuje vysokých propustností. Implementace architektury se může inspirovat u realizace na procesorech Intel [22, 23], kde byla tato varianta také osvojena. Díky souběžnému výpočtu šifrování i integrity dat lze při frekvenci 100 MHz hodinového signálu teoreticky dosáhnout propustnosti až 12,8 Gb/s, což je velice výrazné zrychlení, oproti předchozím variantám, při zachování zhruba stejné spotřeby zdrojů!

Hlavní nevýhodou této architektury je především nekompatibilita s valnou většinou dostupných softwarových implementací (výjimkou je nejnovější OpenSSH verze 6.2 [6]), protože režim GCM je pro protokol SSH standardizován poměrně nově. Nicméně zavedení instrukční sady procesorů Intel [22] podporující tento režim naznačuje, že se jedná o velice perspektivní režim činnosti blokových šifer. Po konzultacích v rámci projektu Sec6Net bylo toto řešení zvoleno pro cílový systém jako nejvhodnější, i za cenu vzniklé nekompatibility, která povede na složitější implementaci softwarové části.

Kapitola 7

Návrh a implementace

Tato kapitola popisuje návrh a implementaci akcelerační jednotky protokolu SSH. Nejprve je představeno rozhraní jednotky v sekci 7.1, dále je v sekci 7.2 představena architektura celé jednotky, jejíž komponenty jsou dále popsány v sekci 7.3. Popis adresového prostoru jednotky a popis příkazů pro řízení jednotky je pak popsán v sekci 7.4.

Implementace akcelerační jednotky a jejích komponent je realizována v jazyce VHDL a probíhala v prostředí Xilinx ISE Design Suite 14.1 [53], jehož součástí jsou i příslušné překladové nástroje.

7.1 Rozhraní jednotky

Rozhraní navrhované akcelerační jednotky je tvořeno tak, aby bylo pokud možno univerzální a použitelné i mimo použití na mikrosondě, pokud to bude potřeba. Součástí rozhraní je port CLK, určený pro připojení hodinového signálu (v rámci mikrosondy na frekvenci 100 MHz), a dále port RESET, který představuje signál synchronního resetu umožňující vynulování jednotky a její opětovnou inicializaci.

Konfigurační a řídicí rozhraní jednotky ze softwaru je reprezentováno rozhraním sběrnice *AXI4-Lite* [8, 52]. Data určená k šifrování jsou pak do jednotky přenášena po rozhraní *AXI4-Stream* [52]. Stejné rozhraní je pak určeno jako výstupní rozhraní jednotky.

7.1.1 Sběrnice AXI4-Lite

Sběrnice AXI4-Lite slouží v rámci celého systému mikrosondy jako konfigurační a řídicí rozhraní hardwarových jednotek ze softwaru. Nepodporuje „burst“ transakce, všechny datové přenosy obsahují vždy jedno slovo datové šířky rovné šířce příslušného datového kanálu (32 bitů). Rovněž není možné přistupovat přes jednu adresu k registru, který má větší datovou šířku, než je šířka datového kanálu. Sběrnice obsahuje nezávislé kanály pro čtení a zápis, čímž umožňuje zadání dvou souběžných požadavků. Rozhraní sběrnice AXI4-Lite se skládá z pěti kanálů: adresového kanálu pro zápis, datového kanálu pro zápis, kanálu pro odpověď na zápis, adresového kanál pro čtení a datového kanálu pro čtení. Bližší popis těchto kanálů a jejich signálů je možné shlédnout v příloze B.1

V rámci navrhované akcelerační jednotky slouží rozhraní AXI4-Lite pro konfiguraci šifrovacího klíče, inicializačního vektoru šifrovacího algoritmu, přenosu příkazů pro spuštění, zastavení a pozastavování jednotky, a přenosu aktuálního stavu jednotky do řídicího softwaru. Adresový prostor jednotky přes toto rozhraní je dále popsán v sekci 7.4.

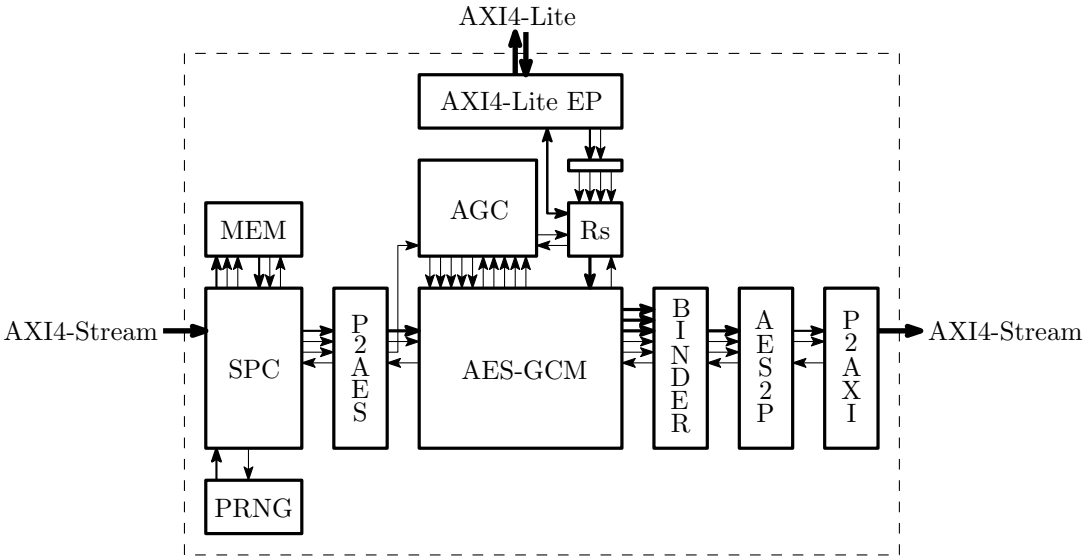
7.1.2 Rozhraní AXI4-Stream

Rozhraní AXI4-Stream představuje datové rozhraní, po kterém se v rámci procesní linky na mikrosondě přenáší zpracovávaná data. Právě data procházející přes toto rozhraní mají být zabezpečeny navrhovanou akcelerační jednotkou. Z tohoto pohledu je pak rozhraní AXI4-Stream jak rozhraním vstupním, tak výstupním. Signály tohoto rozhraní jsou blíže popsány v příloze B.2

Protože v rámci mikrosondy nejsou podporovány transakce, ve kterých by se přenášely po datovém kanálu různě umístěné bajty s využitím platnosti signálu *TKEEP*, přesto může dojít, že transakce nebude zarovnaná na násobek čtyřech bajtů. Proto je kanál *TKEEP* platný pouze při nastaveném signálu *TLAST* a jednotka musí počítat s tím, že se na vstupním rozhraní nezarovnaná transakce vyskytne.

7.2 Architektura akcelerační jednotky

Architektura navržené akcelerační jednotky je přehledně znázorněna na obrázku 7.1. Data do jednotky vstupují po datovém rozhraní AXI4-Stream, popsaném v sekci 7.1.2. První komponentou, která zpracovává data ze vstupního rozhraní, je komponenta *SSH Packet Completer – SPC*, která má za úkol převedení vstupních dat do formátu zprávy transportní vrstvy protokolu SSH. K této komponentě je připojen blok paměti *MEM*, který slouží pro ukládání průběžně přijímaných dat připravované zprávy. Toto je zapotřebí z toho důvodu, že pro správné sestavení cílové zprávy protokolu SSH je potřeba znát velikost přenášených dat. Protože data nemusí být zarovnaná na bloky potřebné velikosti, je k jednotce připojen generátor pseudonáhodných čísel *PRNG*, jehož výstup je použit jako výplň zarovnání zprávy na potřebnou délku.



Obrázek 7.1: Schéma top level architektury akcelerační jednotky

Sestavená zpráva pokračuje do jednotky *PKT To AES Converter – P2AES*, která provádí převod 32 bitového výstupního rozhraní komponenty SPC na 128 bitové vstupní rozhraní komponenty *AES GCM Module – AES-GCM*. Po průchodu jednotkou P2AES je zpráva protokolu SSH připravena k procesu zabezpečení.

Zabezpečení zprávy probíhá v jednotce AES-GCM, blíže popsané v sekci 7.3.3. Tato komponenta provádí zabezpečení zprávy algoritmem AES v režimu GCM, který byl popsán v kapitole 4.1.3. Celý proces zabezpečení v tomto modulu je řízen řídicí jednotkou *AES GCM Controller – AGC*, přičemž konfigurace modulu AES-GCM (šifrovací klíč a inicializační vektor) je uložena v příslušných registrech *Rs*. Spuštění jednotky, její pozastavení, znovuspuštění a konfigurace je prováděno přes konfigurační sběrnici AXI4-Lite. Signály této sběrnice jsou zpracovány koncovou komponentou sběrnice *AXI4-Lite EP* a příslušným adresovým dekodérem.

Zabezpečená zpráva opouští komponentu AES-GCM a vstupuje do komponenty *Packet Binder* (na obrázku označená jako *BINDER*), která zajistí, aby položky zprávy (hlavička, data a integritní součet) byly předány na výstup ve správném pořadí. Data přenášená přes výstupní rozhraní mají stále podobu 128 bitových bloků. Pro jejich převod pro přenos po 32 bitové sběrnici slouží následující komponenta *AES To PKT Converter – AES2P*. Protože výstupním rozhraním je opět AXI4-Stream, následuje v datové cestě ještě jedna komponenta, *PKT To AXI Converter – P2AXI*, která zajistí nastavení příslušných signálů rozhraní AXI4-Stream tak, aby byla zpráva přenášená po tomto rozhraní korektně byla označena příslušnými řídicími signály. Zpráva je vzhledem k povaze zpracování v celé akcelerační jednotce vždy zarovnaná na násobek 32 bitů.

7.3 Komponenty akcelerační jednotky

Obrázek 7.1 znázorňuje modulární architekturu akcelerační jednotky. V rámci této části kapitoly je popsán návrh komponent, ze kterých se akcelerační jednotka skládá.

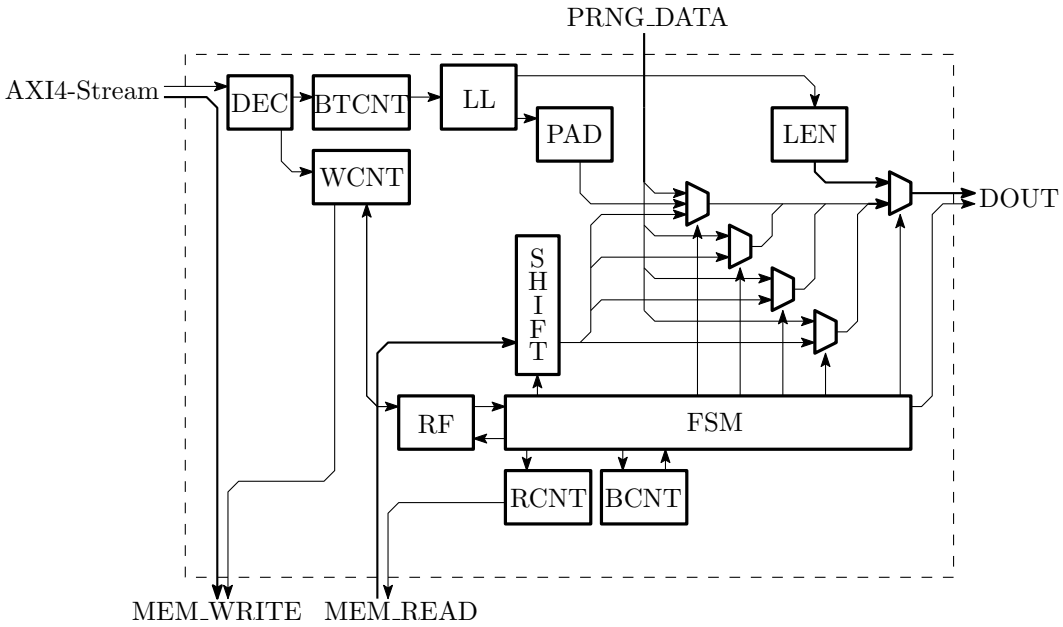
7.3.1 Jednotka SSH Packet Completer

Jednotka *SSH Packet Completer – SPC* slouží pro převedení dat vstupujících do akcelerační jednotky do formátu zprávy transportní vrstvy protokolu SSH. Zjednodušená architektura SPC je znázorněna na obrázku 7.2. V rámci architektury vyznačené na obrázku 7.1 celé akcelerační jednotky odpovídá tato komponenta bloku SPC.

Data vstupují do SPC přes rozhraní AXI4-Stream. Řídicí signály *TKEEP* a *TLAST* tohoto rozhraní jsou dekódovány v bloku označeném *DEC*, který převádí tyto signály na počet platných datových bajtů přenášených po datovém kanálu *TDATA*. O tento počet je inkrementován čítač bajtů vstupních dat *BTCNT*. Tento čítač je důležitý pro odvození délky zprávy a délky jejího případného zarovnání.

Blok *WCNT* reprezentuje čítač zápisové adresy do pomocné paměti, zápis do paměti je realizován přes rozhraní *MEM_WRITE*. Datový vstup paměti je přímo připojen na datový kanál rozhraní AXI4-Stream *TDATA*. Logika povolení zápisu není na obrázku z důvodu přehlednosti znázorněna a je odvozena od aktuální obsazenosti paměti, celkového stavu jednotky (viz dále) a platného signálu *TVALID* vstupního rozhraní AXI4-Stream.

Jakmile dojde k zaplnění paměti (nebo k aktivaci signálu *TLAST* na vstupním rozhraní), dochází k nastavení příznaku čtení v bloku *RF*. Tento příznak způsobí spuštění konečného automatu *FSM*, který řídí proces formátování zprávy protokolu SSH a čtení dat z paměti přes rozhraní *MEM_READ*. Pro účely čtení a řízení je do jednotky zaveden dekrementující čítač zbývajících datových bajtů *BCNT* a čítač čtecí adresy *RCNT*. Logika určení řídicích signálů čtení je podobně jako logika zápisová v obrázku pro přehlednost vynechána a závisí na aktuálním stavu řídicího automatu *FSM* a signálu připravenosti výstupního



Obrázek 7.2: Architektura jednotky SSH Packet Completer

rozhraní. Po dokončení aktuálně tvořené zprávy je příznak čtení *RF* vynulován, čímž je opět povolen zápis dat ze vstupního rozhraní.

Blok *LL* na obrázku 7.2 reprezentuje logiku, která z počtu načtených dat vypočítá nejbližší možnou délku zprávy odpovídající násobku 128 bitů (kvůli zjednodušení procesu budoucího zabezpečení zprávy) a délku příslušného náhodného zarovnání zprávy. Tyto výpočty jsou uložené do registrů *PAD* (délka zarovnání v bajtech) a *LEN* (délka zprávy v bajtech). Náhodná data pro zarovnání přichází po rozhraní *PRNG_DATA*, které je připojeno na generátor pseudonáhodných čísel.

Blok *SHIFT* je posuvná jednotka umožňující vložení délky zarovnání na správné místo v rámci zpracovávané zprávy tím, že se nejnižší bajt přenášeného 32 bitového slova o jeden takt zpozdí, a zbylé bajty se posunou o jednu pozici na jeho místo. Zpožděný bajt je pak přenesen v rámci příštího 32 bitového slova na pozici nejvyššího bajtu. V prvním takto posunutém slově je chybějící bajt doplněn bajtem nesoucím informaci o délce zarovnání, což je hodnota z registru *PAD*.

Výstupní rozhraní *DOUT* jednotky SPC se skládá z následujících portů:

DATA je 32 bitový výstupní datový port, určený pro přenos položek rámce protokolu SSH.

VALID je 1 bitový výstupní port určující platnost dat vystavených na portu *DATA*.

FIRST_FLAG přenáší z jednotky příznak, zda aktuálně přenášené slovo na portu *DATA* je prvním slovem přenášeného rámce.

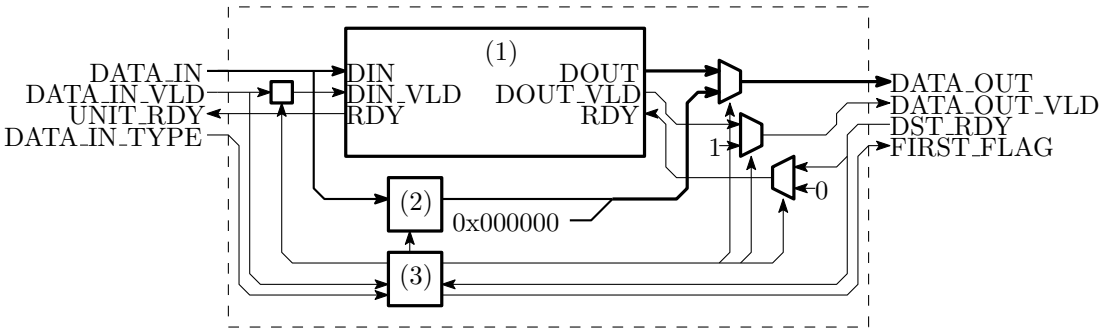
DST_RDY je 1 bitový vstupní port určující připravenost připojené jednotky přijímat data na datovém portu.

Aby nedocházelo ke zdržení na vstupním rozhraní AXI4-Stream při zpracování dat načtených do paměti a při jejich přenosu na výstupní rozhraní, je jednotka uzpůsobena tak, že paměť je rozdělena do dvou regionů, přičemž každý region je určen pro jednu zprávu protokolu SSH. Tím je zajištěno, že zatímco je zpracovávána a odesílána jedna zpráva, data

další zprávy mohou být souběžně načítána do paměti. Díky tomuto překryvu zpracování se podařilo u jednotky dosáhnout teoretické propustnosti 3,1 Gb/s při délce zprávy 512 bajtů[1].

7.3.2 Jednotka PKT To AES Converter

Jednotka *PKT To AES Converter – P2AES* (na obrázku 7.1 označená taktéž jako P2AES) slouží k převodu 32 bitového výstupního rozhraní jednotky SPC na 128 bitové vstupní rozhraní modulu pro šifrování algoritmem AES v režimu GCM (příslušná jednotka bude popsána dále v sekci 7.3.3). Komponenta P2AES je založená na jednotce *Data Width Converter 32 To 128* (1), která provádí převod 32 bitového rozhraní na 128 bitové a je popsána dále. Architektura celého převodníku je znázorněna na obrázku 7.3.



Obrázek 7.3: Architektura jednotky PKT To AES Converter

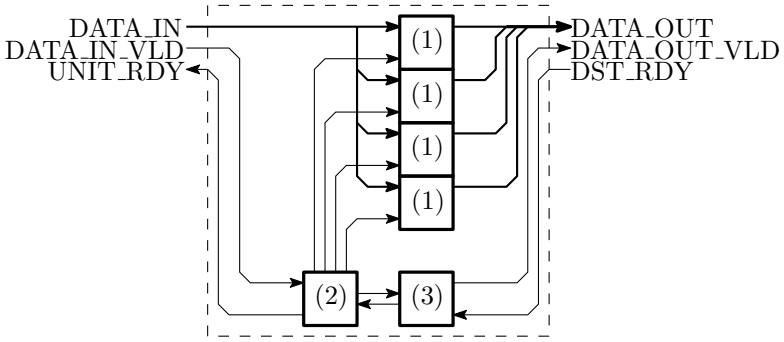
Znázorněný blok (2) je registr sloužící pro uložení 32 bitového vstupního slova, které je označené jako první v rámci vstupující zprávy signálem *DATA_IN_TYPE*. Všechna ostatní datová slova vstupující do jednotky jsou zpracovávána v převodníkové jednotce (1). Blok (3) na obrázku pak reprezentuje logiku potřebnou pro zachování synchronizace mezi vstupním a výstupním rozhraním jednotky a povolení zápisu do registru (2).

Jednotka Data Width Converter 32 To 128

Tato jednotka provádí transformaci vstupního rozhraní o datové šířce 32 bitů na výstupní rozhraní o datové šířce 128 bitů. Zjednodušené schéma jednotky je znázorněno na obrázku 7.4. Jednotka obsahuje čtyři osmibitové registry (1) připojené na vstupní datový port. Zápis do jednotlivých registrů je řízen logikou (2) přihlížející na vstupní signál *DATA_IN_VLD* (signál označující platnost dat na portu *DATA_IN*) a na platný stav uložený ve stavovém registru (3). Hodnota v tomto registru závisí na počtu 32 bitových slov, které byly již do jednotky uloženy a na portu *DST_RDY*, který oznamuje připravenost další jednotky přijímat data. Jestliže je jednotka připravena přijímat data do registrů, je toto oznamováno přes port *UNIT_RDY*.

Protože jednotka pracuje tak, že čtyři takty načítá data do registrů a v jednom taktu je vystaví na 128 bitový výstup, dochází při vyčítání k jednomu taktu pozastavení čtení na vstupním rozhraní. Aby bylo odstraněno toto nežádoucí chování, je navržena obálka

[1]Plné teoretické propustnosti sběrnice AXI4-Stream 3,2 Gb/s jednotka nedosahuje, protože provádí vkládání dvou 32 bitových slov reprezentujících hlavičku protokolu SSH a příslušné zarovnání, čímž je jednotka zpožděna o dva takty. Toto chování lze eliminovat například přidáním dalšího regionu paměti pro třetí zprávu a překryvu jejich zpracování. Protože je ale propustnost jednotky vyšší, než požadavek na propustnost celé akcelerační jednotky protokolu SSH (1 Gb/s), nebyla tato architektura uvažována z důvodu úspory zdrojů.

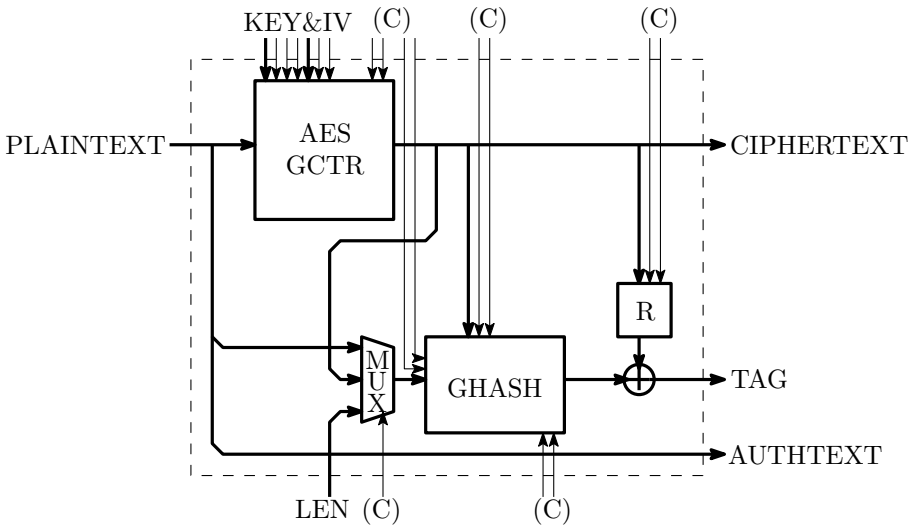


Obrázek 7.4: Architektura jednotky Data Width Converter 32 To 128

Data Width Converter 32 To 128 Wrapper, která obsahuje dva tyto převodníky, zapojené paralelně tak, aby se při načítání dat do registrů průběžně střídaly. Tím je dosaženo, že jednotka nemá na vstupním rozhraní žádné zpoždění a na výstupu jsou data platná každý čtvrtý takt.

7.3.3 Jednotka AES GCM Module

Komponenta *AES GCM Module – AES-GCM* představuje samotné jádro akcelerační jednotky protokolu SSH, protože je komponentou, uvnitř které dochází k zabezpečení dat přenášených z mikrosondy. Jak název napovídá, modul implementuje algoritmus AES v režimu činnosti GCM, který je popsaný v 4.1.3. Architektura modulu je stručně znázorněna na obrázku 7.5, v rámci celkové architektury akcelerační jednotky na obrázku 7.1 odpovídá bloku AES-GCM.



Obrázek 7.5: Architektura jednotky AES GCM Module

Součástí AES-GCM není řídicí logika, která by prováděla synchronizaci jejích jednotlivých částí. Namísto toho má komponenta všechny řídicí signály vyvedené na příslušné vstupně výstupní rozhraní (všechny porty komponenty vyznačené na obrázku 7.5 jako (C)) a pro korektní činnost vyžaduje, aby k tomuto rozhraní byl připojen příslušný řídicí automat, v tomto případě reprezentovaný jednotkou *AES GCM Controller*, která bude popsána

dále v sekci 7.3.4.

Komponenta AES-GCM je především vystavěna na dvou podkomponentách, kterými jsou *AES GCTR*, reprezentující jednotku pro šifrování vstupních dat, a *GHASH Block – GHASH*, provádějící výpočet integritního součtu zabezpečované zprávy. Data v otevřené podobě do komponenty AES-GCM vstupují po 128 bitovém datovém rozhraní *PLAINTEXT*. Na výstupní 128 bitový port *AUTHTEXT* jsou pak tato data přenášena v otevřené podobě v případě, že se jedná o data, která nejsou určena k šifrování, ale pouze k zabezpečení jejich integrity. O platnosti dat na portu *AUTHTEXT*, jako i na portech ostatních, rozhoduje řídicí automat, který není součástí jednotky AES-GCM.

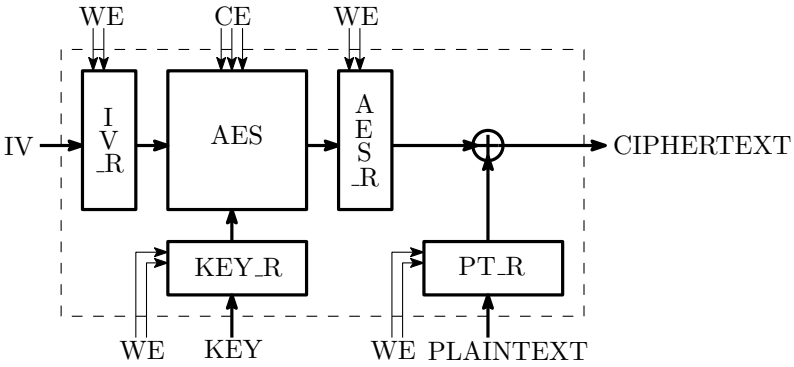
Blok *AES GCTR* provádí šifrování dat algoritmem AES v režimu CTR, popsaném v sekci 4.1.2. Inicializační vektor a šifrovací klíč jsou do modulu nahrány přes rozhraní *KEY&IV*. Šifrovaná data jsou dále přenášena na výstupní port *CIPHERTEXT*. Protože algoritmus AES v režimu CTR neslouží v rámci režimu činnosti GCM jen pro šifrování dat, ale také pro výpočet příslušných hodnot potřebných k výpočtu integritního součtu, jak bylo popsáno algoritmem 3, je výstup z bloku *AES GCTR* připojen také k bloku *GHASH* (pro uložení hashovacího klíče) a k registru *R* pro uložení pomocné hodnoty pro výpočet integritního součtu. Protože režim GCM také provádí výpočet integrity nad zabezpečenými daty, je výstup bloku *AES GCTR* nakonec také připojen ke vstupnímu multiplexoru *MUX* jednotky *GHASH*.

Aby byl výpočet integritního součtu blokem *GHASH* úplný, jsou dále přes rozhraní *LEN* přeneseny příslušné délky autentizovaných a šifrovaných dat zabezpečované zprávy.

Díky tomu, že jsou různé části zprávy platné v různou dobu na různých výstupních portech jednotky AES-GCM, je vyžadováno, aby na výstup AES-GCM byla připojena komponenta, která provede korektní seřazení těchto položek.

Jednotka AES GCTR

Komponenta *AES GCTR* zajišťuje činnost algoritmu AES v režimu CTR, který je popsán v sekci 4.1.2. Architektura komponenty je názorně zobrazena na obrázku 7.6.



Obrázek 7.6: Architektura jednotky AES GCTR

Základem komponenty je *IP Core*[2] [7] implementující zřetězený algoritmus AES, který je schopen v každém taktu vystavit na svůj výstup platný 128 bitový blok, jímž jsou otevřená data šifrována logickou operací xor. Inicializační vektor algoritmu AES-CTR je ukládán přes rozhraní *IV* do registru *IV_R*. Tento registr je čítač, který se s každým taktem inkrementuje

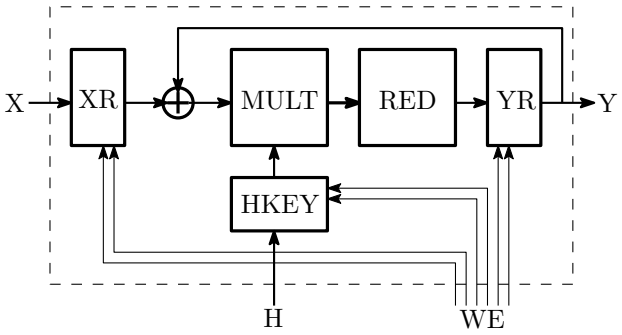
[2] *IP Core (Intellectual Property Core)* je existující implementace podléhající příslušným autorským právům.

o jedničku a jeho hodnota je vstupem zřetězeného bloku AES. Šifrovací klíč je ukládán do registru *KEY_R* přes rozhraní *KEY*. Otevřená data jsou pak ukládána do registru *PT_R* přes rozhraní *PLAINTEXT*. Na hodnoty z registrů *AES_R* a *PT_R* je aplikována operace xor, čímž je provedeno šifrování otevřených dat. Šifrovaná data jsou následně přenášena z jednotky přes 128 bitový datový port *CIPHERTEXT*.

Rozhraní *WE* umožňuje povolovat zápis do jednotlivých registrů, pozastavování zřetězeného bloku je prováděno přes rozhraní *CE*.

Jednotka GHASH Block

Architektura jednotky *GHASH Block* je znázorněna na obrázku 7.7 a je implementací výpočtu funkce GHASH popsané algoritmem 2. Data X zabezpečovaná touto funkcí jsou průběžně ukládána v registru *XR*. Výsledek výpočtu je pak ukládán do registru *YR* a se zpožděním jednoho taktu je vystaven na rozhraní *Y*. V registru *HKEY* je uložen platný hashovací klíč, jež je do jednotky nahrán přes rozhraní *H*. Blok označený jako *MULT* reprezentuje operaci násobení bez přenosu do vyššího řádu (tzv. *carryless* násobení) a blok označený jako *RED* reprezentuje operaci redukce v $GF(2^{128})$. Tyto dva bloky jsou dohromady implementací operace násobení $\bullet$, která byla popsána algoritmem 1. Řízení zápisů do registrů a jejich nulování je provedeno přes rozhraní *WE*.



Obrázek 7.7: Architektura jednotky GHASH Block

Jak již bylo zmíněno, operace násobení $\bullet$ lze optimalizovat pro dosažení vyšších rychlostí jak v softwaru, tak v hardwaru. Společnost Intel představila tyto optimalizace v [23] a z nich také vychází návrh a implementace příslušných výpočetních bloků MULT a RED pro komponentu *GHASH Block*.

Komponentou, která zajišťuje carryless násobení dvou 128 bitových bloků, je *Carryless Multiplier 128b*. Jak si ukážeme, lze komponentu vytvořit čistě jako kombinační logiku, čímž lze odstranit nepříjemné cykly v algoritmu 1. Nejprve je však zapotřebí představit carryless násobení 64 bitových bloků, které umožňuje významnou část logiky ušetřit a také zkrátit možné kritické cesty.

Mějme 64 bitové bloky $A = a_{63}a_{62}\ldots a_0, B = b_{63}b_{62}\ldots b_0$ a 128 bitový blok $C = c_{127}c_{126}\ldots c_0$, který je výsledkem operace carryless násobení bloků A a B. Operace carryless násobení je pak vyjádřena vztahy:

$$c_i = \oplus_{j=0}^{i} a_j b_{i-j}, \qquad \text{pro } i \in 0,\ldots,63 \tag{7.1}$$

$$c_i = \oplus_{j=i-63}^{64} a_j b_{i-j}, \qquad \text{pro } i \in 64,\ldots,127 \tag{7.2}$$

Nyní, když je představena operace carryless násobení 64 bitových bloků, lze přistoupit k vyjádření carryless násobení 128 bitových bloků. Výpočet je založen na použití něko-

lika 64 bitových carryless násobiček, použití logické operace xor na některé jejich výstupy a konkatenace vzniklých bloků.

Mějme bloky $A = A_1 \parallel A_0, B = B_1 \parallel B_0$ ($\parallel$ nechť je operace konkatenace, bloky A_1, A_0, B_1, B_0 jsou 64 bitové). Výpočet carryless násobení $A \cdot B$ pak lze vyjádřit následujícími vzorci [23]:

$$C = C_1 \parallel C_0 = A_1 \cdot B_1 \tag{7.3}$$
$$D = D_1 \parallel D_0 = A_0 \cdot B_0 \tag{7.4}$$
$$E = E_1 \parallel E_0 = (A_1 \oplus A_0) \cdot (B_1 \oplus B_0) \tag{7.5}$$
$$A \cdot B = C_1 \parallel (C_1 \oplus C_0 \oplus D_1 \oplus E_1) \parallel (D_1 \oplus D_O \oplus C_0 \oplus E_0) \parallel D_0 \tag{7.6}$$

Výsledek tohoto násobení je 256 bitový blok. Nyní zbývá představit operaci redukce v $GF(2^{128})$, která, aplikovaná na tento výsledek, uzavře kruh výpočtu operace násobení $\bullet$, představené algoritmem 1. Je důležité podotknout, že následující optimalizace představuje redukci výhradně pro generující polynom vyjádřený konstantou R v algoritmu 1 a ne pro žádný jiný. Mějme tedy 256 bitový vstupní blok operace redukce $X = X_3 \parallel X_2 \parallel X_1 \parallel X_0$, rozdělený na čtyři 64 bitové bloky X_3, X_2, X_1, X_0. Výsledkem bude 128 bitový blok I, vypočtený následovně:

$$X = X \ll 1 \tag{7.7}$$
$$A = X_0 \ll 63 \tag{7.8}$$
$$B = X_0 \ll 62 \tag{7.9}$$
$$C = X_0 \ll 57 \tag{7.10}$$
$$D = X_1 \oplus A \oplus B \oplus C \tag{7.11}$$
$$E = E_1 \parallel E_0 = (D \parallel X_0) \gg 1 \tag{7.12}$$
$$F = F_1 \parallel F_0 = (D \parallel X_0) \gg 2 \tag{7.13}$$
$$G = G_1 \parallel G_0 = (D \parallel X_0) \gg 7 \tag{7.14}$$
$$H = H_1 \parallel H_0 = (D \oplus E_1 \oplus F_1 \oplus G_1) \parallel (X_0 \oplus E_0 \oplus F_0 \oplus G_0) \tag{7.15}$$
$$I = (X_3 \oplus H_1) \parallel (X_2 \oplus H_0) \tag{7.16}$$

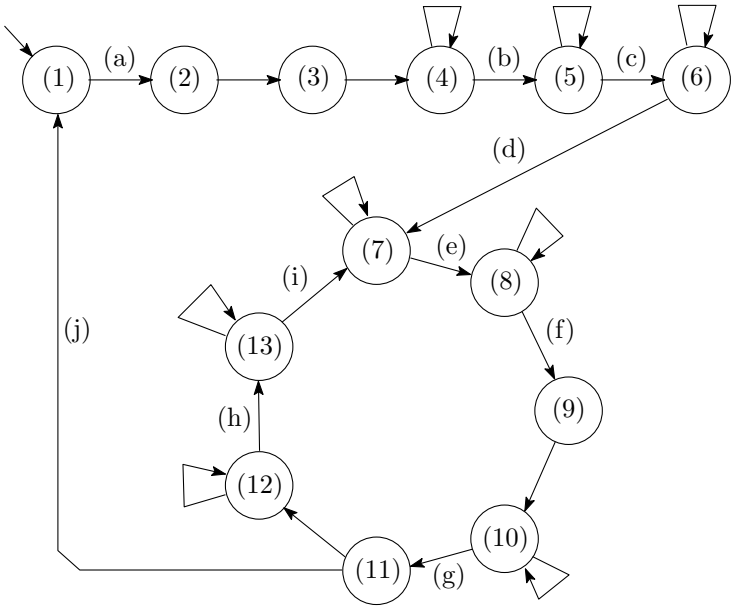
Těmito postupy je ukázáno, že celý výpočet algoritmu 1 lze převést na kombinační logiku. Právě díky tomu dosahují architektury Intel s instrukční sadou [22] tak vysokých propustností při šifrování režimem GCM. Shodným způsobem bude provedena implementace funkce GHASH i v rámci této práce.

7.3.4 Jednotka AES GCM Controller

Jednotka *AES GCM Controller* (na obrázku 7.1 blok s označením AGC) se skládá ze dvou základních částí. První částí je konečný automat provádějící základní řízení modulu *AES GCM Module*, druhá část je pak podpůrná logika pro výpočet signálů označující platnost datových výstupů v závislosti na stavu a připravenosti přímo připojených jednotek.

Graf přechodů konečného automatu je znázorněn na obrázku 7.8. Činnost automatu v závislosti na stavech je následující (čísla stavů na obrázku odpovídají příslušnému pořadí v následujícím výčtu):

(1) V počátečním stavu se automat nachází po prvním spuštění jednotky, případně po události (j) dále popsané u stavu (12). Jednotka opouští počáteční stav do stavu (2) při výskytu události (a), která reprezentuje přijetí příkazu `START`.



Obrázek 7.8: Řídicí automat jednotky AES GCM Controller

(2) V rámci tohoto stavu dochází k nahrání šifrovacího klíče do příslušného registru GCM modulu. Protože nedochází ke generování žádných dat, na jejichž potvrzení by se čekalo, automat plynule přechází do stavu (3).

(3) V tomto stavu provádí automat nastavení zápisu inicializačního vektoru. Současně je v tomto stavu spuštěno šifrování nulového bloku (pro připomenutí se jedná o šestý krok algoritmu 3). Automat plynule přechází do stavu (4).

(4) Tento stav je stavem, během kterého dochází k průchodu průběžně generovaných hodnot čítače zřetězeným algoritmem AES do doby, než je na výstupu zřetězení první platný výsledek tohoto algoritmu. Pro použitou implementaci zřetězeného AES je tato latence 29 taktů. Po tuto dobu setrvává automat v tomto stavu a poté přechází do stavu (5) (dosažení 29. taktu čekání je reprezentováno událostí (b)).

(5) První vypočtená hodnota zřetězeného algoritmu AES je platná na výstupu příslušné jednotky a odpovídá hashovacímu klíči. Tento stav tedy nastavuje zápis do příslušného registru jednotky *GHASH Block*. Současně jsou v rámci tohoto stavu vystavena na výstup první autentizovaná data a zároveň je povolen jejich zápis do jednotky *GHASH Block*. Automat setrvává v tomto stavu tak dlouho, dokud se tento zápis neprovede (např. z důvodu čekání na platná data), což je reprezentováno událostí (c). Stav, do kterého automat přejde po události (c), je stav (6).

(6) V tomto stavu je výstupem zřetězeného algoritmu AES pomocná hodnota, která bude sloužit k výpočtu integritního součtu. V rámci tohoto stavu je povolen zápis této hodnoty do příslušného registru. Rovněž je povolen zápis otevřeného textu do příslušného registru, jehož obsah bude od následujícího stavu šifrován. Automat setrvává v tomto stavu do té doby, než je zápis otevřeného textu úspěšný, tj. na vstupu modulu GCM jsou vystavena platná data. Po této události, označené (d), přechází automat do stavu (7).

(7) Stav (7) je stavem, který reprezentuje šifrování vstupujících dat a výpočet integrity zašifrovaných dat. Tomu odpovídá příslušné povolení zápisu do příslušných registrů a povolení činnosti zřetězeného AES. V případě nepřipravenosti jednotky připojené k výstupu modulu GCM, je činnost modulu pozastavena, a čeká se na potvrzení aktuálně šifrovaných dat. Jakmile je zašifrován celý datagram vstupující do modulu GCM, což je reprezentováno událostí (e), přechází automat do následujícího stavu (8).

(8) V tomto stavu dochází k přípravě GCM modulu pro výpočet integritního součtu. Současně se musí čekat na potvrzení přijetí posledního šifrovaného bloku dat (událost (f)). Poté automat přechází do stavu (9).

(9) Ve stavu (9) je vypočten integritní součet a automat plynule přechází do stavu (10).

(10) Integritní součet vystavený na výstup modulu GCM je platný a čeká se na potvrzení jeho přijetí jednotkou připojenou k výstupu modulu GCM (událost (g)). Poté automat přechází do stavu (11).

(11) Tento stav reprezentuje konec zabezpečení jedné zprávy. V rámci tohoto stavu je provedeno vynulování příslušných registrů tak, aby byla jednotka připravena pro vstup další zprávy. V případě, že byl obdržen příkaz `STOP` (událost (j)) pro zastavení a restart činnosti akcelerační jednotky, přechází automat z toho stavu do stavu počátečního (stav (1)), kde očekává příkaz `START` a celá činnost se opakuje. V případě, že byl obdržen příkaz `SUSPEND`, jednotka setrvává v tomto stavu do obdržení příkazu `RESUME` a poté pokračuje do stavu (12). Pokud nebyl obdržen příkaz `STOP` ani `SUSPEND`, automat přechází do stavu (12) a začíná zpracovávat další zprávu.

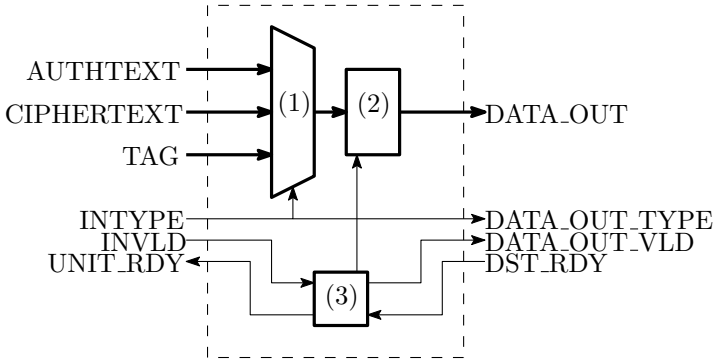
(12) Tento stav je obdobou stavu (5) a provádí zabezpečení autentizovaných dat. Oproti stavu (5) však nedochází k uložení hashovacího klíče, který je platně nahrán ve stavu (5). Po potvrzení přijetí autentizovaných dat výstupní jednotkou automat přechází do stavu (13).

(13) Tento stav je obdobou stavu (6), dochází k povolení zápisu prvních dat určených k šifrování do příslušných registrů, nedochází však k zápisu hodnoty pro výpočet integritního součtu, která zůstává shodná tak, jak byla uložena stavem (6). Jakmile jsou data na vstupu platná (tzn. zápis je proveden), automat přechází do stavu (7).

7.3.5 Jednotka Packet Binder

Architektura jednotky *Packet Binder* slouží k zajištění, že jednotlivé položky přenášené zprávy (hlavička, data, integritní součet) jsou z jednotky AES-GCM přenesené ve správném pořadí. Svou podstatou není jednotka ničím jiným, než multiplexorem (1), který přepíná jednotlivé položky zprávy v závislosti na jejich typu (*INTYPE*) v pořadí, v jakém vycházejí z modulu AES-GCM. V rámci schématu celé akcelerační jednotky vyobrazeném na obrázku 7.1 představuje tato komponenta blok BINDER.

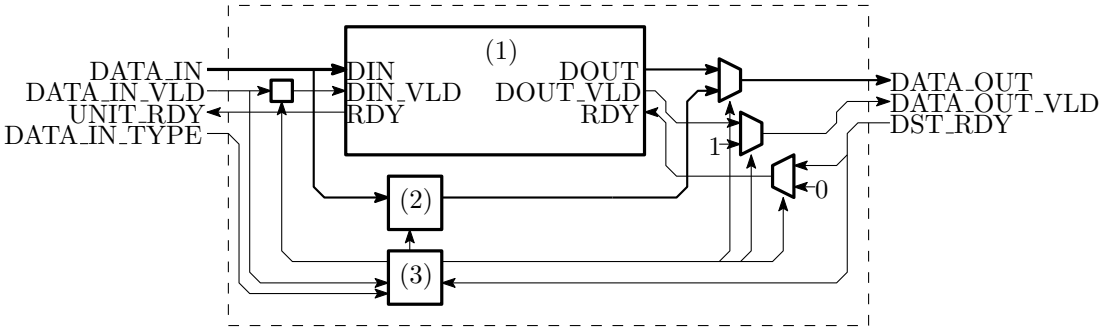
Komponenta musí zajistit, aby v případě nepřipravenosti další jednotky přijímat výstupní data, byla tato informace předána jednotce AES-GCM a rovněž aby se aktuálně přenášená informace neztratila. K tomu slouží datový registr *(2)*. K uchování informace o platnosti dat v registru (2), povolení zápisu do něj a výpočtu platného příznaku připravenosti slouží logický blok (3).



Obrázek 7.9: Architektura jednotky Packet Binder

7.3.6 Jednotka AES To PKT Converter

Jednotka *AES To PKT Converter* (AES2P v kontextu obrázku 7.1) slouží k převodu 128 bitového výstupního rozhraní jednotky *Packet Binder* na 32 bitové rozhraní, které lze připojit k rozhraní AXI4-Stream. Komponenta je založená na jednotce *Data Width Converter 128 To 32* (1), která provádí převod 128 bitového rozhraní na 32 bitové a je popsána dále. Architektura celého převodníku je znázorněna na obrázku 7.10.



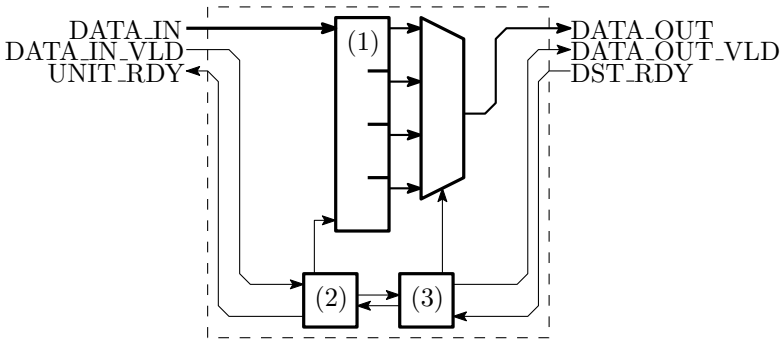
Obrázek 7.10: Architektura jednotky AES To PKT Converter

Znázorněný blok (2) je registr sloužící pro uložení horních 32 bitů 128 bitového vstupního slova, které je označené jako první v rámci vstupující zprávy hodnotou 1 na portu *DATA_IN_TYPE*. Horních 32 bitů je vybráno z toho důvodu, že dolních 96 bitů je nulové zarovnání. První 128 bitové slovo totiž nese 32 bitovou položku hlavičky zprávy protokolu SSH vyjadřující velikost zprávy. Všechna ostatní datová slova vstupující do jednotky AES2P, jsou zpracovávána v převodníkové jednotce (1). Blok (3) na obrázku pak reprezentuje logiku potřebnou pro zachování synchronizace mezi vstupním a výstupním rozhraním jednotky a povolení zápisu do registru (2).

Jednotka Data Width Converter 128 to 32

Tato jednotka provádí transformaci vstupního rozhraní o datové šířce 128 bitů na výstupní rozhraní o datové šířce 32 bitů. Zjednodušené schéma jednotky je znázorněno na obrázku 7.11. Jednotka obsahuje jeden 128 bitový registr (1) připojený na vstupní datový port. Zápis do registru je řízen logikou (2) přihlížející na vstupní signál *DATA_IN_VLD* (signál označující platnost dat na portu *DATA_IN*) a na platný stav uložený ve stavovém registru

(3). Registr (3) uchovává informaci o tom, zda jednotka provádí čtení jednotlivý 32 bitových slov uložených v registru (1), nebo zda je povolen zápis do tohoto registru.



Obrázek 7.11: Architektura jednotky Data Width Converter 128 To 32

Protože jednotka pracuje tak, že jeden takt načítá data do registru a následující čtyři takty vybírá jednotlivá 32 bitová slova z tohoto registru, dochází při čtení ke čtyřem taktům pozastavení čtení na vstupním rozhraní. Aby byla alespoň částečně kompenzována prodleva při zápisu, je navržena obálka *Data Width Converter 128 To 32 Wrapper*, která obsahuje dva tyto převodníky zapojené paralelně tak, aby se při načítání dat do registrů průběžně střídaly. Tím je dosaženo, že jednotka vystavuje na výstup každý takt platné 32 bitové slovo a provádí načítání každý čtvrtý takt.

7.3.7 Jednotka PKT To AXI Converter

Komponenta *PKT To AXI Converter* (v kontextu obrázku 7.1 označená jako P2AXI) přidává ke vstupnímu rozhraní řídicí signály rozhraní AXI4-Stream, konkrétně *TKEEP* a *TLAST*. Protože data, která do komponenty vstupují, i data, která z ní vystupují, jsou zarovnaná na násobek celých 32 bitů, signál *TKEEP* je fixně nastaven na hodnotu `0xF`. Signál *TLAST* indikuje poslední 32 bitové slovo přenášené zprávy. Je aktivován vždy po přenesení množství dat odpovídajícího velikosti přenášené zprávy protokolu SSH.

7.3.8 Ostatní jednotky

V rámci architektury navržené akcelerační jednotky byly také instancovány komponenty, jejichž návrhem a implementací se tato práce nezabývá. Mezi tyto jednotky patří *Half Dual Port BlockRAM Memory* (na obrázku 7.1 MEM), sloužící jako paměť pro uložení zpracovávané zprávy v jednotce SPC, dále se jedná o IP Core implementující algoritmus AES, generátor pseudonáhodných čísel *mLSFR* a koncový uzel sběrnice AXI4-Lite EP, pro který je využita komponenta *AXI4-Lite To MI32 Converter*.

Paměť Half Dual Port BlockRAM Memory

Tento blok paměti poskytuje klasické čtecí a zápisové rozhraní, přes které umožňuje současně provádět operace zápis a čtení. Kapacita je plně nastavitelná, v rámci této práce je nastavena na $2 \cdot 508$ bajtů. Implementace pochází z svn repozitáře projektu Sec6Net.

AES IP Core

Protože je algoritmus AES běžně používaným šifrovacím algoritmem a existuje k němu mnoho softwarových i hardwarových implementací, bylo rozhodnuto, že není zapotřebí provádět opětovnou implementaci tohoto algoritmu. Proto byl zvolen existující IP Core [7], který je implementací algoritmu AES s vlastnostmi vhodnými pro použití v rámci akcelerační jednotky. Mezi tyto vlastnosti patří především zřetězená architektura, která umožňuje snadné zapojení tohoto algoritmu v režimu činnosti CTR a tím i GCM.

Inicializace dané jednotky probíhá 30 taktů, po níž je na její výstup vystaven první platný 128 bitový blok použitelný pro šifrování. Za přepokladu, že je provedena výměna inicializačního vektoru v průběhu relace, je zapotřebí jednotku znovu 30 taktů inicializovat.

Protože rozhraní akcelerační jednotky umožňuje její průběžné pozastavování dle stavu procesní linky, je nutné zajistit, aby také tento IP Core měl možnost být pozastaven. Původně získaná implementace tímto nedisponuje a proto bylo zapotřebí zasáhnout do implementace vnitřních registrů IP Core a přidat signál Clock Enable.

IP Core disponuje po úpravě následujícím rozhraním (hodinový signál a RESET signál není ve výpisu zahrnut):

KEY je 128 bitový vstupní port určený pro nahrání platného klíče, kterým mají být šifrována data na vstupním portu *PLAINTEXT*.

PLAINTEXT je 128 bitový vstupní port určený pro nahrání otevřených dat, která mají být šifrována s využitím klíče *KEY*.

CE je Clock Enable signál umožňující pozastavení činnosti jednotky.

CIPHERTEXT je 128 bitový výstupní port, na kterém jsou vystavena šifrovaná data.

Tvůrce tohoto IP Core otestoval jednotku testovacími vektory zveřejněnými organizací NIST [43], což by měl být doklad správné funkčnosti této jednotky.

Generátor pseudonáhodných čísel

Jako generátor pseudonáhodných čísel, potřebný pro generování náhodného zarovnání v rámci zpráv protokolu SSH v jednotce SPC, byla zvolena existující implementace [37] několikanásobného paralelního LSFR. Generátory pseudonáhodných čísel se tato diplomová práce nezabývá, přesto by bylo do budoucna vhodné, aby byly ověřeny vlastnosti tohoto generátoru pro použití v kombinaci se šifrováním, a v případně nutnosti zvolit jinou implementaci vhodnějšího, kryptografického generátoru náhodných čísel.

Komponenta AXI4-Lite To MI32 Converter

Protože je akcelerační jednotka připojena na sběrnici AXI4-Lite, bylo zapotřebí zvolit vhodný koncový bod sběrnice, který provádí převod rozhraní sběrnice na zápisové a čtecí signály příslušných registrů jednotky. Pro tyto účely byla zvolena existující komponenta *AXI4-Lite To MI32 Converter*, která převádí rozhraní sběrnice AXI4-Lite na následující rozhraní:

ADDR je 32 bitový port přenášející adresu registru, do kterého bude prováděn zápis nebo ze kterého bude probíhat čtení.

ARDY představuje signál potvrzující převzetí požadavku na čtení nebo zápis.

WR je vstupní signál indikující požadavek na zápis, současně určuje platnost adresy vystavené na adresovém kanálu *ADDR* a datovém kanálu *DWR*.

DWR je 32 bitový datový kanál určený pro zápis dat do komponenty.

BE je signál určující platnost jednotlivých bajtů 32 bitového slova přenášeného na datovém kanálu *DWR*.

RD signál indikující požadavek na čtení.

DRD je 32 bitový datový kanál určený pro přenos čtené informace z komponenty.

DRDY je signál označující platnost dat vystavených na datovém kanálu *DRD*.

Toto rozhraní neumožňuje narozdíl od AXI4-Lite současný zápis a čtení, což ovšem nepředstavuje problém, protože zvolená komponenta obsahuje vyrovnávací FIFO pro případ, že by k souběžnému zápisu a čtení došlo.

Příslušné registry akcelerační jednotky jsou připojené k výstupnímu rozhraní komponenty *AXI4-Lite To MI32 Converter* dle adresového schématu popsaného v sekci 7.4. Pro správné nastavení zápisových signálů do registrů dle platné adresy je implementován příslušný adresový dekodér.

7.4 Adresový prostor a ovládání jednotky

Protože jednotka obsahuje několik konfiguračních registrů (registr příkazů, registr pro šifrovací klíč, registr pro inicializační vektor, stavový registr), musí být jednotka přístupná z adresového prostoru konfigurační sběrnice AXI4-Lite. Adresový prostor jednotky je uveden v tabulce 7.1 a vychází z počtu adresovatelných bajtů registrů jednotky přístupných ze sběrnice AXI4-Lite.

Adresa	Určení	Typ
0x00000000	Registr příkazů, 4 B	Zápis
0x00000004	Šifrovací klíč, bajty 3 - 0	Zápis
0x00000008	Šifrovací klíč, bajty 7 - 4	Zápis
0x0000000C	Šifrovací klíč, bajty 11 - 8	Zápis
0x00000010	Šifrovací klíč, bajty 15 - 12	Zápis
0x00000014	Inicializační vektor, bajty 3 - 0	Zápis
0x00000018	Inicializační vektor, bajty 7 - 4	Zápis
0x0000001C	Inicializační vektor, bajty 11 - 8	Zápis
0x00000040	Stavový register, 4 B	Čtení

Tabulka 7.1: Adresový prostor akcelerační jednotky

Protože je sběrnice AXI4-Lite 32 bitová a neumožňuje „burst“ přenosy, je nutné u registrů větších 32 bitů provést adresaci jejich jednotlivých částí. Tak tomu je u registrů ukládajících šifrovací klíč a inicializační vektor. Při konfiguraci jednotky je toto potřeba mít na paměti, protože pro očekávané chování akcelerační jednotky je zapotřebí nakonfigurovat celý šifrovací klíč a inicializační vektor.

7.4.1 Příkazy

Akcelerační jednotku je možné ovládat přes sběrnici AXI4-Lite pomocí příkazů uvedených v tabulce 7.2. Jednotlivé příkazy jsou nahrány do registru příkazů akcelerační jednotky a ovlivňují především činnost řídicího konečného automatu jednotky *AES GCM Controller*.

Hodnota	Příkaz
0x00000001	START
0x00000002	STOP
0x00000004	SUSPEND
0x00000008	RESUME

Tabulka 7.2: Seznam příkazů akcelerační jednotky

Příkaz `START` uvede akcelerační jednotku do stavu činnosti. Je potřeba si uvědomit, že po obdržení tohoto příkazu provede jednotka svou inicializaci a k tomu, aby inicializace proběhla korektně, je zapotřebí, aby byly do jednotky nahrány inicializační vektor a šifrovací klíč ještě před příkazem `START`.

Na základě přijetí příkazu `STOP` ukončuje jednotka svou činnost. Než jednotka svou činnost na základě obdržení tohoto příkazu ukončí, dokončí zabezpečení a odeslání právě zpracovávané zprávy, a to z toho důvodu, aby nedošlo ke ztrátě dat. Po přijetí příkazu `STOP` se jednotka dostává do počátečního stavu a vyčkává na příkaz `START`. Příkaz je zde zaveden především kvůli nutnosti provést novou inicializaci jednotky při výměně šifrovacího klíče a inicializačního vektoru.

Příkaz `SUSPEND` umožňuje přerušit činnost jednotky, ale narozdíl od příkazu `STOP` tak činí bez ztráty stavové informace. Po obdržení příkazu `RESUME` jednotka pokračuje zpracováním další zprávy při zachování nastavené konfigurace. Pozastavení jednotky může být užitečné, pokud je zapotřebí vložit mezi odesílané zprávy například zprávu vytvořenou v softwaru.

7.4.2 Stavy jednotky

Okamžitý stav jednotky je uložen ve stavovém registru, který je adresovatelný přes sběrnici AXI4-Lite na adrese `0x00000040`. Registr může nabývat hodnot a odpovídajících stavů popsaných v tabulce 7.3.

Hodnota	Příkaz
0x00000000	RUNNING
0x00000002	STOPPED
0x00000003	INIT
0x00000004	SUSPENDED

Tabulka 7.3: Možné stavy akcelerační jednotky

Na počátku se nachází jednotka ve stavu `INIT`, kterým indikuje, že je zastavená a že dochází k inicializaci pseudonáhodného generátoru *mLSFR*. Po skončení jeho inicializace jednotka přechází do stavu `STOPPED`. Kdykoli v průběhu stavu `INIT` a `STOPPED` jednotka očekává konfiguraci inicializačního vektoru a šifrovacího klíče. Ve stavu `STOPPED` pak očekává přijetí příkazu `START`, aby mohla zahájit svou činnost. Jakmile tento příkaz obdrží, přechází jednotka do stavu `RUNNING`, během kterého ji lze zastavit příkazem `STOP`, případně pozastavit příkazem `SUSPEND`. Pokud byla jednotka zastavena, přechází do stavu `STOPPED`

a očekává příkaz `START`. V případě obdržení příkazu `SUSPEND` přechází do stavu `SUSPENDED`, ve kterém očekává příkaz `RESUME`, který ji opětovně uvede do stavu `RUNNING`.

Inicializační vektor a šifrovací klíč by měl být měněn výhradně ve stavu jednotky `STOPPED`, aby nedošlo k narušení šifrování aktuálně zpracovávané zprávy.

Kapitola 8

Dosažené výsledky

Tato kapitola prezentuje dosažené výsledky práce. Návrh celé jednotky již byl popsán v předcházející kapitole, průběh testování jednotky je popsán v sekci 8.1. V sekci 8.2 jsou prezentovány výsledky dosažené při syntéze, které naznačují, že nasazení jednotky do cílového systému by nemělo být problematické. Propustnost dosažená v simulačním prostředí je prezentována v sekci 8.3 a možná pokračování práce jsou diskutována v sekci 8.4.

8.1 Testování

Testování akcelerační jednotky a jejích komponent probíhalo v simulačním nástroji *ISim* [54], který je součástí vývojového prostředí Xilinx ISE Design Suite. Pro každý logický celek (logickým celkem může být například *SSH Packet Completer* s připojenou pamětí a náhodným generátorem) byl připraven testbench pro ověření, že tento celek správně reaguje na své vstupy a dává validní výsledky na svém výstupu.

IP Core reprezentující implementaci algoritmu AES byl testován na referenčních testovacích vektorech od organizace NIST [45]. Snaha byla podobný postup aplikovat také na implementaci komponenty *AES GCM Module*, protože však referenční testovací vektory [15] obsahují data, která nejsou zarovnaná a nejsou ve formátu zprávy, která do komponenty skutečně vstupuje, byly cílové testy pouze inspirovány referenčními vektory.

Pro důkladnější ověření funkčnosti jednotky by vzhledem k její komplexnosti bylo vhodné vytvořit funkční verifikaci. Jak by tato verifikace mohla vypadat, je diskutováno v rámci sekce 8.4.2. Implementace takové verifikace přesahuje rozsah této práce.

Vytvořená jednotka nebyla testována na cílové sondě, protože její nasazení závisí na dalších částech systému, které dosud nejsou vytvořené. Těmito částmi je softwarová část, zajišťující vytváření zabezpečované relace a konfiguraci jednotky, a hardwarový TCP/IP modul, který má zajistit přenos dat z akcelerační jednotky protokoly TCP a IP a dále přes počítačovou síť.

8.2 Syntéza

V rámci použitého vývojového prostředí Xilinx ISE Design Suide verze 14.1 jsou i nástroje pro překlad a syntézu. V tabulce 8.1 je možné shlédnout spotřebované zdroje na cílovém čipu a srovnat je s dostupnými zdroji na čipu (dostupné zdroje vychází z největšího známého designu pro čip FPGA mikrosondy v rámci projektu Sec6Net). Podrobný souhrn spotřebovaných zdrojů na čipu lze shlédnout v příloze C.

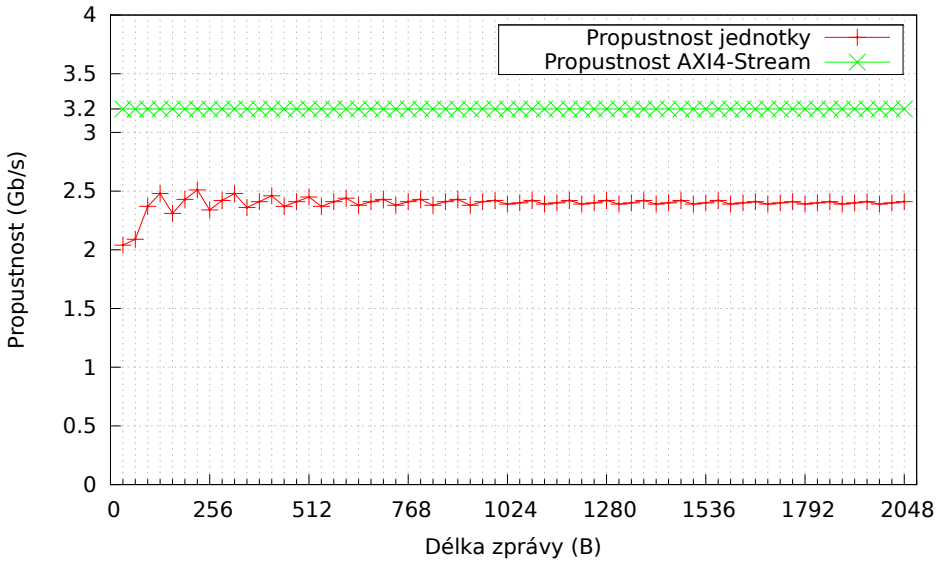
Zdroj	Počet využitých	Počet dostupných
Slice Registers	11 710 (6 %)	51 %
Slice LUTs	18 884 (20 %)	81 %
BlockRAMs	3 (1 %)	60 %

Tabulka 8.1: Využité zdroje na cílovém čipu FPGA

Z tabulky je zřetelné, že na čipu stále zůstává dostatek prostoru na další jednotky, například na v kapitole 6 zmíněný TCP/IP modul. Akcelerační jednotku se podařilo syntetizovat na frekvenci 121 MHz, která tím plní požadavek systému minimální frekvence 100 MHz. Úzkým hrdlem systému, které brání dosažení vyšších frekvencí, jsou kombinační cesty v komponentě *GHASH Block*, která provádí výpočet funkce *GHASH*. Pokud by tato kritická cesta představovala problém při instancování akcelerační jednotky do reálného designu, je možné navrhnout případné úpravy, které jsou diskuzované dále v 8.4.3.

8.3 Propustnost v simulaci

V rámci simulace implementované akcelerační jednotky bylo provedeno také měření její propustnosti v závislosti na různých délkách zpráv. Měření bylo provedeno od zpráv délky 32 do 2048 bajtů, délka zprávy byla vždy násobkem 32 bajtů. Výsledek je vynesen do grafu na obrázku 8.1. Pro porovnání je ve stejném grafu vynesena také maximální teoretická propustnost sběrnice AXI4-Stream, která je vstupním a výstupním datovým rozhraním akcelerační jednotky.



Obrázek 8.1: Graf závislosti propustnosti jednotky na délce datagramu

Z obrázku je patrné, že jednotka dosahuje maximální propustnosti pohybující se kolem 2,4 Gb/s. Fakt, že jednotka nedostahuje maximální propustnosti sběrnice, je zapříčiněn drobným zpožděním v rámci jednotky *SSH Packet Completer* tím, že provádí připojení hlavičky k datům a příslušné zarovnání na bloky, které jsou násobkem 16 bajtů. Další zpoždění je způsobeno připojením 128 bitového integritního součtu vypočteného komponentou *AES*

GCM Module k vytvořené zprávě. Toto dává prostor pro případné optimalizace, pokud by bylo zapotřebí dosáhnout propustnosti vyšší. Protože ale požadavek na propustnost jednotky byl na počátku stanoven na 1 Gb/s, nepředstavuje dosažená propustnost žádný problém a optimalizace za účelem jejího zvýšení není nutná.

8.4 Možnosti pokračování práce

Ačkoli je akcelerační jednotka navržená a implementovaná, stále existuje celá řada prací, které je potřeba provést pro reálné nasazení jednotky na mikrosondu. Hlavní je pochopitelně tvorba softwarové části bezpečnostního systému. Jakým směrem by se implementace softwarové části měla ubírat, je naznačeno v sekci 8.4.1. Dále by bylo zapotřebí provést implementaci rozsáhlejších testů akcelerační jednotky pro její plné ověření před nasazením do hardwaru. Protože je jednotka poměrně komplikovaná, mohla by pro tyto účely být vytvořena funkční verifikace, jejíž schéma je navržené v sekci 8.4.2. Další práce na akcelerační jednotce může spočívat v případných optimalizacích dle potřeb cílového systému, které mohou v průběhu času ještě vzniknout. Tyto optimalizace mohou spočívat ve snaze o úsporu zdrojů, případně o přerušení některých kritických cest, které by bránily práci designu na vyšších frekvencích. Tyto možnosti jsou diskutovány v sekci 8.4.3.

8.4.1 Softwarová část akcelerovaného systému

Klientské aplikace, vytvořené s využitím knihoven implementujících prostředky protokolu SSH, zpravidla dodržují následující případ užití:

1. Vytvoření systémového soketu TCP pro navázání spojení a komunikace se serverem,
2. výměna úvodních zpráv a autentizačních informací a ustanovení relace SSH přes vytvořený soket,
3. zabezpečení dat určených k přenosu dle dohodnutých parametrů vytvořené relace,
4. přenos zabezpečených dat přes vytvořený soket a příjem zabezpečených dat ze strany serveru,
5. pokud jsou přijata data, jejich dešifrování a interpretace,
6. pokud dojde k vypršení příslušných časovačů nebo čítačů nebo přijetí požadavku na obnovu bezpečnostních parametrů relace provedení dohody nových parametrů mezi komunikujícími stranami.

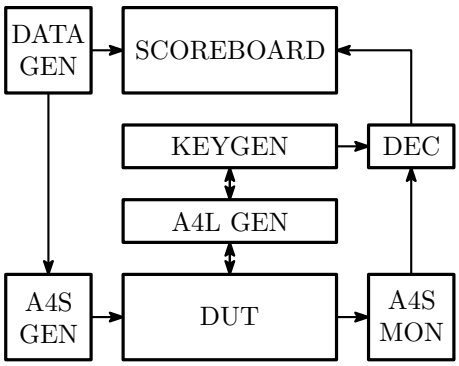
Úprava softwarové části, která je v této sekci představena, je založena na předpokladu existence hardwarového modulu TCP/IP a příslušné vrstvy operačního systému, která je schopna s tímto modulem komunikovat. Pak prvotní úprava softwarové knihovny spočívá ve výměně volání operačního systému pro vytvoření soketu TCP a provedení případné konfigurace modulu TCP/IP.

Dále je zapotřebí implementovat prostředky pro zajištění konfigurace akcelerační jednotky navržené a implementované v rámci této práce. Toto je možné učinit vytvořením příslušné funkce nad existujícím ovladačem pro konfiguraci registrů na čipu FPGA přes sběrnici AXI4-Stream. Konfigurace je prováděna po dohodnutí parametrů vytvořené relace. V případě, že je provedena výměna těchto parametrů v průběhu relace, musí tato funkce respektovat činnost a ovládání akcelerační jednotky popsané v sekci 7.4.

Výměna zabezpečených dat probíhá pouze ve směru z mikrosondy, konkrétně z procesní linky, a jejich zabezpečení neprobíhá v softwaru, nýbrž v akcelerační jednotce. Všechna příchozí data v zabezpečené podobě musí být naproti tomu přenesena do softwaru. Předpokládá se, že tato data jsou požadavky na výměnu parametrů relace nebo informace podobného charakteru. Dešifrování a ověření integrity těchto přijatých dat je provedeno v softwaru. V případě výměny bezpečnostních parametrů (např. šifrovacího klíče) je zapotřebí pozastavit akcelerační jednotku v její činnosti a provést její rekonfiguraci.

8.4.2 Funkční verifikace

Funkční verifikací pro hardwarově akcelerované systémy se zabývá například práce [60]. Zjednodušené schéma možné verifikace pro navrženou akcelerační jednotku je znázorněné na obrázku 8.2.



Obrázek 8.2: Schéma zapojení funkční verifikace

Blok *DATA GEN* na obrázku 8.2 představuje generátor dat přenášených protokolem SSH, která budou zabezpečena v akcelerační jednotce. Tato data je potřeba převést na formát příslušný sběrnici AXI4-Stream, k čemuž slouží blok označený jako *A4S GEN*. Generovaná data jsou rovněž předána bloku *SCOREBOARD*, který provádí porovnání a vyhodnocení vstupů a odpovídajících výstupů testované akcelerační jednotky, označované jako *Design Under Test – DUT*.

Blok označený jako *KEYGEN* představuje generátor šifrovacích klíčů a inicializačních vektorů. Tento blok musí umět zohlednit konfiguraci testované jednotky *DUT* a také její řízení z pohledu příkazů a stavů představených v sekci 7.4. Výstupy jednotky jsou předávány bloku *A4L GEN*, který provádí konverzi příkazů na rozhraní sběrnice AXI4-Lite. V případě šifrovacích klíčů a inicializačních vektorů pak také dochází k jejich uložení do bloku *DEC*, který bude popsán dále.

Výstup testované jednotky *DUT* je připojen na monitorovací jednotku rozhraní AXI4-Stream *A4S MON*, která provádí transformaci dat přenášených po AXI4-Stream na příslušný formát pro další blok *DEC*. Blok *DEC* představuje dešifrovací jednotku, která provede dešifrování přenášené zprávy a ověří také příslušný integritní součet. Data v otevřené podobě a výsledek ověření integritního součtu předává bloku *SCOREBOARD*.

Poznámky k funkční verifikaci

Při tvorbě příslušné simulační verifikace je potřeba zohlednit, že data generovaná generátorem *DATA GEN* jsou považována v rámci přenosu protokolem SSH za jeden stream

a jsou rozdělena na zprávy protokolu SSH různé velikosti. Výstupem testované jednotky jsou tato data zapouzdřená příslušnou hlavičkou určující velikost dat a velikost zarovnání. Proto se velikosti bloků dat vstupující do jednotky *SCOREBOARD* mohou lišit svou velikostí ze vstupů *DATA GEN* a *DEC*. O ověření správnosti hlaviček a zabezpečení by se měla postarat dešifrovací jednotka *DEC* a blok *SCOREBOARD* by pak měl provést výhradně srovnání přenesených dat.

8.4.3 Možné optimalizace

Teoretická propustnost sběrnice AXI4-Stream při frekvenci hodinového signálu 100 MHz dosahuje maximálně 3,2 Gb/s. Pokud ale uvážíme, že jednotka *AES GCM Module*, realizující zabezpečení dat algoritmem AES v režimu činnosti GCM po 128 bitových blocích, na stejné frekvenci to dává maximální teoretickou propustnost 12,8 Gb/s. Toto může být výhoda v tom, že tato jednotka je použitelná i na systémech s požadavkem na vyšší propustnost. Naopak na mikrosondě to může znamenat, že jednotka dosahuje vysoké propustnosti například na úkor spotřebovaných zdrojů. Výsledky překladu diskutované v předchozí kapitole 8.2 sice jsou v současné době velice pozitivní a s počtem dostupných zdrojů na mikrosondě problém není, v budoucnu by ale mohl problém nastat, pokud budou kladené další požadavky na rozšíření funkčnosti mikrosondy. Vyšší teoretická propustnost, než je požadováno, tedy nabízí možnosti optimalizace spotřeby zdrojů.

Vysoká propustnost je způsobena použitím zřetězeného algoritmu AES, který každý takt generuje 128 bitový blok pro šifrování, a souběžného výpočtu integritního součtu zabezpečovaných dat funkcí GHASH. Pro dosažení maximální propustnosti datové sběrnice přitom postačuje, aby blok AES generoval na jeden takt pouze 32 bitové bloky, resp. aby generoval 128 bitový blok jednou za čtyři takty. Optimalizace může spočívat ve výměně plně zřetězeného bloku AES, za takový, který bude odpovídat popsanému chování. Díky této úpravě by mohlo dojít ke snížení zdrojů spotřebovaných na zřetězený AES až na čtvrtinu. Porovnání tohoto odhadu se současným stavem je vynesen do tabulky 8.2 (uvedená procenta představují spotřebu z celkového množství zdrojů na čipu).

Zdroj	Využitých celkem	Spotřeba AES	Odhadovaná úspora
Slice Registers	11 710 (6 %)	8 002 (4 %)	až 6 000
Slice LUTs	18 884 (20 %)	10 782 (11 %)	až 8 000
BlockRAMs	3 (1 %)	2 (1 %)	1–2

Tabulka 8.2: Odhadované úspory zdrojů úpravou zřetězeného AES

Další možná optimalizace se může zaměřit například na tu část akcelerační jednotky, která ji „zpomaluje“ nejdelší kombinační cestou. Touto jednotkou je jednotka *GHASH Block* pro výpočet funkce integritního součtu přenášených dat popsaný algoritmem 2. Maximální dosažitelná frekvence této jednotky při syntéze je 121 MHz, což může při velkém zaplnění čipu být podstatně méně. Pokud nahlédneme na architekturu této jednotky na obrázku 7.7, optimalizace kritické cesty může spočívat ve vložení registrů mezi bloky MULT a RED. Doba výpočtu jednotky se pak prodlouží o jeden takt, což vzhledem k vysoké propustnosti jednotky nemusí představovat problém.

Kapitola 9

Závěr

Cílem této diplomové práce bylo vytvoření hardwarové akcelerační jednotky pro zabezpečení přenosu dat z vestavěného zařízení. Tímto zařízením je mikrosonda, která je součástí systému pro realizaci zákonných odposlechů vyvíjeného v rámci projektu Sec6Net. Data přenášená ze systému jsou považována za citlivá a je potřeba je zabezpečit proti neoprávněnému prozrazení a modifikaci. Tato práce seznamuje čtenáře s mechanismy pro zabezpečení přenosu dat počítačovou sítí a provádí jejich srovnání s cílem vybrat nejvhodnější mechanismus pro použití na mikrosondě. Pro tyto účely byl zvolen protokol SSH.

Součástí cílového vestavěného zařízení je čip FPGA, který obsahuje pouze pomalé procesory MicroBlaze. Z tohoto důvodu není možné provádět zabezpečení velkého objemu dat softwarově a je potřeba zvolený mechanismus hardwarově akcelerovat, aby zpracování dosahovalo požadované propustnosti 1 Gb/s. Kompletní implementace protokolu SSH v hardwaru však není reálná z hlediska dostupných zdrojů na čipu FPGA. Proto se tato práce zabývá také srovnáním existujících softwarových implementací protokolu SSH. Cílem srovnání je vybrat možný základ pro softwarovou část systému a zjistit kompatibilitu dostupných řešení s navrhovanou akcelerační jednotkou.

Hlavní přínos této práce spočívá především v návrhu a implementaci hardwarové akcelerační jednotky. Konečná varianta jednotky byla vybrána ze třech navržených a diskutovaných variant a je založena na šifrovacím algoritmu AES v režimu činnosti GCM. Navržená jednotka je rovněž uvedena do kontextu celého systému mikrosondy, včetně návrhu jejího zapojení k modulům mikrosondy, které dosud nebyly implementovány.

Akcelerační jednotka byla implementována v jazyce VHDL. Ve vývojovém prostředí Xilinx ISE Design Suite 14.1 [53] se ji podařilo syntetizovat na frekvenci 100 MHz s využitím přiměřeného množství zdrojů. Maximální dosažená propustnost jednotky v simulaci pak dosahovala 2,4 Gb/s, což s výraznou rezervou plní požadavek na propustnost systému 1 Gb/s.

Čtenáři jsou rovněž prezentována možná pokračování této práce. Nejdůležitějším pokračováním je především implementace softwarové části akcelerovaného systému. Pro důkladné ověření funkčnosti akcelerační jednotky je vhodné, aby byla implementována funkční verifikace. Dále jsou představené možnosti optimalizace jednotky, a to jak z pohledu spotřeby zdrojů, tak z pohledu dosažení vyšší frekvence.

Výsledky srovnání bezpečnostních mechanismů pro účely mikrosondy byly publikovány v technické zprávě projektu Sec6Net [30] a návrh akcelerační jednotky byl rovněž publikován a prezentován na soutěži EEICT [36].

Literatura

[1] *WWW stránky knihovny FlowSsh* [online]. [cit. 3. ledna 2013]. Dostupné na: <`http://www.bitvise.com/flowssh`>.

[2] *WWW stránky knihovny libssh* [online]. [cit. 3. ledna 2013]. Dostupné na: <`http://www.libssh.org`>.

[3] *WWW stránky knihovny libssh2* [online]. [cit. 3. ledna 2013]. Dostupné na: <`http://www.libssh2.org`>.

[4] *WWW stránky knihovny NetSieben SSH* [online]. [cit. 3. ledna 2013]. Dostupné na: <`http://www.netsieben.com/products/ssh`>.

[5] *WWW stránky knihovny paramiko* [online]. [cit. 3. ledna 2013]. Dostupné na: <`http://www.lag.net/paramiko/`>.

[6] *WWW stránky projektu OpenSSH* [online]. [cit. 3. ledna 2013]. Dostupné na: <`http://www.openssh.org`>.

[7] *WWW stránky projektu OpenCores: Pipelined AES* [online]. 2010 [cit. 10. května 2013]. Dostupné na: <`http://www.opencores.org/project,aes_pipe`>.

[8] ARM Ltd. *AMBA AXI Protocol Specification*. 2010. Verze 2.0.

[9] Baker, F., Foster, B. a Sharp, C. *RFC 3924 Cisco Architecture for Lawful Intercept in IP Networks*. Říjen 2004.

[10] Bellare, M., Kohno, T. a Namprempre, C. *RFC 4344 The Secure Shell (SSH) Transport Layer Encryption Modes*. Leden 2006.

[11] Bider, D. a Baushke, M. *RFC 6668 SHA-2 Data Integrity Verification for the Secure Shell (SSH) Transport Layer Protocol*. Červenec 2012.

[12] Chilkat Software. *WWW stránky knihovny Chilkat SSH* [online]. [cit. 3. ledna 2013]. Dostupné na: <`http://www.chilkatsoft.com/ssh-features.asp`>.

[13] Dierks, T. *RFC 5246 The Transport Layer Security (TLS) Protocol Version 1.2*. Srpen 2008.

[14] Dworkin, M. *Recommendation for Block Cipher Modes of Operation: Methods and Techniques*. Prosinec 2001. NIST Special Publication 800-38A.

[15] Dworkin, M. *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*. Listopad 2007. NIST Special Publication 800-38D.

[16] European Telecommunications Standards Institute. *ETSI TR 101 943: Telecommunications Security; Lawful Interception (LI); Concepts of Interception in a Generic Network Architecture.* Francie: European Telecommunications Standards Institute, Červenec 2001. Verze 1.1.1.

[17] European Telecommunications Standards Institute. *ETSI TR 102 528: Lawful Interception (LI); Interception Domain Architecture for IP Networks.* Francie: European Telecommunications Standards Institute, Říjen 2006. Verze 1.1.1.

[18] European Telecommunications Standards Institute. *ETSI TR 102 661: Lawful Interception (LI); Security Framework in Lawful Interception and Retained Data Environment.* Francie: European Telecommunications Standards Institute, Listopad 2009. Verze 1.2.1.

[19] Frankel, S., Glenn, R. a Kelly, S. *RFC 3602 The AES-CBC Cipher Algoritm and Its Use with IPsec.* Září 2003.

[20] Freier, A., Karlton, P. a Kocher, P. *RFC 6101 The Secure Sockets Layer (SSL) Protocol Version 3.0.* Srpen 2011.

[21] Galbraith, J. a Saarenmaa, O. *SSH File Transfer Protocol.* Červenec 2006. Internet-Draft.

[22] Gopal, V., Ozturk, E., Feghali, W. et al. *Optimized Galois-Counter-Mode Implementation on Intel Architecture Processors* [online]. Srpen 2010 [cit. 14. května 2013]. Dostupné na: <`http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/communications-ia-galois-counter-mode-paper.pdf`>.

[23] Gueron, S. a Kounavis, M. E. *Intel Carry-Less Multiplication Instruction and its Usage for Computing the GCM Mode* [online]. Květen 2010 [cit. 14. května 2013]. Dostupné na: <`http://software.intel.com/sites/default/files/m/4/1/2/2/c/1230-Carry-Less-Multiplication-and-The-GCM-Mode_WP_.pdf`>.

[24] Haller, N. *RFC 1995 The S/Key One-Time Password System.* Únor 1995.

[25] Hoffman, P. *RFC 4308 Cryptographic Suites for IPsec.* Prosinec 2005.

[26] Housley, R. *RFC 3686 Using Advanced Encryption Standard (AES) Counter Mode With IPsec Encapsulating Security Payload (ESP).* Leden 2004.

[27] Housley, R. *RFC 4309 Using Advanced Encryption Standard (AES) CCM Mode with IPsec Encapsulating Security Payload (ESP).* Prosinec 2005.

[28] Igoe, K. *RFC 6239 Suite B Cryptographic Suites for Secure Shell (SSH).* Květen 2011.

[29] Igoe, K. a Solinas, J. *RFC 5647 AES Galois Counter Mode for the Secure Shell Transport Layer Protocol.* Srpen 2009.

[30] Kajan, M., Koranda, K. a Polčák, L. *Spolehlivá a zabezpečená komunikace v rámci systému pro zákonné odposlechy.* Brno, Česká republika: Fakulta informačních technologií, Vysoké učení technické v Brně, 2012. 26 s. Technická zpráva, FIT-TR-2012-007.

[31] Kaufman, C. *RFC 4306 Internet Key Exchange (IKEv2) Protocol*. Prosinec 2005.

[32] Kaufman, C., Perlman, R. a Speciner, M. *Network Security: Private Communication in a Public World*. Druhé vydání. New Jersey: Prentice Hall PTR, 2002. ISBN 0-13-046019-2.

[33] Kent, S. *RFC 4302 IP Authentication Header*. Prosinec 2005.

[34] Kent, S. *RFC 4303 IP Encapsulating Security Payload (ESP)*. Prosinec 2005.

[35] Kent, S. a Seo, K. *RFC 4301 Security Architecture for the Internet Protocol*. Prosinec 2005.

[36] Koranda, K. Hardware Accelerated System for Securing Network Traffic. In *Proceedings of the 19th Conference STUDENT EEICT 2013 Volume 2*. Brno: Fakulta elektrotechniky a komunikačních technologií a Fakulta informačních technologií, Vysoké učení technické v Brně, duben 2013. S. 285–287. ISBN 978-80-214-4694-6.

[37] Korček, P. *Generování pseudonáhodných čísel v FPGA*. Brno: Fakulta informačních technologií, Vysoké učení technické v Brně, 2007. Bakalářská práce.

[38] Kořenek, J., Korček, P. a Kaštil, J. *Sondy pro monitorování provozu*. Brno, Česká republika: Fakulta informačních technologií, Vysoké učení technické v Brně, 2011. Technická zpráva, FIT-TR-2011-09.

[39] Kořenek, J., Korček, P., Košař, V. et al. A New Embedded Platform for Rapid Development of Networking Applications. In *Proceedings of the 2012 Seventh ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS 2012)*. Austin, US: IEEE Computer Society, 2012. S. 81–82. ISBN 978-1-4503-1684-2.

[40] Lehtinen, S. a Lonvick, C. *RFC 4250 The Secure Shell (SSH) Protocol Assigned Numbers*. Leden 2006.

[41] Liu, Y., Wu, L., Niu, Y. et al. A High-Speed SHA-1 IP Core for 10 Gbps Ethernet Security Processor. In *2012 Eighth International Conference on Computational Intelligence and Security (CIS)*. Prosinec 2012. S. 237–241. ISBN 978-1-4673-4725-9.

[42] Maughan, D., Schertler, M., Schneider, M. et al. *RFC 2408 Internet Security Association and Key Management Protocol (ISAKMP)*. Listopad 1998.

[43] McGrew, D. a Viega, J. *The Galois/Counter Mode of Operation (GCM)* [online]. Květen 2005 [cit. 10. května 2013]. Dostupné na: <`http://www.csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/gcm/gcm-revised-spec.pdf`>.

[44] Mocana Corporation. *WWW stránky produktu NanoSSH* [online]. [cit. 3. ledna 2013]. Dostupné na: <`http://www.mocana.com/nanossh.html`>.

[45] National Institute of Standards and Technology. *Annouincinf the Advanced Encryption Standard (AES)*. Listopad 2001. Federal Information Processing Standards Publication 197.

[46] Orman, H. *RFC 2412 The OAKLEY Key Determination Protocol*. Listopad 1998.

[47] Polčák, L., Kramoliš, P., Kajan, M. et al. *Architektura systému pro zákonné odposlechy.* Brno, Česká republika: Fakulta informačních technologií, Vysoké učení technické v Brně, 2011. 25 s. Technická zpráva, FIT-TR-2011-008.

[48] Rescorla, E. *RFC 2631 Diffie-Hellman Key Agreement Method.* Červen 1999.

[49] Rescorla, E. a Modadugu, N. *RFC 6347 Datagram Transport Layer Security Version 1.2.* Leden 2012.

[50] Salman, A., Rogawski, M. a Kaps, J.-P. Efficient Hardware Accelerator for IPSec based on Partial Reconfiguration on Xilinx FPGAs. In *2011 International Conference on Reconfigurable Computing and FPGAs.* Mexico: IEEE Computer Society, prosinec 2011. S. 242–248.

[51] Turner, S. a Polk, T. *RFC 6176 Prohibiting Secure Sockets Layer (SSL) Version 2.0.* Březen 2011.

[52] Xilinx. *AXI Reference Guide* [online]. Listopad 2012 [cit. 14. května 2013]. Verze 14.3. Dostupné na: <`http://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/lates/ug761_axi_reference_guide.pdf`>.

[53] Xilinx. *ISE Design Suite* [online]. 2013 [cit. 10. května 2013]. Dostupné na: <`http://www.xilinx.com/products/design-tools/ise-design-suite/index.htm`>.

[54] Xilinx. *ISE Simulator (ISim)* [online]. 2013 [cit. 11. května 2013]. Dostupné na: <`http://www.xilinx.com/tools/isim.htm`>.

[55] Ylonen, T. a Lonvick, C. *RFC 4251 The Secure Shell (SSH) Protocol Architecture.* Leden 2006.

[56] Ylonen, T. a Lonvick, C. *RFC 4252 The Secure Shell (SSH) Authentication Protocol.* Leden 2006.

[57] Ylonen, T. a Lonvick, C. *RFC 4253 The Secure Shell (SSH) Transport Layer Protocol.* Leden 2006.

[58] Ylonen, T. a Lonvick, C. *RFC 4254 The Secure Shell (SSH) Connection Protocol.* Leden 2006.

[59] Zhu, L., Jaganathan, K. a Hartman, S. *RFC 4121 The Kerberos Version 5 Generic Security Service Application Program Interface (GSS-API) Mechanism: Version 2.* Červenec 2005.

[60] Šimková, M. *Hardware Accelerated Functional Verification.* Brno: Fakulta informačních technologií, Vysoké učení technické v Brně, 2011. Diplomová práce.

Seznam příloh

Příloha A

Srovnání dostupných implementací protokolu SSH

Příloha uvádí tabulkové srovnání vybraných implementací protokolu SSH. Tím doplňuje text kapitoly 5.

Knihovna	Licence	Linux	Klient	Server
libssh	GNU LGPL	Ano	SSHv1, SSHv2	SSHv1, SSHv2
libssh2	BSD	Ano	SSHv2	Ne
FlowSsh	Komerční	Ne	SSHv2	SSHv2
NetSieben SSH	QPL i komerční	Ano	SSHv2	Ne
Chilkat SSH	Komerční	Ano	SSHv2	Ne
paramiko	GNU LGPL	Ano	SSHv2	SSHv2

Tabulka A.1: Implementace SSH - srovnání vlastností I

Knihovna	SFTP	IPv6	Komprese
libssh	Ano	Ano	Ano
libssh2	Ano	Ano	Ano
FlowSsh	Ano	Ne	Ano
NetSieben SSH	Ano	Ne	Ne
Chilkat SSH	Ano	Ano	?
paramiko	Ano	Ano	Ano

Tabulka A.2: Implementace SSH - srovnání vlastností II (? v tabulce symbolizuje, že daná informace není dostupná.)

Knihovna	Jazyk	Dokumentace	Příklady	Aktuální
libssh	C	Ano, generovaná	Ano	Ano, 20.11.2012
libssh2	C/C++	Ano	Ano	Ano, 27.11.2012
FlowSsh	C/C++, .NET	Ano	Ano	Ano
NetSieben SSH	C/C++	Ano, generovaná	Ano	Ne, 19.11.2009
Chilkat SSH	C/C++, C# a jiné	Nedostupná	Ano	Ano, 13.12.2012
paramiko	Python	Ano, generovaná	Ano	Ano, 21.5.2011

Tabulka A.3: Implementace SSH - srovnání vlastností III

Knihovna	Heslo	Veřejné klíče	Další
libssh	Ano	Ano, RSA, DSS	—
libssh2	Ano	Ano, RSA, DSS	*hostbased*, *keyboard interactive*
FlowSsh	Ne	Ano, RSA, DSS	Kerberos
NetSieben SSH	Ano	Ano, RSA, DSS	Ne
Chilkat SSH	Ano	Ano, RSA, DSS	*keyboard interactive*
paramiko	Ano	Ano, RSA, DSS	*keyboard interactive*

Tabulka A.4: Implementace SSH - dostupné autentizační mechanismy

Knihovna	RC4	Blowfish	3DES	AES	Další
libssh	Ne	Ano	Ano	Ano, 128, 192, 256	—
libssh2	Ano	Ano	Ano	Ano, 128, 192, 256	Cast
FlowSsh	Ano	Ano	Ano	Ano, 128, 192, 256	Crypto++ knihovna
NetSieben SSH	Ne	Ano	Ano	Ano, 128, 256	Twofish, Cast
Chilkat SSH	?	?	?	?	?
paramiko	Ano	Ano	Ano	Ano	—

Tabulka A.5: Implementace SSH - dostupné šifrovací algoritmy (? v tabulce symbolizuje, že daná informace není dostupná.)

Knihovna	MD5	SHA-1	SHA-256	AES
libssh	Ano	Ano	Ne	Ne
libssh2	Ano	Ano	Ne	Ne
FlowSsh	Ano	Ano	Ne	Ne
NetSieben SSH	Ano	Ano	Ne	Ne
Chilkat SSH	?	?	?	?
paramiko	Ano	Ano	Ne	Ne

Tabulka A.6: Implementace SSH - dostupné prostředky pro zajištění integrity dat (? v tabulce symbolizuje, že daná informace není dostupná.)

Příloha B

Popis signálů sběrnic AXI4-Lite a AXI4-Stream

V rámci této přílohy jsou popsány signály sběrnic AXI4-Lite a AXI4-Stream [52, 8], které tvoří rozhraní navržené akcelerační jednotky. Tato příloha doplňuje sekci 7.1.

B.1 Signály sběrnice AXI4-Lite

Rozhraní sběrnice AXI4-Lite se skládá z pěti kanálů: adresového kanálu pro zápis, datového kanálu pro zápis, kanálu pro odpověď na zápis, adresového kanál pro čtení a datového kanálu pro čtení.

B.1.1 Adresový kanál pro zápis

Adresový kanál pro zápis slouží pro vystavení požadavku na zápis do příslušného adresovaného registru a skládá se z následujících signálů:

AWADDR představuje kanál o šířce 32 bitů, přenášející adresu určenou pro zápis.

AWPROT je signál umožňující nastavení typu ochrany přenášených dat; u sběrnice AXI4-Lite je fixně nastavený na hodnotu 000_2 a jedná se ignorovaný signál [52].

AWVALID je signál reprezentující požadavek na zápis a platnost adresy vystavené na kanálu *AWADDR*, a je nastaven, dokud není požadavek potvrzen signálem *AWREADY*.

AWREADY je signálem, jímž jednotka dává najevo schopnost přijmout požadavek na zápis.

B.1.2 Datový kanál pro zápis

Datový kanál pro zápis slouží pro přenos hodnoty zapisované do registru adresovaného přes adresový kanál pro zápis:

WDATA je datový kanál o šířce 32 bitů sloužící pro přenos 32 bitové hodnoty určené pro zápis.

WSTRB je čtyřbitový kanál, v rámci kterého jednotlivé bity signalizují platnost jednotlivých bajtů v 32 bitovém slově přenášeném po kanálu *WDATA*, a tím určuje, které

bajty adresované 32 bitového registru mají být aktualizovány na zapisovanou hodnotu.

WVALID je signál určující platnost dat vystavených na kanálu *WDATA* a rovněž platnost *WSTRB*.

WREADY reprezentuje připravenost jednotky zapisovat data a potvrzuje přijetí platných dat na datovém kanálu *WDATA*.

B.1.3 Kanál pro odpověď na zápis

Kanál pro odpověď na zápis je určen pro indikaci případného chybového stavu, ke kterému mohlo dojít při pokusu o zápis do konfigurované či řízené jednotky. V rámci AXI4-Lite je pro tyto účely definované toto rozhraní:

BRESP je kanálem reprezentujícím odpověď konfigurované jednotky ve formě stavu `OKAY` (zápis se podařil), `SLVERR` (jednotka vrací chybu zápisu) nebo `DECERR` (adresovaná jednotka neexistuje).

BVALID určuje platnost kódu vystaveného na kanálu *BRESP*.

BREADY signalizuje připravenost sběrnice přijmout informaci přenášou po kanále *BRESP*.

B.1.4 Adresový kanál pro čtení

Adresový kanál pro čtení je určený pro vystavení požadavku na čtení hodnoty z adresovaného 32 bitového registru jednotky. Skládá se z následujících signálů:

ARADDR představuje kanál o šířce 32 bitů, přenášející adresu registru, ze kterého je požadováno čtení.

ARPROT je tříbitový kanál určující typ ochrany čtených dat, v případě AXI4-Lite fixně nastavený na hodnotu 000_2, jedná se o ignorovaný signál [52].

ARVALID je signál reprezentující požadavek na čtení a určující platnost adresy vystavené na kanálu *ARADDR*.

ARREADY je signál, jímž jednotka dává najevo schopnost přijmout požadavek na čtení.

B.1.5 Datový kanál pro čtení

Datový kanál pro čtení slouží pro přenesení čtených dat z jednotky. Rovněž je schopen oznamovat, zda došlo k chybě při čtení.

RDATA je datový kanál o šířce 32 bitů, po kterém se přenáší vyčtená data.

RRESP představuje odpověď na požadavek čtení (obdobný signál je *BRESP* v případě zápisu) signalizovaný formou kódů `OKAY`, `SLVERR` a `DECERR`, jejichž význam se neliší od stejně jmenovaných kódů na kanálu *BRESP*.

RVALID je signál určující platnost dat přenášených po kanálu *RDATA*.

RREADY indikuje připravenost sběrnice přijímat vyčítaná data z jednotky.

B.2 Signály sběrnice AXI4-Stream

Rozhraní sběrnice AXI4-Stream se skládá z následujících signálů:

TDATA představuje datový kanál o šířce 32 bitů.

TVALID je signálem určujícím platnost dat přenášených po datovém kanálu a stejně tak platnost ostatních řídicích signálů rozhraní.

TKEEP je čtyřbitový kanál určující platnost jednotlivých bajtů přenášeného slova po datovém kanálu *TDATA*.

TLAST signalizuje poslední přenášené slovo přenášeného rámce.

TREADY je signál potvrzující schopnost příjemce přijmout slovo přenášené po datovém kanálu.

Příloha C

Výsledky překladu – využití zdrojů

Příloha uvádí detailní rozpis spotřeby zdrojů na čipu FPGA při syntéze akcelerační jednotky na frekvenci 100 MHz. Příloha doplňuje sekci 8.2.

```
Slice Logic Utilization:
  Number of Slice Registers:                11,710 out of 184,304 6%
    Number used as Flip Flops:              11,710
    Number used as Latches:                      0
    Number used as Latch-thrus:                  0
    Number used as AND/OR logics:                0
  Number of Slice LUTs:                     18,884 out of  92,152 20%
    Number used as logic:                   18,249 out of  92,152 19%
      Number using O6 output only:          16,580
      Number using O5 output only:              12
      Number using O5 and O6:                1,657
      Number used as ROM:                        0
    Number used as Memory:                      64 out of  21,680 1%
      Number used as Dual Port RAM:              0
      Number used as Single Port RAM:            0
      Number used as Shift Register:            64
        Number using O6 output only:            64
        Number using O5 output only:             0
        Number using O5 and O6:                  0
    Number used exclusively as route-thrus:    571
      Number with same-slice register load:    567
      Number with same-slice carry load:         4
      Number with other load:                    0

Slice Logic Distribution:
  Number of occupied Slices:                 6,445 out of  23,038 27%
  Nummber of MUXCYs used:                      144 out of  46,076 1%
  Number of LUT Flip Flop pairs used:       20,409
    Number with an unused Flip Flop:        10,599 out of  20,409 51%
    Number with an unused LUT:               1,525 out of  20,409 7%
    Number of fully used LUT-FF pairs:       8,285 out of  20,409 40%
    Number of slice register sites lost
```

```
      to control set restrictions:                 0 out of 184,304 0%

Specific Feature Utilization:
  Number of RAMB16BWERs:                           0 out of     268 0%
  Number of RAMB8BWERs:                            3 out of     536 1%
  Number of BUFIO2/BUFIO2_2CLKs:                   0 out of      32 0%
  Number of BUFIO2FB/BUFIO2FB_2CLKs:               0 out of      32 0%
  Number of BUFG/BUFGMUXs:                         1 out of      16 6%
    Number used as BUFGs:                          1
    Number used as BUFGMUX:                        0
  Number of DCM/DCM_CLKGENs:                       0 out of      12 0%
  Number of ILOGIC2/ISERDES2s:                    32 out of     586 5%
    Number used as ILOGIC2s:                      32
    Number used as ISERDES2s:                      0
  Number of IODELAY2/IODRP2/IODRP2_MCBs:           0 out of     586 0%
  Number of OLOGIC2/OSERDES2s:                     4 out of     586 1%
    Number used as OLOGIC2s:                       4
    Number used as OSERDES2s:                      0
  Number of BSCANs:                                0 out of       4 0%
  Number of BUFHs:                                 0 out of     384 0%
  Number of BUFPLLs:                               0 out of       8 0%
  Number of BUFPLL_MCBs:                           0 out of       4 0%
  Number of DSP48A1s:                              0 out of     180 0%
  Number of ICAPs:                                 0 out of       1 0%
  Number of MCBs:                                  0 out of       4 0%
  Number of PCILOGICSEs:                           0 out of       2 0%
  Number of PLL_ADVs:                              0 out of       6 0%
  Number of PMVs:                                  0 out of       1 0%
  Number of STARTUPs:                              0 out of       1 0%
  Number of SUSPEND_SYNCs:                         0 out of       1 0%
```

Příloha D

Obsah přiloženého CD

Adresář `doc`

Adresář `doc` obsahuje zdrojové kódy textu této práce a použité obrázky ve formátu pdf. Součástí je také soubor `Makefile`, který umožňuje práci vysázet s využitím systému LaTeX.

Adresář `src`

Adresář `src` obsahuje zdrojové kódy implementace navržené akcelerační jednotky a jejích podkomponent. Hierarchie jednotlivých komponent je popsána v souboru `hierarchy.txt`. Adresář `src` rovněž obsahuje příslušné soubory potřebné pro otevření projektu ve vývojovém prostředí Xilinx ISE Design Suite 14.1.

Adresář `readme.txt`

Soubor `readme.txt` stručně popisuje adresářovou strukturu projektu.

Adresář `xkoran01-dp.pdf`

Soubor `xkoran01-dp.pdf` je vysázený text diplomové práce s funkčními odkazy.