Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Energeticky efektivní ověřování v bezdrátových sítích

Plzeň 2017 Petr Podávka

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 18. května 2017

Petr Podávka

Poděkování

Rád bych poděkoval Ing. Jiřímu Ledvinovi, Csc. za cenné rady, věcné připomínky a vstřícnost při konzultacích a vypracování diplomové práce.

Abstract

Master thesis deals with energy efficient cryptographic protocols suitable to be used in wireless sensor networks. Nodes of such networks consist of so called resource constrained devices having limited computing power, storage and capacity and limited lifespan caused by being powered from battery. Energy efficiency of used protocols is therefore crucial. Selected protocols designed for such use are presented in this thesis. These protocols are implemented on specific microcontroller and their time and power consumption is compared.

Abstrakt

Diplomová práce se věnuje energeticky efektivním kryptografickým protokolům, které jsou vhodné k použití v bezdrátových senzorických sítích. Jako uzly takových sítí jsou použita tzv. nízkonákladová zařízení představující výpočetně, paměťově a napěťově omezená zařízení. Energetická efektivita použitých protokolů je tedy klíčová. V této práci jsou představeny vybrané protokoly určené k tomuto použití. Tyto protokoly jsou dále implementovány na konkrétním mikrokontroléru a je srovnána časová, respektive energetická náročnost těchto implementací.

Obsah

1 Úvod

Schopnost ověřit identitu protistrany v komunikaci mezi uzly sítě je stejně důležitá, jako schopnost obsah této komunikace utajit. Výkonná zařízení mohou k tomuto ověření použít robustní kryptografické protokoly založené na asymetrické kryptografii, jakým je například protokol TLS. Uzly bezdrátových senzorických sítí však disponují malým výpočetním výkonem, malou pamětí ROM i RAM a omezenou životností v podobě napájení z baterie. Přesto je bezpečnost komunikace v bezdrátové senzorické síti neméně důležitá. Tento typ sítí je totiž použit nejenom v nekritických aplikacích, jakými jsou např. monitorování parkovacích míst v parkovacích domech, ale také v kritických aplikacích ve vojenství, zdravotnictví aj. Omezení uzlů bezdrátových senzorických sítí nedovolují použití standardních protokolů a vedou k výzkumu nových efektivních protokolů, většinou založených na symetrické kryptografii.

Cílem této práce je prozkoumat dostupné metody ověření s přihlédnutím na použití v bezdrátových senzorických sítích. Vybrané kryptografické protokoly budou implementovány na poskytnutém mikrokontroléru a porovnány z hlediska časové, respektive energetické, náročnosti.

2 Teoretická část

2.1 Bezpečná komunikace

Bezpečná komunikace přes libovolné médium musí splňovat několik základních požadavků:

- **Důvěrnost** zamezuje neoprávněnému subjektu čtení obsahu komunikace.
- **Integrita** zajišťuje celistvost a nezměnitelnost zpráv.
- **Autentičnost** ověřuje identitu autora zpráv.
- **Nepopiratelnost** vylučuje možnost popřít provedenou operaci.

Všechny tyto atributy mohou být zajištěny pomocí primitiv ze symetrické a asymetrické kryptografie. Symetrická kryptografie používá sdílený tajný klíč k šifrování i dešifrování zprávy. Tím lze zajistit *důvěrnost*, avšak již ne *autentičnost*, protože každý držitel tajného klíče zašifruje stejnou zprávu do shodné podoby a nelze tedy rozhodnout, která strana je autorem. Asymetrická kryptografie využívá dvojici klíčů, soukromý a veřejný, kde veřejný klíč je odvozený z klíče soukromého. Zároveň není možné zpětně z veřejného klíče odvodit klíč soukromý. Asymetrickou kryptografii lze využít nejenom k zašifrování zprávy a tím zajistit *důvěrnost*, ale také k podepsání zprávy a ověření tohoto podpisu a tím dosáhnout *autentičnosti*, *nepopiratelnosti* a *integrity*. Veřejným klíčem je zpráva šifrována a soukromým klíčem je dešifrována. Podpis se naopak vytváří soukromým klíčem a jeho ověření probíhá pomocí veřejného klíče. Algoritmy asymetrické kryptografie jsou obecně časově a energeticky náročnější než algoritmy symetrické kryptografie. Běžná praxe, nejenom na zařízeních s omezenými prostředky, tedy je, že je komunikace šifrována pomocí symetrické kryptografie. Aby bylo možné komunikaci šifrovat algoritmem symetrické kryptografie, musí obě komunikující strany znát společný tajný klíč. V 70. let 20. století byl publikován Diffieho–Hellmanův protokol výměny klíčů, který umožňuje vytvoření sdíleného tajemství (klíče) po nechráněném komunikačním kanálu [2]. Do té doby bylo nutné klíče distribuovat po zabezpečeném médiu např. při osobním setkání či důvěryhodnou kurýrní službou.

Další rozdíl mezi symetrickou a asymetrickou kryptografií spočívá v délce klíče, který je pokládaný za bezpečný. Podle společnosti *RSA Security*[1] odpovídá 1024-bitový RSA klíč 80-bitovému klíči symetrické šifry, 2048-bitový RSA klíč 112-bitovému symetrickému klíči a 3072-bitový RSA klíč 128-bitovému symetrickému klíči. Algoritmy založené na ECC (Eliptic Curve Cryptography) jsou bezpečné s mnohem kratšími klíči. Podle instituce NIST[2] by měli být ECC klíče dvakrát tak dlouhé než jejich symetrický ekvivalent. 224-bitový ECC klíč je pak tedy stejně bezpečný jako 112-bitový symetrický klíč [10].

Důvěrnost je dosažená zašifrováním zprávy. Šifrovanou zprávu může přečíst pouze subjekt disponující odpovídajícím klíčem a je to rozšířený bezpečnostní prvek používaný v mnoha komunikačních protokolech používaných v telekomunikačních sítích včetně internetu a IoT (*Internet of Things*).

Integrita zprávy značí, že její obsah není upraven během transportu. Jako nejjednodušší prostředek zajištění integrity zprávy lze uvést vytvoření otisku zprávy odesílatelem pomocí hašovací funkce, bezpečné odeslání otisku příjemci spolu se zprávou a kontrolu otisku zprávy na straně příjemce vůči otisku obdrženému od odesílatele.

Hašovací funkce převádějí data libovolné délky na data konstantní délky. Tento výstup se pak nazývá výtah, otisk, fingerprint či hash (česky haš). V případě kryptografické hašovací funkce se dále očekává, že je funkce jednosměrná.

Autentičnost zprávy, tedy jistotu, že je zpráva skutečně odeslaná odesílatelem, může zaručit např. MAC funkce (Message Authentication Code). MAC funkce je podobná hašovací funkci s tím rozdílem, že výsledek funkce není vytvořen jenom zpracováním vstupních dat, ale také klíče. MAC funkci lze zkonstruovat z kryptografických primitiv jako např. kryptografické hašovací funkce (HMAC) nebo blokových šifer (OMAC, PMAC). Proto MAC funkce zároveň z principu zajišťuje integritu zprávy, protože její výstup nelze bez tajného klíče reprodukovat.

Nepopiratelnost znamená možnost dokázat, že autor zprávy je skutečně jejím autorem. Toho lze dosáhnout např. vytvořením podpisu zprávy pomocí soukromého klíče, protože soukromý klíč zná pouze podepisující strana a nikdo jiný. Ze stejného důvodu podpis zprávy přináší *autentičnost* zprávy a také

[1] https://www.rsa.com/en-us

[2] https://www.nist.gov/

integritu zprávy, protože upravená zpráva nebude odpovídat dodanému podpisu.

2.2 Zařízení s omezenými prostředky

Vestavěné systémy disponují řídící jednotkou, která je ve většině případů jednoúčelová a určená pro předem definované činnosti. Je tedy možné tyto systémy optimalizovat již při návrhu systému výběrem adekvátního, nepředimenzovaného hardware a tím snížit jejich cenu. Takový hardware zahrnuje i nízkonákladová zařízení v podobě levných mikrokontrolérů s omezenou paměťovou kapacitou a výpočetním výkonem. Mezi vestavěné systémy patří bankomaty, mobilní telefony, zdravotnické přístroje, routery, tiskárny, klimatizační jednotky atd. Takové systémy jsou většinou řízeny řídící jednotkou s vlastním firmwarem.

Mikrokontroléry bývají označovány jako jednočipové mikropočítače. V jednom pouzdře se nacházejí všechny důležité součásti mikropočítače: procesor (aritmeticko-logická jednotka a řadič), paměť programu (EPROM, flash nebo ROM), zapisovatelná volatilní paměť dat RAM, řadiče přerušení. Bývá zvykem, že mikrokontroléry obsahují také vlastní generátor hodinového signálu a mohou obsahovat i další technické prostředky jako A/D a D/A převodníky, watchdog, programátor, čítače a časovače. Mikrokontrolér může mít prostředky pro rozšíření o další periferní obvody jako např. paralelní vstupní a výstupní I/O porty, sériová rozhraní, čítače a časovače. Určité typy mikrokontrolérů mají k dispozici i další specifické moduly, které plně nebo částečně hardwarově implementují různé kryptografické algoritmy, např. AES, DES, 3DES, RSA, ECC apod., hašovací funkce, např. MD5, SHA-1, SHA-2 atd., případně generátor náhodných čísel. Různé mikrokontroléry používají různé procesory a tedy různé instrukční sady. Pro lepší přenositelnost kódu mezi jednotlivými typy procesorů se zdrojové kódy píší v univerzálních nízkoúrovňových jazycích, případně makrojazycích. Nejčastěji je použit jazyk C, který je dobře přenositelný mezi různými architekturami.

Mezi nízkonákladová zařízení bychom zařadili též čipové karty nebo RFID čipy (Radio Frequency Identification). Jednoduché čipové karty obsahují integrovaný obvod, který je schopný zpracovávat data. Nejčastěji jsou takové karty použity pro spolehlivou a bezpečnou autentizaci skrze digitální identifikaci, kdy se autentizuje identita karty. Nejběžnějším příkladem autentizace je ověření veřejným klíčem PKI (*Public Key Infrastructure)*. V paměti karty je

uložený zašifrovaný digitální certifikát s relevantními informacemi od poskytovatele. Pro urychlení autentizace mohou karty obsahovat i kryptografický koprocesor, který provede kryptografické výpočty mnohonásobně rychleji než standardní univerzální mikroprocesor. Kontaktní čipové karty mají kontakt o velikosti asi jeden centimetr čtvereční nebo menší a kontaktní plošky jsou pozlacené kvůli lepšímu přenosu energie a signálu. Karty nejsou vybaveny vlastním zdrojem proudu. Místo toho jsou jejich obvody napájeny z terminálu, ve kterém je čtečka zasunuta. V případě bezkontaktních čipových karet jsou obvody napájeny bezdrátově indukcí z terminálu. Čipové karty jsou použity v SIM, kreditních kartách, ale také v různých přístupových kartách nebo elektronických peněženkách.

RFID čipy jsou určeny k jednoznačné bezkontaktní identifikaci na krátkou vzdálenost. RFID čip obsahuje 96-bitové unikátní identifikační číslo, takzvané EPC (*Electronic Product Code)*. Toto číslo se přiděluje centrálně výrobcům v jednotlivých řadách a je dlouhé 96 bitů. RFID čipy mohou být pasivní nebo aktivní. V případě pasivních čipů vysílá snímač do okolí elektromagnetické pulzy. Pokud se pasivní čip dostane do dosahu tohoto pole, využije přijatou energii k nabití napájecího kondenzátoru a odešle odpověď, většinou EPC. Aktivní čipy mají vlastní zdroj napětí a vysílají aktivně informace do okolí sami bez ohledu na to, zda jsou v dosahu nějakého přijímače.

2.3 Bezdrátové sítě

Uzly mohou být v počítačové sítí propojeny vodiči či optickými kabely. Cena takových sítí je však navýšena o cenu použitých kabelů a vedení kabelu není vždy jednoduché. Bezdrátová síť je použita tam, kde je kladen důraz na mobilitu, případně na jednoduchost spojení, např. mezi patry dvou u sebe stojících budov. Tak jako existují různé topologie drátových sítí, lze definovat topologie bezdrátových sítí:

- **Hvězda** (Star) – je používána např. Wi-Fi (IEEE 802.11) a připomíná topologie ethernetu používající switch. Roli switche zde zastává přístupový bod (AP – Access Point), ke kterému se jednotlivé uzly připojují a přes který spolu komunikují.
- **Klient-klient** (P2P – Peer-to-peer) – spojuje dvě zařízení mezi sebou a používá jí například Bluetooth. Tato topologie je vhodná především pro dočasné propojení dvou zařízení za účelem výměny dat. Přestože

je zmiňovaný Bluetooth používán nejčastěji v P2P módu, podporuje i topologii hvězda. Podobně i zařízení Wi-Fi mohou být propojeny jen mezi sebou jako klient-klient.

- **Mesh** – sestává z klientů, směrovačů a bran. Klienty v této sítí mohou být zařízení jako notebooky, telefony či tablety. Ty jsou připojeny ke směrovačům, jenž jsou propojeny redundantně mezi sebou a směrují data mezi zařízeními, případně k bráně, která je připojena k internetu.

Data mohou být mezi zařízeními vysílána pouze do jednoho uzlu (unicast), do více vybraných uzlů (anycast), do všech uzlů (broadcast) nebo do všech uzlů, které chtějí data přijímat (multicast). Speciálním případem je agregace dat (convergecast) v síti s topologií stromu, kdy uzel přijímá data od svých podřízených uzlů a hormadně je posílá do kořenového uzlu skze své nadřazené uzly. Podle velikosti a dosahu bezdrátové sítě lze definovat následující kategorie:

- **WCN** (Wireless Cellular Network) – Rádiová síť složená z buněk (angl. *cells*), které reprezentují území pokryté signálem. Každou buňku obsluhuje alespoň jedna fixní základnová stanice (angl. *base station*). Jednotlivé buňky se překrývají, ale pracují na odlišných frekvencích, aby se zamezilo vzájemnému rušení. Tímto způsobem může být pokryto signálem velice rozsáhlé území. Jednotlivá zařízení pak mohou zůstat připojená v síti, i když se pohybují mezi jednotlivými buňkami. Původně byly tyto sítě vybudovány pro hlasovou komunikaci, tedy mobilní telefony. S příchodem chytrých telefonů vznikly protokoly umožňující přenos dat v této síti a také připojení WCN do internetu. Systém založený na rozdělení pokrytého území na buňky využívají např. technologie GSM, EDGE, HSPA, CDMA2000 a jiné.
- **WPAN** (Wireless Personal Area Network) – Rozsah tohoto typu sítě pokrývá relativně malou oblast, obvykle v rámci dosahu člověka. Taková síť je určená pro propojení osobních zařízení, jako například chytrého telefonu a handsfree modulu v automobilu, chytrých hodinek nebo televize nebo notebooku a bezdrátové myši. Nepředpokládá se připojení této sítě do internetu, ale možné to je. Tuto síť používají technologie Bluetooth, ZigBee, NFC, IrDA a jiné. V posledních letech se do oblasti WPAN prosazuje i technologie Wi-Fi, např. v *Intel My Wifi*, jenž umožňuje přenos videa nebo obrázků z počítače do televize.
- **WLAN** (Wireless Local Area Network) – Tato bezdrátová síť spojuje dvě a více zařízení na vzdálenost do 100 metrů, obecně v budovách, ale

i mimo ně. Zařízení komunikující v této síti mohou být počítače, notebooky, tablety, chytré telefony atd. WLAN je obvykle propojená do internetu. Nejčastěji WLAN využívá technologii Wi-Fi (IEEE 801.11), která může pracovat v ad-hoc módu, tedy decentralizovaně, nebo v infrastrukturním módu, ve kterém jsou klientská zařízení připojena k síti přes statický přístupový bod. WLAN sítě mezi sebou mohou být propojeny na větší vzdálenost skrze dedikované mikrovlnné směrové antény, případně modulovaný laserový paprsek. Tento přístup vyžaduje, aby oba spojované body byli v přímém dohledu. Tento postup se často používá ve městech pro propojení sítí dvou budov bez nutnosti instalace kabelové linky a výsledná síť se může označovat jako WMAN nebo WWAN.

- **WMAN** (Wireless Metropolitan Area Network) – Tímto termínem se označuje typ sítě bezdrátově propojující dvě a více LAN sítí na vzdálenosti do 50 km. Sítě mohou být propojeny do WMAN v rámci jednoho města, avšak WMAN pokrývá i spojení sítí mezi městy, pokud jsou od sebe přiměřeně vzdálená. Nejpoužívanějším standardem pro WMAN sítě je WiMAX (IEEE 802.16). Tato technologie je používána především poskytovali (angl. *IPS – Internet Service Provider*), pro poskytnutí konektivity na velkou vzdálenost, tzv. překlenout poslední míli. WiMAX může pracovat se spojit typu *Point-MultiPoint*, ve kterém je k jedné pevně umístěné základnové stanici připojeno více koncových terminálů, které mohou být mobilní, avšak pouze v rámci dosahu základnové stanice. Existuje i varianta tzv. mobilního WiMAX, která cíleně podporuje mobilitu koncových uzlů. Tato varianta je považována za přímého konkurenta mobilních 3G sítí (WCN), neboť na rozdíl od Wi-Fi byl WiMAX navržen s ohledem na podporu QoS (*Quality of Service*) a tedy nabízí jak vysokou rychlost přenosu dat, tak i hlasové služby.

- **WWAN** (Wireless Wide Area Network) – Sítě typu WWAN obvykle pokrývají velkou plochu nebo spojují menší sítě na velké vzdálenosti. Sítě typu WWAN mohou být použity např. pro propojení firemních pobočkových sítí. Bezdrátové propojení mezi dvěma přístupovými body je většinou realizováno pomocí dedikovaných mikrovlnných spojů využívajících směrové antény.

Topologie IoT sítí mohou kombinovat různé uvedené přístupy a velikost takové sítě může být též různá. Senzorická síť v chytrém domě bude nejspíš sestávat maximálně z několika desítek čidel, pravděpodobně bude mít topo-

logii *hvězda* a velikostí bude odpovídat síti LAN. Naproti tomu senzorická síť farmy pro sledování prostředí pěstovaných plodin nebo např. síť určená pro sledování volných parkovacích míst v ulicích města se bude skládat z tisícovek senzorů, bude mít pravděpodobně *mesh* topologii a velikostí bude odpovídat WMAN nebo dokonce WWAN.

2.4 Symetrická kryptografie

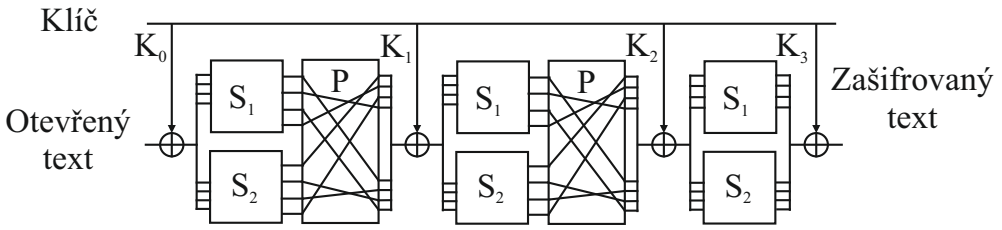
Symetrická kryptografie používá k šifrování i dešifrování dat tentýž klíč. Podstatnou výhodou symetrických šifer je jejich nižší výpočetní náročnost ve srovnání s šiframi asymetrickými. Nevýhodou je však problém sdílení tajného klíče, který musí být mezi komunikujícími stranami domluven dopředu po zabezpečeném kanálu. Jako další nevýhodu lze uvést, že symetrická šifra implicitně poskytuje pouze *důvěrnost* zašifrovaných dat. Symetrické šifry lze rozdělit na *blokové* a *proudové*.

2.4.1 Blokové symetrické šifry

Bloková šifra je deterministický algoritmus pracující nad skupinou bitů pevné délky, tzv. blok. Velikost bloků záleží na konkrétním algoritmu, ale běžně je to 64 nebo 128 bitů. Šifrovaná data jsou rozdělena do bloků konstantní velikosti dané použitým algoritmem. Poslední blok je v případě potřeby doplněn o tzv. výplň (angl. *padding*), tedy předem určenou posloupnost bitů. Moderní blokové šifry jsou založeny na konceptu iterování produkčních šifer, což jsou dvě a více transformací vstupních dat zkombinovaných za účelem zesílení odolnosti šifry vůči kryptoanalýze, konkrétně znemožnění nalezení vztahu mezi použitým tajným klíčem a zašifrovaným textem. Princip produkčních šifer navrhl C. Shanon v roce 1949 ve své práci *Communication Theory of Secrecy Systems* [18]. Dvě nejpoužívanější verze produkčních šifer jsou *substitučně-permutační síť*, zkráceně *SPN* z anglického *Substitution-Permutation Network*, a *Feistelova síť* [12].

SPN, použitá například v šifrovacím algoritmu AES, je posloupnost matematických operací, které převedou otevřený text a klíč na zašifrovaný text. Šifrování probíhá v několika kolech. V každém kole jsou použity substituční boxy (S-box) a permutační boxy (P-box). S-box zajišťuje substituci bitových posloupností, tedy blok bitů nahradí za jiný blok bitů stejné délky. P-box

provádí permutaci bitů ve vstupním bloku dat. V každém kole je dílčí klíč, získaný z hlavního klíče např. pomocí S-boxu nebo P-boxu, zkombinovaný se vstupem logickou operací, např. XOR, viz obrázek 2.1. Dešifrování je provedeno obrácením pořadí operací, pořadí použitých klíčů a použitím inverzních verzí S-boxů a P-boxů.



Obrázek 2.1: Návrh substitučně-permutační sítě se třemi koly.

Feistelovu síť používá například algoritmus DES. Základní princip tkví v opakovaném použití operace XOR v kombinaci s funkcí F, aplikované na jednu polovinu šifrovaného bloku a podklíč K_i. Klíč K je rozdělen na n podklíčů $k_1, k_2, ..., k_n$, kde n představuje počet iterací algoritmu. Otevřený text šifrovaného bloku dat se rozdělí na dvě poloviny (L_0, R_0) a v každém kole se provádí následující operace:

$$L_{i+1} = R_i$$
$$R_{i+1} = L_i \oplus F(R_i, K_i)$$

Výsledným šifrovaným textem je výstup posledního kola v opačném pořadí: (R_n, L_n). Výhodou *Feistelovo sítě* je, že funkce F nemusí být inverzní, na rozdíl od *SPN*, kde použité S-boxy i P-boxy inverzní být musí. k dešifrování se používá naprosto identický postup se stejnou funkcí F, pouze se otočí pořadí použitých podklíčů K_i. Konkrétní šifry založené na *Feistelově síti* se pak od sebe liší počtem kol, definicí funkce F a postupem, jakým se z klíče odvozují jednotlivé podklíče [12].

2.4.2 Proudové symetrické šifry

Proudové šifry zpracovávají vstupní text po jednotlivých bytech, případně bitech. Bity otevřeného textu jsou kombinovány s pseudonáhodným proudem bitů (angl. *keystream*). Kombinace je tvořena typicky operací XOR. Pseudonáhodný proud bitů je vytvořen na základě tajného klíče a šifrovacího algoritmu. Proudové šifry se podle způsobu tvoření pseudonáhodného proudu bitů dělí na synchronní a samosynchronní proudové šifry [6]

V synchronní proudové šifře je proud pseudonáhodných čísel generován nezávisle na vstupním textu nebo zašifrované zprávě. Generování proudu bitů je závislé pouze na stavu šifrovacího algoritmu a na tajném klíči. Při používání synchronních proudových šifer musí být komunikující strany synchronizovány, tedy musí sdílet tajný klíč a stav algoritmu proudové šifry. Synchronizace je ztracena, dojde-li v procesu šifrování k přidání nebo odebrání znaku ze zprávy. Pro obnovu synchronizace je možné opatřit šifrovaný text v pravidelném intervalu značkami. Dojde-li při přenosu šifrované zprávy ke změně jednoho znaku za jiný, bude při dešifrování ovlivněn pouze tento znak. Tato vlastnost způsobuje, že jsou synchronní proudové šifry náchylné k aktivním útokům – pokud útočník může změnit jedno číslo v šifrovaném textu, mohl by být schopen provést předvídatelné změny na odpovídajícím bitu otevřeného textu. Samosynchronní proudové šifry využívají ke generování pseudonáhodného proudu bitů tajný klíč a n předcházejících znaků šifrovaného textu. Dojde-li ke ztrátě nebo změně jednoho znaku šifrovaného textu, chyba se projeví v n následujících znacích, avšak poté se šifrovací algoritmus sám sesynchronizuje [6].

Proudové symetrické šifrovací algoritmy jsou v zásadě rychlejší než blokové symetrické šifrovací algoritmy. Šifrování i dešifrování probíhá znak po znaku a nemusí se tak čekat na příjem celého bloku dat. Mezi nevýhody patří nižší úroveň difuze, kdy zašifrovaný text může vykazovat stejné frekvenční a statistické charakteristiky jako původní otevřený text, což usnadňuje kryptoanalýzu. Dále nejsou proudové šifry odolné vůči aktivním útokům. V případě prolomení šifry může být přenášený text modifikován, aniž by to příjemce rozpoznal [20].

2.5 Asymetrická kryptografie

Asymetrická kryptografie je kryptografický systém, který používá dvojici klíčů: soukromý klíč, jenž je znám pouze vlastníkovi klíče, a veřejný klíč, který je volně distribuován. Soukromý klíč může být matematicky odvozen z veřejného, v některých algoritmech je to naopak. Bezpečnost asymetrického kryptografického systému je založena na obtížnosti určení soukromého klíče z odpovídajícího veřejného klíče. Asymetrická kryptografie umožňuje dosáhnout *autentičnosti*, když je veřejný klíč použit k ověření, že zprávu odeslal vlastník odpovídajícího soukromého klíče, a také *důvěrnosti*, neboť pouze vlastník soukromého klíče může rozšifrovat zprávu zašifrovanou odpovídajícím veřejným klíčem.

Algoritmy asymetrické kryptografie jsou založeny na matematických problémech, pro které v současné době neexistuje efektivní řešení. Mezi tyto problémy patří prvočíselný rozklad celých čísel nebo řešení diskrétních logaritmů [19].

Výhodou asymetrického kryptografického systému je schopnost vytvořit sdílené tajemství přes zcela nezabezpečený kanál. Nevýhodou je, že pro dosáhnutí *důvěrnosti* musí ověřující strana nejprve autentizovat odesílatele a vyloučit tak útok typu *Man in the middle* (*MITM*). Z tohoto důvodu byla zavedena rozsáhlá globální infrastruktura veřejných klíčů, která je popsána v kapitole 2.5.3. Mezi další nevýhody patří větší velikost klíčů, než je potřeba pro symetrickou kryptografii, a také vyšší výpočetní nároky [19, 27].

2.5.1 RSA

RSA je prvním prakticky použitelným algoritmem asymetrické kryptografie vhodným jak pro šifrování, tak i podepisování. Název RSA je vytvořen z prvních písmen příjmení vědců Ron Rivest, Adi Shamir a Leonard Adleman, kteří algoritmus poprvé veřejně popsali v roce 1977. Bezpečnost algoritmu RSA je postavena na problému prvočíselného rozkladu celých čísel (faktorizace), který je považován za obtížný. Není totiž znám žádný algoritmus faktorizace, jenž by zjistil z čísla $n = pq$ činitele p a q a který by pracoval v polynomiálním čase vůči velikosti binárního zápisu čísla n [28].

Algoritmus se skládá ze čtyř kroků: vytvoření klíčového páru, distribuce klíče, šifrování a dešifrování.

Vytvoření klíčového páru

1. Zvol dvě náhodná prvočísla p a q.
2. Spočítej jejich součin $n = p \cdot q$. Číslo n je použito jako modulo pro soukromý i veřejný klíč. Bitová délka čísla n je označována jako *délka klíče*.
3. Spočítej hodnotu Carmichaelovy funkce

$$\lambda(n) = \mathrm{NSN}(\lambda(p), \lambda(q)) = \mathrm{NSN}(p-1, q-1),$$

   kde λ je Carmichaelova funkce a NSN je nejmenší společný násobek.

4. Zvol náhodné celé číslo $e; e < \lambda(n)$, které je nesoudělné s $\lambda(n)$.
5. Nalezni číslo d podle rovnice

$$d \equiv e^{-1} (\text{mod } \lambda(n)),$$

kde symbol $\equiv$ značí kongruenci zbytkových tříd. K nalezní čísla d lze využít *rozšířený Euklidův algoritmus.*

Veřejný klíč je pak dvojice čísel (n,e). Soukromý klíče je dvojice čísel (n,d). Čísla p, q a $\lambda(n)$ nesmí být prozrazena, neboť znalost těchto čísel lze využít k výpočtu soukromého d [11].

Distribuce klíče

Předpokládejme, že *Bob* chce odeslat tajnou zprávu *Alici*. Pro zašifrování zprávy musí *Bob* znát *Alicin* veřejný klíč (n, e). Tento klíč může *Alice* odeslat *Bobovi* přes *spolehlivý*, ne však nutně bezpečný kanál [11].

Šifrování

Protože RSA šifrování je deterministický algoritmus, může útočník zašifrovat vybraný pravděpodobný otevřený text veřejným klíčem a testovat shodnost vůči analyzovanému zašifrovanému textu (CPA – *Chosen-plaintext attack)*. Tento nedostatek je odstraněn posunutím zprávy M o náhodný počet náhodných bitů před vlastním šifrováním, čímž je vytvořena zpráva m. Tato úprava musí být při komunikaci dopředu dohodnuta mezi komunikujícími stranami a musí být reverzibilní. Toto posunutí zaručuje, že m nebude spadat do rozsahu ne-bezpečných textů, a že se stejná zpráva po posunutí zašifruje do různých zašifrovaných textů. Zpráva m je převedena na šifrovaný text c za použití *Alicina* veřejného klíče podle rovnice [11]

$$c \equiv m^e \pmod{n}.$$

Dešifrování

Zašifrovaná zpráva c může být převedna zpět na m použitím soukromého klíče d:

$$c^d \equiv (m^e)^d \equiv m \pmod{n}.$$

Obrácením metody pro posun bitů je z m získána zpráva M [11].

Podepisování

Předpokládejme, že *Alice* pošle *Bobovi* zprávu zašifrovanou jeho veřejným klíčem e. V takové situaci *Bob* nemůže nijak ověřit, že zprávu skutečně zašifrovala *Alice*, protože kdokoliv může použít *Bobův* veřejný klíč e. K zajištění *autentičnosti* zprávy může být RSA použito také k jejímu podepsání.

Alice chce odeslat *Bobovi* podepsanou zprávu M. *Alice* vytvoří haš h zprávy M a vytvoří podpis $s = h^d (\mathrm{mod}\ n)$. *Bobovi* odešle zprávu M a podpis s. Pro ověření musí *Bob* použít stejný hašovací algoritmus pro vytvoření vlastního haše h_B zprávy M. Původní podepsaný haš h získá použitím *Alicina* veřejného klíče jako $h = s^e (\mathrm{mod}\ n)$ a porovná vlastní haš h_B a získaný haš h. Pokud jsou oba haše stejné, je zřejmé, že autor podpisu a tedy i zprávy je držitelem *Alicina* soukromého klíče.

2.5.2 ECC

Zkratka ECC je zkratka anglického slovního spojení *Elliptic Curve Cryptography*, česky *kryptografie nad eliptickými křivkami*. ECC je algoritmus asymetrické kryptografie založený na algebraických strukturách eliptických křivek nad konečnými tělesy. Použití eliptických křivek v kryptografii navrhli nezávisle na sobě Neal Koblitz a Victor S. Miller v roce 1985 [26]. ECC je, podobně jako RSA, vhodné pro šifrování, podepisování i výměnu klíče přes nezabezpečený kanál. Pro zajištění stejné bezpečnosti vyžaduje menší velikost klíče oproti algoritmům asymetrické kryptografie založených na jiných problémech, např. RSA. Přehled velikostí klíčů používaných NSA pro vybrané algoritmy viz tabulka 2.1.

Tabulka 2.1: Velikosti klíčů používaných agenturou NSA v roce 2016 [21].

Algoritmus	Velikost klíče
RSA	3072 bitů nebo delší
EDSA	384 bitů (NIST P-384[3])
AES-256	256 bitů
SHA-384	384-bitové bloky

V ECC je eliptická křivka rovinná křivka definovaná nad konečným tělesem celých čísel, jež se skládá z bodů vyhovujících rovnici

$$y^2 = x^3 + ax + b, \tag{2.1}$$

spolu s definovaným nevlastním bodem označeným znakem ∞. Charakteristika použitého konečného tělesa nesmí být rovna 2 nebo 3, protože by výše uvedená rovnice neplatila [13]. Množina řešení rovnice 2.1 tvoří společně se skupinou operací nad teorií množin eliptických křivek Abelovu grupu s bodem ∞ jako neutrálním prvkem.

Před použitím ECC se musí všechny zúčastněné strany dohodnout na parametrech použité křivky (p, a, b, G, n, h) [16].

- p – prvočíslo určující velikost konečného tělesa.
- a, b – koeficienty rovnice eliptické křivky.
- G – bod, jenž generuje cyklickou podgrupu.
- n – řád cyklické podgrupy, tedy vyhovující rovnici $nG = 0$. Pro kryptografické účely musí být n prvočíslo.
- h – kofaktor vypočítaný podle Lagrangeovo věty $h = \frac{1}{n}|E(\mathbb{F}_p)|$. V kryptografických aplikacích by mělo být $h \leq 4$, nejlépe však $h = 1$.

Pokud není jisté, že parametry křivky byly vygenerovány důvěryhodnou stranou, musí být před použitím vždy ověřena jejich správnost. Generování parametrů křivky běžně neprobíhá na koncových uzlech, neboť vyžaduje časově náročné výpočty bodů na křivce. Namísto toho se používají předgenerované parametry pro různé velikosti konečného tělesa, jenž byli zveřejněny několika známými instituty [16]:

- NIST, *Recommended Elliptic Curves for Government Use.*
- SECG, *SEC 2: Recommended Elliptic Curve Domain Parameters.*
- ECC Brainpool (RFC 5639), *ECC Brainpool Standard Curves and Curve Generation.*

[3] Jedna z doporučených křivek standardizovaných institutem NIST.

Vytvoření klíčového páru

Soukromý klíč d je náhodné číslo vybrané z množiny $\{1, ..., n-1\}$, tedy $0 < d < n$. Veřejný klíč je bod na křivce $Q = dG$. Zjištění soukromého klíče d je obtížné, protože vyžaduje vyřešení problému diskrétního logaritmu [16].

Distribuce klíče

Stejně jako v případě RSA i veřejný klíč Q může být odeslán ostatním účastníkům přes spolehlivý, ale ne nutně zabezpečený kanál [16]. Je také zřejmé, že všichni účastníci musí používat stejné parametry eliptické křivky. V případě použití pojmenované křivky je její identifikátor odeslán spolu s veřejným klíčem Q. Pokud není použita standardní křivka, musí být s klíčem Q zveřejněny všechny parametry použité eliptické křivky, tedy šestice (p, a, b, G, n, h).

Šifrování

Pro zašifrování zprávy m je nejprve vygenerováno náhodné číslo r takové, že $0 < r < n$, a je vypočítán odpovídající bod na křivce $R = rQ$. Bod R je nyní veřejně známý a je přenesen spolu se zprávou m, která je zašifrována vybraným symetrickým šifrovacím algoritmem a klíčem $S = rQ$ [16].

Dešifrování

Pro rozšifrování zprávy je nutné zjistit klíč S ze soukromého klíče d a veřejně známého bodu R podle rovnice $S = dR$. Proč jsou klíče S stejné lze dokázat jednoduchou substitucí [16]

$$S = dR = d(rG) = r(dG) = rQ = S.$$

Podepisování

Pro výpočet podpisu zprávy m jsou nutné následující kroky [16]:

1. Vytvoř haš h zprávy m.
2. Vyber náhodně číslo k takové, že $0 < k < n$.

3. Vypočítej $r = x \pmod{n}$, kde $(x, y) = kG$. Pokud $r = 0$, běž zpět na krok 2.
4. Vypočítej $s = k^{-1}(h + dr) \pmod{n}$. Pokud $s = 0$, běž zpět na krok 2.
5. Výsledný podpis zprávy m je dvojice (r, s).

Zpráva m je odeslána spolu s podpisem (r, s). K ověření musí mít ověřující účastník k dispozici veřejný klíč Q. Podpis je ověřen algoritmem [16]:

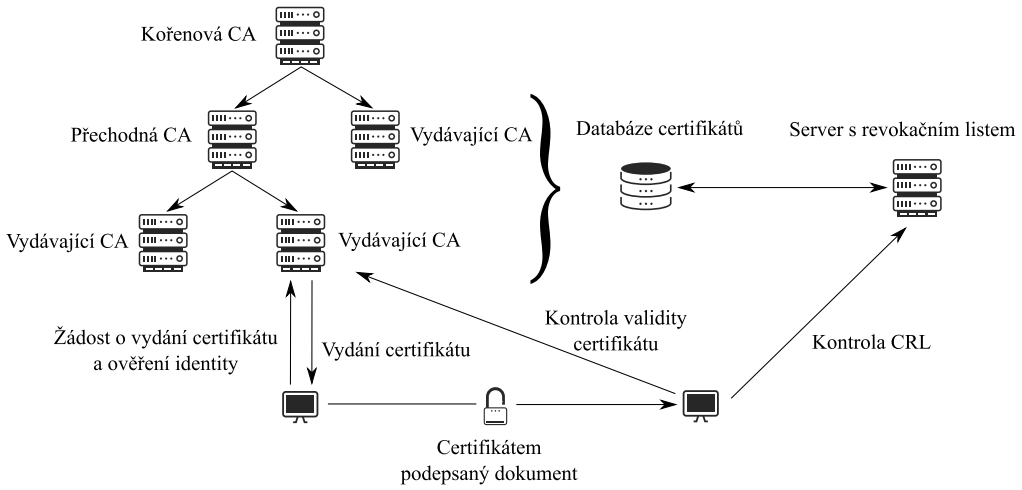
1. Ověř, že r a s jsou celá čísla taková, že $0 < r, s < n$. Pokud toto neplatí, podpis je nevalidní.
2. Vytvoř haš h zprávy m stejnou hašovací funkci, který byla použita na vytvoření haše při podepisování zprávy.
3. Vypočítej $w = s^{-1} \pmod{n}$.
4. Vypočítej $u_1 = hw \pmod{n}$ a $u_2 = rw \pmod{n}$.
5. Vypočítej $(x, y) = u_1G + u_2Q$.
6. Podpis je validní, pokud $x = r \pmod{n}$.

2.5.3 Infrastruktura veřejných klíčů

Předpokládejme, že subjekt *Alice* vlastní soukromý a veřejný klíč libovolného algoritmu asymetrické kryptografie, zde např. soukromý klíč Q a veřejný klíč d pojmenované křivky *NIST P-384*. Pokud *Alice* podepíše dokument m svým soukromým klíčem d, musí spolu s dokumentem a jeho podpisem publikovat i svůj veřejný klíč Q. Jiný subjekt může z dodaných informací ověřit, že podpis vytvořila *Alice* za předpokladu, že bude věřit, že dodaný klíč Q skutečně patří *Alici* a ne například útočníkovi, který se snaží za *Alici* vydávat. Tento problém řeší zavedení digitálního certifikátu, tedy elektronického dokumentu, jenž je použit k dokázání vlastnictví daného klíče. Digitální certifikát obsahuje veřejný klíč, informace o identitě subjektu, jemuž klíč patří, a podpis vytvořený důvěryhodnou třetí stranou, tzv. vydavatelem certifikátu.

Infrastruktura veřejných klíčů (zkráceno PKI z anglického *Public Key Infrastructure*) je sada pravidel, rolí a procesů, které jsou nutné pro vytváření, správu, distribuci, používání, ukládání a zneplatňování digitálních certifikátů. PKI váže veřejné klíče s jejich vlastníky (osoby nebo organizace) skrze proces registrace a vystavení certikátu vydavatelem [22, 1].

Přístup k PKI může být hierarchický nebo distribuovaný. V případě hierarchického uspořádání mluvíme o certifikačních autoritách (CA). Hlavní rolí CA je vytváření digitálních certifikátů, jejich přechovávání a distribuce. Vystavení certifikátu znamená vytvoření podpisu veřejného klíče žadatele privátním klíčem certifkační autority. Důvěryhodnost veřejného klíče žadatele je tedy přímo závislá na důvěryhodnosti certifikační autority. Každá kořenová CA se prokazuje certifikátem, jenž sama vystavila. Tento druh certifikátu se označuje jako *self-signed* a pokud patří CA, které uživatel důvěřuje, jsou všechny certifikáty vystavené touto CA také důvěryhodné. CA může vystavit certifikát subjektu, který dál vystupuje jako CA a vystavuje certifikáty. Taková CA je označována jako *intermediate CA*. Důvěryhodnost certikátů vystavených takovou CA je však pořád závislá na důvěryhodnosti kořenové CA. Znázornění hierarchie CA viz obrázek 2.2. Zneplatnění vystaveného certifikátu je provedeno přidáním záznamu na tzv. revokační seznam (CRL – *Certificate Revocation List*). Uživatel PKI tedy musí kontrolovat nejenom, že certifikát byl vystaven důvěryhodnou CA, ale také že se nenechází na některém z CRL [22, 23].



Obrázek 2.2: Hierarchické uspořádání certifikačních autorit.

Alternativa k hierarchicky uspořádaným CA je decentralizovaná síť důvěry (angl. *Web of Trust*). Princip je založen na důvěře v zúčastněné strany. Pokud například *Alice* důvěřuje *Bobovo* veřejnému klíči, může ho podepsat svým soukromým klíčem, čímž de facto vznikne certifikát *Bobovi* vystavený *Alicí*. Pokud *Bob* tento certifikát použije např. pro podepsání emailu poslanému entitě, jež důvěřuje *Alicině* veřejnému klíči, bude důvěřovat i *Bobovu* klíči. Na principu sítě důvěry je založen např. standard GnuPG [30].

2.5.4 Kryptografické protokoly relační vrstvy

Bezpečný přenos dat může být zajištěn na různých vrstvách ISO/OSI modelu. Na síťové vrstvě to může být např. protokol IPsec. Jedná se o rozšíření běžného IP protokolu, které autentizuje a šifruje každý datagram a tím transparentně zabezpečuje přenos pro všechny vyšší vrstvy. Naproti tomu bezpečnostní protokoly relační vrstvy, jakými jsou SSL (*Secure Sockets Layer*) a TLS (*Transport Layer Security*), musí být implementovány v jednotlivých aplikacích, které je chtějí využívat. Tyto protokoly jsou hojně využívány pro zabezpečení komunikace v internetu pro služby jako WWW, elektronická pošta, IM nebo VoIP [7].

TLS protokol má za úkol poskytnout důvěrnost a integritu mezi dvěma komunikujícími účastníky, např. mezi webovým prohlížečem a webovým serverem. Pokud je spojení zabezpečeno protokolem TLS, je těmito způsoby dosaženo jedné nebo více následujících vlastností [7]:

- **Důvěrnost** – veškerá komunikace přenášená protokolem TLS je šifrována dohodnutým symetrickým šifrovacím algoritmem. Klíč této šifry je unikátní pro každé spojení, tzv. *session*, a je dohodnut spolu s dalšími parametry před zahájením přenosu vlastních dat. Dohodnutí sdíleného klíče je bezpečné a spolehlivé, neboť klíč není nikdy přenášen přes síť ani není možné komunikaci ovlivnit bez vědomí komunikujích stran.
- **Autentičnost** – identitu komunikujících účastníků lze ověřit skze digitální certifikáty a PKI. Tento krok není povinný, ale je obecně vyžadován alespoň od jednoho účastníka (typicky webový server se prokáže digitálním certifikátem webovému prohlížeči).
- **Integrita** – každá odeslaná zpráva obsahuje výsledek MAC funkce. Jakákoliv úprava přenášených dat je tedy okamžitě detekována.

TLS protokol dále umožňuje dosáhnout dopředné bezpečnosti při použití odpovídajího algoritmu pro výměnu klíče. Dopředná bezpečnost zajišťuje, že komunikace nemůže být rozšifrována ani v případě, že dojde k pozdějšímu prozrazení soukromých klíčů komunikujících stran [4]. Dosažené vlastnosti jsou přímo závislé na zvoleném souboru šifer (angl. *cipher suite*), což je pojmenovaná kombinace autentizačních a šifrovacích algoritmů a autentizačních kódů MAC použitých v konkrétní relaci. Soubor šifer může být např.

`TLS_DHE_RSA_WITH_AES_256_GCM_SHA384`,

kde `DHE` specifikuje algoritmus výměny klíčů, `RSA` typ algoritmu autentizace, zde RSA certifikát, `AES_256_GCM` určuje symetrickou šifru použitou pro šifrování vlastní komunikace a délku klíče, případně další parametry, a konečně `SHA384` specifikuje hašovací funkci použitou pro zajištění integrity [7].

Protože aplikace můžou komunikovat i bez TLS, je nutné, aby klient indikoval serveru, že chce použít TLS. Jedna z možností, jak tohoto dosáhnout, je použití odlišného portu, např. 443 pro HTTPS. Dalším způsobem je odeslání specifické zprávy serveru, kterou mu dá klient na vědomí, že chce přepnout na TLS spojení, např. STARTTLS požadavek pro SMTP protokol. Jakmile jsou klient a server dohodnutí na použití TLS, vyjednají trvalé spojení pomocí *handshake* procedury. V rámci procedury *handshake* je použita asymetrická kryptografie pro dohodnutí použitého souboru šifer a relačního klíče. Zjednodušená *handshake* procedura se skládá z těchto kroků [7]:

1. Klient pošle serveru zprávu *ClientHello* specifikující nejvyšší klientem podporovanou verzi TLS, náhodné číslo a seznam podporovaných souborů šifer.
2. Server odpoví zprávou *ServerHello* obsahující vybranou verzi protokolu, náhodné číslo a vybraný soubor šifer ze seznamu poslaného klientem. Verze použitého protokolu by měla být vždy nejvyšší podporovaná serverem i klientem.
3. Server odešle svůj digitální certifikát k ověření identity.
4. Server odešle zprávu *ServerKeyExchange*, která obsahuje data potřebná pro výpočet finálního symetrického klíče, např. v případě použití DHE algoritmu (DH pracující s dočasnými klíči, určenými pouze pro použití v jedné relaci).
5. Server odešle zprávu *ServerHelloDone* indikující dokončení procedury *handshake*.
6. Klient odešle zprávu *ClientKeyExchange* obsahující v závislosti na dohodnutém souboru šifer veřejný klíč klienta, tajný klíč použitý pro generování relačního klíče nebo nic. V případě, že je odesílán tajný klíč, je zašifrován veřejným klíčem serveru.
7. Z dostupných údajů vypočítají klient i server relační klíč, jenž je použit pro šifrování vlastních dat odesílaných přes TLS.

V uvedeném procesu se ověřuje pouze identita serveru. Lze však ověřovat identitu obou stran za cenu mírně složitější procedury a výměny několika zpráv navíc [7].

Protokol TLS implementuje řada knihoven vydaných pod různými licencemi. Jako příklad těch rozšířených lze uvést OpenSSL, crpytlib, Bouncy Castle, GnuTLS nebo mbedTLS [25].

2.6 Protokoly správy a distribuce symetrických klíčů v bezdrátových senzorických sítích

Z textu předchozích kapitol je zřejmé, že algoritmy asymetrické kryptografie nejsou vhodné pro zabezpečení komunikace mezi uzly v bezdrátových senzorických sítích (WSN – *Wireless Sensor Network*). Složité algoritmy vyžadují více paměti ROM i RAM, výpočty jsou časově náročné a vzrůstá spotřeba energie, což je na zařízeních s tak omezenými prostředky, jakými jsou uzly WSN, nepřijatelné. Přesto existují implementace algoritmů asymetrické kryptografie (PKC – *Public Key Cryptography*) určené pro WSN, např. TinyECC [14] nebo TinyPK [24]. Tyto implementace obsahují složité optimalizace a asembly kód, který není univerzální pro všechna zařízení. Například TinyECC je určené pouze pro operační systém TinyOS a zdrojový kód je napsaný v jazyce nesC [14]. Portování takové implementace na zařízení nepodporující TinyOS a nesC je složité a obtížné.

Řešením výše uvedených problémů mohou být efektivní protokoly pro správu a distribuci symetrických klíčů (KMS – *Key Management Schemes*). Tyto protokoly mají za úkol vytvořit bezpečné spojení mezi dvěma uzly WSN za použití algoritmů symetrické kryptografie. Efektivní KMS protokoly by měly poskytnout *autentičnost*, *důvěrnost*, *integritu*, *škálovatelnost* a *flexibilitu* [29].

- **Autentičnost** – KMS protokol by měl poskytovat nástroje pro vzájemné ověření identity komunikujících uzlů.
- **Důvěrnost** – KMS protokol by měl zabránit prozrazení šifrovaných dat. V případě útoku na síť může útočník získat tajný klíč uzlu, aby rozšifroval dosud proběhlou komunikaci. Pokročilý KMS protokol zajistí, že žádná další data nebudou kompromitována, například revokací klíče.
- **Integrita** – KMS protokol zajistí, že nemůže dojít k úpravě dat během

přenosu. Pouze uzly sítě mají přístup k symetrickým klíčům a pouze dedikovaná základnová stanice má právo klíč zneplatnit nebo aktualizovat. Tím se efektivně zabrání neautorizovaným uzlům v získání informací o klíčích v síti.

- **Škálovatelnost** – Efektivní KMS protokol musí poskytovat dostatečnou bezpečnost pro malé WSN a zároveň si stejné vlastnosti zachovat v případě nasazení ve velkých sítích.
- **Flexibilita** – KMS protokol by měl být schopný fungovat v libovolném prostředí a poradit si i s dynamickým přidáním nových uzlů do sítě.

Základní omezení při návrhu a implementaci KMS protokolu vychází ze skutečnosti, že WSN uzly mohou být snadno ztraceny nebo zničeny. Cena WSN uzlů tedy musí být co nejnižší, což mimo jiné znamená, že uzly nemohou být zcela zapečetěné a odolné vůči fyzické manipulaci (angl. *temper-proof*). Další limitace spočívají v omezené kapacitě napájecího zdroje, dosahu signálu, rychlosti přenosu, dostupuné paměti a výpočetní síly [29].

- **Kapacita napájecího zdroje** – uzly WSN nejsou standardně připojeny k trvalému napájení. Místo toho jsou napájeny z baterie nebo akumulátoru. Kapacita takových zdrojů je značně omezená, a proto je vhodné namísto asymetrické kryptografie používat symetrickou kryptografii, která je energeticky úspornější a méně náročná na výpočetní výkon.
- **Dosah signálu** – omezená kapacita napájecího zdroje zároveň přímo ovlivňuje dosah vysílaného signálu. Uzly WSN nemohou používat výkonné všesměrové vysílání a místo toho jsou omezeny na vysílání na krátkou vzdálenost.
- **Rychlost přenosu** – uzly WSN mají z výše uvedených důvodů také omezenou přenosovou rychlost. Přenos velkých bloků dat tedy není efektivní. KMS protokoly by tedy měly minimalizovat množství dat přenášených za účelem vytvoření relace.
- **Dostupná paměť** – velikost dostupné paměti RAM a ROM je u většiny uzlů WSN v řádech jednotek nebo desítek, výjimečně stovek kB, z čehož je část dedikována operačnímu systému.

Efektivní KMS protokol musí mimo výše uvedených náležitostí brát v potaz i možnost útoku na WSN a jednotlivé uzly. Mezi hlavní požadavky na KMS protokol v tomto směru jsou [29]:

- **Odolnost vůči podstrčeným uzlům** – útočník může zaútočit na síť kompromitováním několika uzlů, které zreplikuje a nasadí zpět do sítě. Bezpečný KMS protokol by měl implementovat techniky, kterými se může bránit proti tomuto typu útoku.
- **Zneplatnění kompromitovaných uzlů** – KMS protokol by měl mít prostředky pro efektivní zneplatnění napadených uzlů.
- **Odolnost sítě vůči kompromitovaným uzlům** – pokud je uzel kompromitován, KMS protokol by měl zajistit, že nebude ohrožena bezpečnost komunikace mezi nekompromitovanými uzly.

2.6.1 Jeden klíč pro celou síť

Použití jednoho stejného klíče v každém uzlu WSN je ten nejjednodušší přístup. Před vlastním nasazením uzlů je do každého nahrán předem vygenerovaný klíč. Po nasazení je tento klíč použit k šifrování a dešifrování komunikace mezi uzly. Mezi výhody tohoto řešení patří minimální paměťová náročnost a jednoduchost požadovaného protokolu. Hlavní nevýhodou je, že prozrazení klíče kompromituje komunikaci mezi všemi uzly v síti.

2.6.2 Párové klíče

Přístup využívající párových klíčů je jeden z velmi efektvních pro WSN, jelikož nabízí několik implicitních výhod, mezi které patří schopnost komunikace a autentizace libovolných dvou uzlů v sítí a odolnost vůči replikaci uzlů. V tomto schématu má každý uzel vygenerován unikátní klíč pro komunikaci s libovolným jiným uzlem v síti. Má-li síť n uzlů, musí být v každém z nich uloženo $n-1$ klíčů. Toto schéma není dobře škálovatelné ani flexibilní. Díky jeho odolnosti vůči různým útokům však může být efektivně využito v menších sítích.

2.6.3 Důvěryhodná centrální stanice

V tomto schématu figuruje důvěryhodná stanice s vyšším výpočetním výkonem, jež není omezená kapacitou paměti nebo kapacitou zdroje. Každý uzel vlastní unikátní klíč sdílený s touto stanicí, která se chová jako důvěryhodná třetí strana (KDC – *Key Distribution Center*). Každý uzel se *autentizuje* pouze centrální stanici. Centrální stanice na základě *autentizace* vygeneruje relační klíč a bezpečně ho pošle oběma komunikujícím uzlům. Příkladem KMS protokolu založeném na KDC je protokol SPINS [17]. Toto schéma poskytuje obranu proti útokům na uzly a umožňuje zneplatnit kompromitované uzly. Zároveň představuje minimální paměťové a výpočetní nároky pro uzly sítě. Je však omezené dosahem stanice a jeho škálovatelnost je omezena výkonem stanice.

2.6.4 Náhodná distribuce klíče

Ve fázi generování klíčů je vygenerována velká množina klíčů a jejich identifikátorů S. Každému uzlu v síti je z této množiny náhodně přiřazeno m klíčů, které jsou uloženy do paměti uzlu. Počet klíčů v množině S a počet náhodně vybraných klíčů m je vybrán tak, aby dvě náhodné podmnožiny množiny S velikosti k sdílely alespoň jeden společný klíč s pravděpodobností p [8]. Pokud chtějí dva uzly získat relační klíč pro komunikaci mezi sebou, musí si vyměnit informaci o tom, jaké klíče sdílejí. Toho lze dosáhnout odesláním seznamu identifikátorů klíčů, jenž mají k dispozici, nebo hádankou, např. náhodné číslo α zašifrované různými klíči. Pouze uzel, který má k dispozici stejný klíč, může odpovědět zprávou s nešifrovaným α a tím dokázat, že sdílí stejný klíč. Na principu této metody jsou založena schémata *Q-composite random key predistribution scheme* nebo *Multipath key reinforcement scheme* [5].

2.6.5 Polynomiální metoda výpočtu klíče

Schéma založené na výpočtu společného klíče pomocí polynomů nabízí několik výhod oproti pravděpodobnostním schématům [29].

- Jakékoliv dva uzly sítě mohou vypočítat společný klíč, pokud žádné uzly sítě nejsou kompromitovány.

- Přestože je určitý počet uzlů sítě kompromitován, ostatní uzly stále mohou vypočítat tajný sdílený klíč.

Základní schéma popsané v [15] spočívá v náhodně vygenerovaném polynomu dvou proměnných $f(x, y)$ stupně t v konečném tělěsu GF(q), kde

$$f(x, y) = \sum_{i,j=0}^{t} a_{ij} x^i y^j.$$

Číslo q musí být prvočíslo dostatečně velké na to, aby šlo použít jako kryptografický klíč. Rovnice $f(x, y)$ má vlastnost $f(x, y) = f(y, x)$. Po vygenerování polynomu je pak polynom částečně vyřešen pro každý uzel v síti, např. pro uzel i je vyřešen polynom $f(i, y)$ a pro uzel j polynom $f(j, y)$. Pokud uzly i a j chtějí komunikovat, uzel i vypočítá polynom $f(i, j)$ a uzel j vypočítá polynom $f(j, i)$. Tento postup je bezpečný a neodhaluje žádnou komunikaci mezi ostatními uzly, dokud není kompromitováno t uzlů [15, 32].

Každý senzor musí mít v paměti uložený polynom, jenž zabírá $(t+1)\log(q)$ místa. Zvětšení velikosti sítě zvyšuje šanci na kompromitaci více než t uzlů. Z toho důvodu se zavádí modifikace spočívající v použití setu několika polynomů namísto jednoho. Z tohoto setu je ve fázi inicializace každému uzlu přidělena náhodná podmnožina polynomů a jejich identifikátorů. Zjištění, zda dva uzly sdílejí stejný polynom a tedy jsou schopné vypočítat sdílený klíč, může být statické nebo dynamické. Do paměti uzlu může být ve fázi inicializace uložen seznam identifikátorů uzlů, které sdílejí některý polynom ze skupiny polynomů přidělených uzlu. Tento přístup však znemožňuje možnost přidat do sítě další uzly a zjednodušuje útočníkovi jeho práci při napadení sítě, protože odhalený seznam uzlů sdílejících polynom mu může pomoci k nasměrování útoku na správné uzly. Dynamický přístup je např. broadcast vysílání seznamu identifikátorů polynomů, které uzel zná. Tento přístup však zvyšuje zátěž sítě, protože musí být mezi uzly přeneseno relativně velké množství dat.

Na této základní metodě (*Polynomial Based Pair-wise Key Pre-Distribution*) je založeno několik schémat využívajících například znalost polohy uzlů v síti (*Location-based Pair-wise Keys Scheme Using Bivariate Polynomials*, *Closes Polynomials Scheme*), případně upravujících způsob rozdělení generovaných polynomů (*Grid-based Key Pre-distribution*, *Hypercube-base Key Predistribution*) [15].

2.6.6 Maticová metoda výpočtu klíče

Tato metoda využívá vlastností matic G, tzv. *generátoru*, a matice A. Část dat matic G a A je pak uložena v uzlech a použita k výpočtu sdíleného párového klíče. Princip této metody je založen na schématu popsaného švédským kryptografem Rolfem Blomem v roce 1984 [3].

Oproti původnímu Blomem popsanému schématu se liší v několika detailech, základní princip je však stejný. Všechny matice jsou počítány nad konečným tělesem GF(p^q). Matice G typu $k \times N$ je složená z MRD (*Maximum Rank Distnace*) kódů [9], kde N je velikost sítě (počet uzlů) a $p^q > N$. Matice G je definována jako

$$G_k = \begin{pmatrix} g_1 & g_2 & \cdots & g_n \\ g_1^{[1]} & g_2^{[1]} & \cdots & g_n^{[1]} \\ \vdots & \vdots & \ddots & \vdots \\ g_1^{[k-1]} & g_2^{[k-1]} & \cdots & g_n^{[k-1]} \end{pmatrix} \tag{2.2}$$

kde $g_1, g_2, \ldots, g_n$ jsou lineárně nezávislé prvky z konečného tělesa $GF(p^q)$. Prvek $g^{[i]}$ značí Frobeniovu mocninu [9] prvku g [31]

$$g^{[i]} = g^{p^{(i \bmod N)}}. \tag{2.3}$$

Sloupce matice G jsou lineárně nezávislé a matice není tajná. Spolu s maticí G je vygenerována symetrická matice D typu $k \times k$, která je tajná. Konečně matice A typu $N \times k$ je vypočítána jako $A = (D \cdot G)^T$. Matice K s párovými klíči je vypočtena jako $K = (A \cdot G)$. Protože D je symetrická matice, K musí být také symetrická matice. Každému uzlu sítě je náhodně přidělen jeden řádek matice A. Pokud spolu chtějí komunikovat dva uzly, např. uzel S_i a uzel S_j, mohou vypočítat společný klíč z přiděleného řádku matice A a veřejné matice G. Předpokládejme, že uzlu S_i byl přidělen i-tý řádek matice A a uzlu S_j byl přidělen j-tý řádek matice A. Uzly si vymění informaci o tom, kolikátý řádek matice A vlastní a provedou skalární součin vlastního řádku matice A a odpovídajícího sloupce veřejné matice G. Uzel S_i vypočítá klíč

$$K_{ij} = \sum_{m=0}^{k} a_{im} g_{mj},$$

uzel S_j vypočítá klíč

$$K_{ji} = \sum_{m=0}^{k} a_{jm} g_{mi}.$$

Protože klíče K_{ij} a K_{ji} jsou prvky symetrické matice K, musí platit, že $K_{ij} = K_{ji}$ [31].

Metoda popsaná v [31] navíc rozděluje N uzlů sítě do t skupin s $k-\lambda$ uzly, kde $\lambda \geq 1$. Pro každou skupinu je pak vygenerována matice G_i a vypočítána matice $A_i = (D \cdot G_i)^T, i = \{1, 2, \ldots, t\}$. Pro výpočet všech prvků matice G_i je potřeba znát pouze první řádek, který bude značen g_i. První řádek matice $G_i + 1$ je vypočten vynásobením náhodného čísla β_i a jednoho již známeho řádku náhodně vybrané matice G, tedy

$$\begin{aligned} g_2 &= \beta_1 g_1 \\ g_3 &= \beta_2 \{g_1, g_2\} \\ &\vdots \\ g_t &= \beta_{t-1} \{g_1, g_2, \cdots, g_{t-1}\} \end{aligned}$$

Každému uzlu v i-té skupině, kde $i = 1, 2, \ldots, t$ je přidělen náhodný řádek matice A_i a první prvek odpovídajícího sloupce matice G_i. Pokud chtějí dva uzly spolu komunikovat, vymění si prvky z matice G a získají sloupec vypočtením zbylých elementů podle rovnic 2.2 a 2.3.

Důvod k rozdělení uzlů sítě do t skupin je, že původní Blomovo schéma je pouze k bezpečné. Pokud je kompromitováno alespoň k uzlů sítě, kompromituje se celá síť, neboť útočník může se znalostí k řádek matice A sestrojit vlastní matici A_{adv}. Matice G je veřejná a útočník získá celou množinu klíčů $K_{\text{adv}} = (A_{\text{adv}} \cdot G)$. Aby útočník odhalil matici A vytvořenou podle schématu [31], musel by odhlalit k uzlů ze stejné skupiny. Každá skupina však má pouze $k - \lambda$ uzlů.

3 Realizační část

Součástí diplomové práce je implementace vybraných algoritmů pro mikrokontrolér (MCU – *Microcontroller Unit*) *SimpleLink Wi-Fi CC3200*[1] společnosti *Texas Instruments*. V následujících kapitolách je popsáno, jak zprovoznit vývojové prostředí pro tento mikrokontrolér a jak přeložit a spustit dodané zdrojové kódy, jenž implementují vybrané algoritmy. Zdrojové kódy jsou v této kapitole detailně popsány a vysvětleny a je srovnána časová náročnost výpočtů, jenž algoritmy provádí.

3.1 Použitý mikrokontrolér

Programy, jenž jsou součástí této práce a které demonstrují vybrané algoritmy vytvoření relačního klíče, jsou určeny pro mikrokontrolér *SimpleLink Wi-Fi CC3200*. Vývoj probíhal na vývojové desce *CC3200-LAUNCHXL*, která spolu s čipem *CC3200* obsahuje USB emulaci JTAG rozhraní od společnosti FTDI, debuger součásti, UART rozhraní aj. Vývojová deska je k zakoupení u společnosti *Texas Instruments* v přepočtu za asi 730 Kč. Vlastní čipy *CC3200* jsou pak při odběru nad 1000 kusů v přepočtu za cenu kolem 122 Kč[2]. Základní specifikace čipu *CC3200*:

- Dvoujádrový procesor ARM® Cortex®-M4 s integrovaným síťovým procesorem pro Wi-Fi.
- 256 kB paměti RAM, z toho 16 kB je alokováno pro *bootloader*.
- 1 MB sériové flash paměti (např. pro SSL certifikáty).
- Kryptografický koprocesor pro akceleraci blokové šifry AES s délkou klíče 128, 192 nebo 256 bitů.
- Kryptografický koprocesor pro akceleraci hašovacích funkcí MD5, SHA-1 a SHA-2 včetně HMAC funkcí s velikostí klíče až 64 bytů.
- Kryptografický koprocesor pro akceleraci RSA algoritmů, včetně hardwarového generátoru náhodných čísel. Tyto části však nejsou přístupné

[1] http://www.ti.com/lsds/ti/wireless-connectivity/wi-fi/simplelink-wi-fi-cc31xx-cc32xx/overview.page

[2] Eshop *TI Store* (https://store.ti.com/) ke dni 7.5.2017

uživateli a využívá je pouze knihovna *SimpleLink* pro akceleraci SSL a TLS protokolu.

Pro správné fungování vývojové desky je nutná instalace balíku *CC3200 SDK*[3] (*Software Development Kit*), který obsahuje potřebné ovladače, knihovny, konfigurační soubory a zdrojové kódy ukázkových aplikací. Rozložení vývojové desky viz obrázek 3.1. Nejdůležitější komponenty jsou:

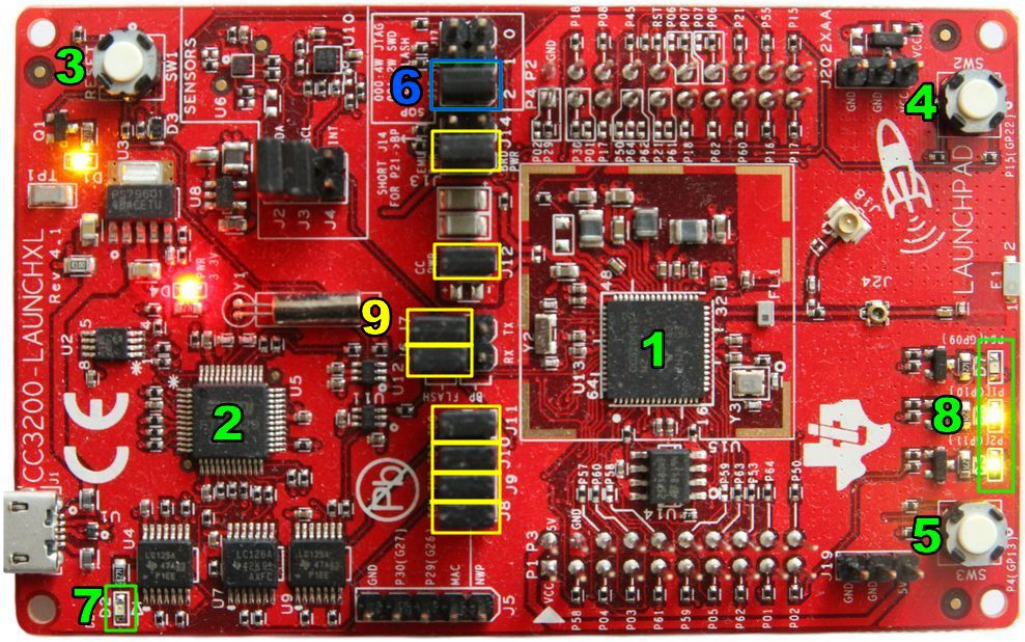
1. Čip MCU *CC3200*.
2. Čip FTDI JTAG emulátoru.
3. Tlačítko pro reset desky.
4. Tlačítko *SW2*.
5. Tlačítko *SW3*.
6. *SOP2* propojka (angl. *jumper*), modře.
7. Dioda signalizující debug mód.
8. Signalizační diody.
9. Základní zapojení propojek pro vývoj a debugování v *CCSv6*, žlutě.

3.1.1 Vývojové prostředí

Softwarový balík *CC3200 SDK* podporuje několik různých nástrojů pro vývoj. Mezi tyto nástroje patří:

- **Code Composer Studio** – software vyvíjený společností *Texas Instruments*. Je postaven na vývojovém prostředí Eclipse. Nabízí komplexní nástroje pro debugování, podporuje všechny ukázkové programy pro *CC3200* a po registraci na stránkách *Texas Instruments* je tento nástroj volně ke stažení.
- **IAR Workbench** – software vyvíjený společností *IAR Systems*. Přehledné a stabilní IDE podporující více než jedenáct tisíc zařízení. Volně stažitelná verze má však časově omezenou licenci na 30 dní.

[3]Dostupný na adrese http://www.ti.com/tool/cc3200sdk



Obrázek 3.1: Rozložení vývojové desky *SimpleLink Wi-Fi CC3200-LAUNCHXL*.

- **GNU GCC** – zcela otevřený zdrojový kód, nabízí však pouze kompilační nástroj pro příkazovou řádku. Možnosti debugování jsou omezené a chybí jakékoliv GUI.
- **Energia** – open source projekt založený Robertem Wesselsem, odvozený z Wiring IDE a Arduino IDE. Podpora a komunita je menší než u **Code Composer Studia**.

Pouze nástroj **Code Composer Studio**, dále *CCSv6*, je bez licenčních poplatků, současně stabilní a nabízí komplexní ladící nástroje v rámci dodaného vývojového prostředí (IDE – *Integrated Development Environment*). Projekty, jenž jsou součástí této práce, byly vyvíjeny v *CCSv6* a v dalším textu se předpokládá použití tohoto vývojového prostředí.

Postup pro instalaci *CCSv6* popsaný v následujících odstavcích je zkrácený a zjednodušený. Pro úplný a oficiální návod viz uživatelskou příručku *Getting Started Guide*[4]. Předpokládá se použití operačního systému Windows a instalace balíku SDK do výchozího umístění, tedy `C:/TI`. Programový balík

[4] http://www.ti.com/lit/ug/swru376d/swru376d.pdf

SDK i *CCSv6* jsou k dispozici i pro operační systémy Mac OSX a Linux. Tyto verze však nebyly v rozsahu této práce testovány.

Konfigurace vývojové desky

Zkontrolujte, že propojky jsou na desce zapojeny tak, jako na obrázku 3.1 (žlutě a modře označené). Připojte vývojovou desku k počítači pomocí dodaného USB kabelu. Ve správci zařízení se vývojová deska zobrazí jako položka v sekci *Porty (COM a LPT)* s názvem *CC3200LP Dual Port (COM 3)*. Číslo portu COM se může lišit a je nutné ho znát pro např. zobrazení UART terminálu nebo programování sériové flash paměti.

Instalace a konfigurace CCSv6

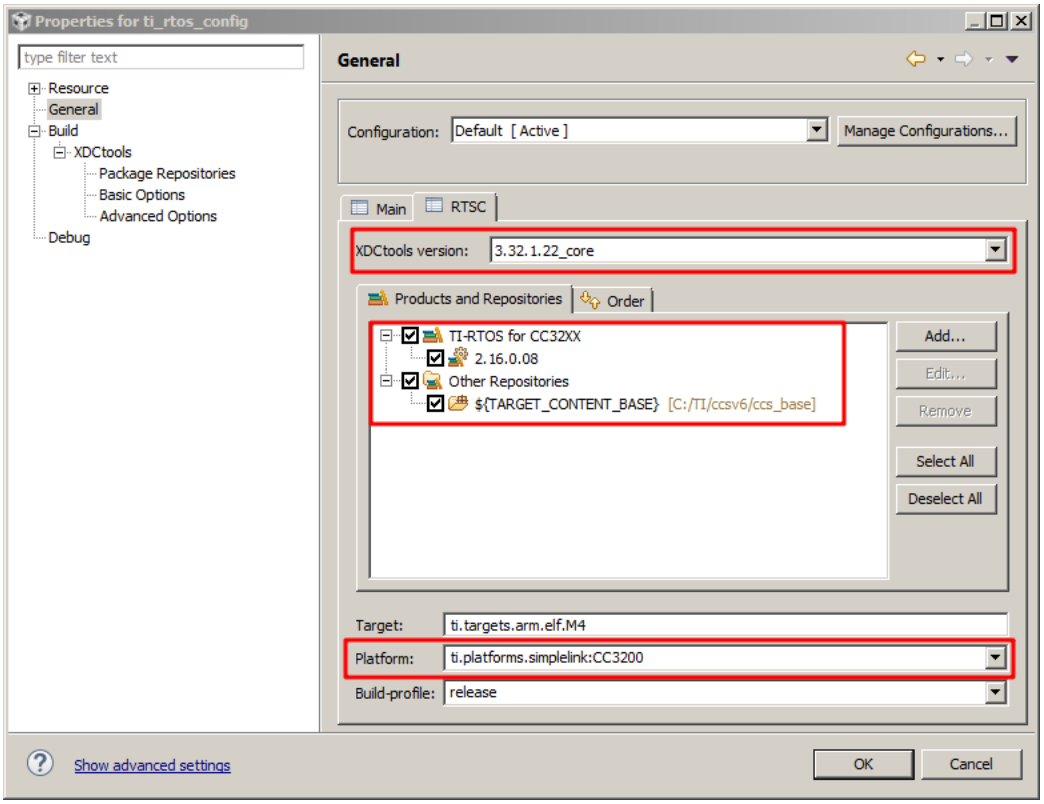
Nainstalujte *CCSv6* IDE. Verze programu musí být 6.1.1 nebo vyšší. Při instalaci budete dotázáni, jaké procesory mají být v instalaci podporovány. Vyberte položky *SimpleLink Wireless MCUs* → *CC32xx Device Support* a *SimpleLink Wireless MCUs* → *TI ARM Compiler*.

Naimportujte projekty `driverlib`, `simplelink`, `oslib` a `ti_rtos_config` pomocí dialogového okna *Project* → *Import CCS Projects* ze složky SDK, jejíž výchozí umístění je `C:/TI/CCC3200SDK/_1.2.0/cc3200-sdk`. Nakonfigurujte projekt `ti_rtos_config` v nabídce *Project* → *Properties* → *General* → *RTSC* podle obrázku 3.2. Nyní přeložte importované projekty (pravé tlačítko myši na projekt → *Build Project*).

Poslední krok je import konfiguračního souboru *Target Configuration* v nabídce *View* → *Target Configurations*. Pravým tlačítkem myši na položku *User Defined* vyvolejte kontextové menu, naimportujte soubor `C:/TI/CC3200SDK_1.2.0/cc3200-sdk/tools/ccs_patch/CC3200.ccxml` a nastavte ho jako výchozí.

3.1.2 Import projektu, překlad a spuštění v CCSv6

Práce obsahuje tři *CCSv6* projekty, jenž implementují tři různé způsoby vytvoření relačního klíče: pomocí ECC, polynomiální a maticové metody. Import projektu probíhá skrze menu *Project* → *Import CCS Projects...*.



Obrázek 3.2: Konfigurace projektu `ti_rtos_config`.

V otevřeném okně zvolíme soubor `zip` s vybraným projektem a potvrdíme kliknutím na talčítko *Finish*. Není-li SDK instalováno ve výchozím umístění, je potřeba upravit proměnné prostředí importovaného projektu v nabídce *Project* → *Properties* → *Resource* → *Linked Resources*. Úspěšný import projektu a správnou konfiguraci ověříme přeložením programu kliknutím pravým tlačítkem myši na projekt → *Build Project*.

Programy jsou určeny pro běh na dvou zařízeních zároveň a je tedy potřeba obě zařízení rozlišit. Byla zvolena nejjednodušší metoda, tedy čtení MAC adresy, která se v případě poskytnutých zařízení liší v posledním bytu. Jednotlivá zařízení jsou pak pojmenována jako *Alice* a *Bob*. Program nemá žádný externí konfigurační soubory a veškeré parametry je nutné nastavit ve zdrojových souborech. Všechny konfigurační parametry se nachází v souboru `config.h`, který je bohatě komentovaný. Nejdůležitější konfigurační parametry jsou společné pro všechny projekty:

- `_SSID_NAME` – Název sítě, ke které se budou zařízení připojovat.

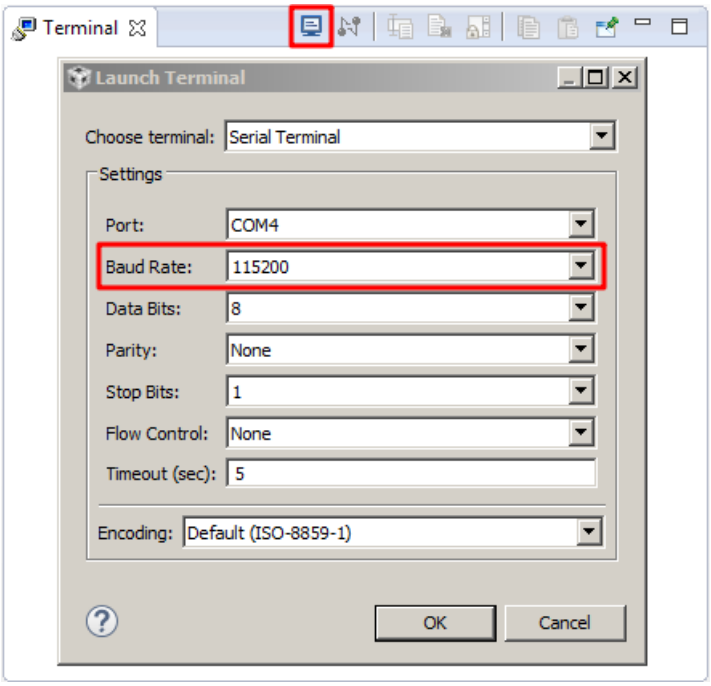
- `_SECURITY_TYPE` – Typ zabezpečení sítě (`SL_SEC_TYPE_OPEN/WEP/WPA`).
- `_SECURITY_KEY` – Heslo sítě, pokud je zabezpečená.
- `_GW_IP_ADDRESS` – IP adresa výchozí brány.
- `_NET_MASK` – Maska sítě.

Alias jednotlivých zařízení, poslední byte MAC adresy a staticky přidělená IP adresa jsou definovány v souboru `config.c` jako první, druhá a třetí položka struktury `device`. Tyto tři položky struktury `device` jsou stejné ve všech projektech.

Po úpravě konfiguračních parametrů programu můžeme projekt přeložit a spustit v debug módu na připojeném zařízení kliknutím pravým tlačítkem myši na projekt → *Debug As* → *Code Composer Debug Session*. Zařízení indikuje program běžící v debug módu rozsvícením zelené LED diody označené na obrázku 3.1 číslem 7. Program provádí výpis do UART terminálu. Terminál lze zobrazit ve *View Terminal*: *Windows* → *Show View* → *Other* → *Terminal*. Terminál je otevřen kliknutím na levou ikonu nabídky pohledu, viz obrázek 3.3. Hodnota položky *Baud Rate* by měla být 115200. Příklad výpisu spuštěného programu viz ukázku kódu 3.1.

```
************************************************
   Matrix-based key predistribution and authentication of CC32XX nodes
   version: 1.0.0
************************************************

Host Driver Version: 1.0.1.11
Build Version 2.8.0.0.31.1.4.0.1.1.0.3.37
Device is configured in default state
Started SimpleLink Device: STA Mode
MAC address is 54:4A:16:2D:F2:0D
[WLAN EVENT] STA Connected to the AP: ssid_name , BSSID: xx:xx:xx:xx:xx:xx
[NETAPP EVENT] IP acquired by the device

Device has connected to ssid_name
Device IP Address is 192.168.0.6

Network initialized. Running other tasks now...
```

Ukázka kódu 3.1: Výpis spuštěného programu v UART terminálu.



Obrázek 3.3: Otevření a konfigurace UART terminálu.

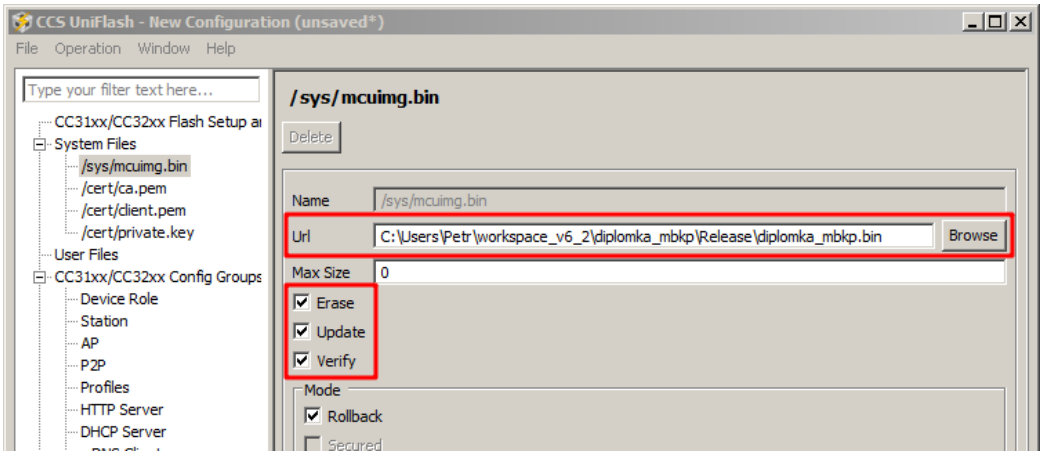
3.1.3 Spuštění programu bez debug režimu

Protože *CCSv6* může spustit v debug módu program pouze na jednom zařízení, je nutné na druhé zařízení nahrát přeložený program ručně nástrojem *CCS UniFlash*.

1. Připojte zařízení, na které bude přeložený program nahráván pomocí *CCS UniFlash*.
2. Zkontrolujte, že je odpojený UART terminál tohoto zařízení.
3. Zkontrolujte, že je zapojena *SOP2* propojka, označená na obrázku 3.1 modře číslem 6.
4. Stiskněte tlačítko pro reset vývojové desky, na obrázku 3.1 označeno číslem 3.
5. Otevřete program *CCS UniFlash* a vytvořte novou konfiguraci typu *CC3x Serial(UART) Interface*.
6. V nabídce *System Files* → */sys/mcuimg.bin* zvolte cestu k binárnímu souboru přeloženého programu a zaškrtněte položky *Erase*, *Update* a *Verify*, viz obrázek 3.4.

7. V nabídce *CC31xx/CC32xx Flash Setup and Control* nastavte správné číslo COM portu a klikněte na tlačítko *Program*. Tím je program nahrán do sériové flash paměti zařízení.

8. Vypojte *SOP2* propojku a restartujte zařízení tlačítkem reset.



Obrázek 3.4: Prostředí programu CCS UniFlash.

3.1.4 Spuštění na dvou zařízeních

Přestože oficiální příručka upozorňuje, že při spuštění debugeru musí být připojeno pouze jedno zařízení, je možné při dodržení následujícího postupu mít zařízení připojená obě zároveň a program na jednom opakovaně spouštět v debug módu a do druhého přeložený program opakovaně nahrávat programem *CCS UniFlash*.

1. Odpojte obě zařízení od počítače.

2. Připojte zařízení, které má být spouštěno v debug režimu. Ujistěte se, že je zapojena *SOP2* propojka, označená na obrázku 3.1 modře číslem 6.

3. Spusťte program na tomto zařízení v debug režimu pomocí *CCSv6*.

4. Připojte druhé zařízení, na které bude přeložený program nahráván pomocí *CCS UniFlash*.

5. Druhému zapojenému zařízení je přiděleno vyšší číslo COM portu. Dbejte na to, aby při restartování programu, jenž běží v debug režimu na prvním zařízení, byla vypojena *SOP2* propojka na druhém zařízení.

6. Nyní lze debugovat první zařízení v *CCS6* a programovat druhé zařízení nástrojem *CCS UniFlash*, aniž by musely být odpojovány od počítače.

3.2 Implementace algoritmů

Projekty jsou založeny na ukázkovém projektu *Email Demo*[5]. Programy jsou napsány pro běh na TI-RTOS, což je minimalistický operační systém reálného času s konfigurovatelnými moduly, který je vyvíjen společností *Texas Instruments* a který je určen pro MCU této společnosti. Z původního projektu je převzata inicializace síťového procesoru, TI-RTOS, vytváření vláken a obsluha přerušení. V kořenovém adresáři se nachází soubory:

- `main.c` – obsahuje hlavní programovou smyčku, volá funkce samotných modulů.
- `config.h` – konfigurační parametry a globální proměnné.
- `config.c` – definice některých globálních proměnných a jejich hodnot, např. klíče zařízení, vygenerované polynomy nebo matice atd.
- `session_context.h` – deklarace struktury `session_ctx` představující jednu relaci. Ve struktuře je uložen otevřený TCP soket a dále proměnné relace, které se liší podle použitého algoritmu.
- `session_context.c` – implementace funkcí pro inicializaci a uvolnění proměnné struktury `session_ctx`.

Zdrojové kódy vlastních algoritmů a pomocných funkcí jsou členěny podle svého účelu do tzv. modulů a nachází se ve složce `modules`:

- `my_utils.h`, `.c` – modul s pomocnými funkcemi pro vyplnění bufferu náhodnými čísly, vypsání bufferu na UART terminál v hexadecimálním formátu aj.
- `network.h`, `.c` – obsahuje funkce pro inicializaci síťového procesoru a pro práci s TCP sokety.
- `sha1.h`, `.c` – modul usnadňující použití hašovacího kryptoprocesoru.
- `authentication.h`, `.c` – implementace funkcí pro ověření uzlů. Vždy dvě funkce:

[5] http://processors.wiki.ti.com/index.php/CC32xx_Email_Demo_Application

  - `authenticate_client_side(session_ctx *ctx);` – funkce volaná klientem. Po skončení funkce je klient ověřen vůči serveru.
  - `authenticate_server_side(session_ctx *ctx);` – funkce volaná serverem. Po skončení funkce je klient ověřen vůči serveru.

  Server se může ověřit vůči klientovi voláním funkce určené pro ověření klienta. Klient s touto skutečností musí počítat a zavolat funkci se serverovou částí.

- `session.h`, `.c` – modul implementuje funkci pro vytvoření relačního klíče na základě úspěšné autentizace zařízení.

Pro každý projekt je dále k dispozici program napsaný v jazyce C++, který slouží ke generování klíčů, polynomů a matic, tedy dat, které mají být v uzlech uloženy před jejich nasazením. Tyto programy byly vyvinuty ve *Visual Studiu 2017*. Jedná se o jednoduché konzolové aplikace skládající se pouze ze souboru `main.c`, případně souborů knihovny *mbed TLS*, bez možnosti uživatelského vstupu. Různé vstupní parametry pro generování, jako např. typ eliptické křivky, stupeň polynomu nebo velikost matice Q, jsou definovány na začátku souboru `main.c`.

Každý z implementovaných algoritmů vyžaduje ke svému fungování generátor náhodných čísel. Přestože MCU *CC3200* implementuje hardwarový generátor náhodných čísel, není tento generátor dostupný skrze API a program k němu nemá přímý přístup. Implementace algoritmů v této práci tedy využívají pseudonáhodný generátor `rand()` knihovny `<stdlib.h>`. Je vhodné na tomto místě zdůraznit, že tento generátor je krajně nevhodný pro použití v kryptografických algoritmech, avšak pro potřeby této práce, která si dává za cíl pouze demonstrovat princip vybraných kryptografických algoritmů, postačuje.

3.2.1 Společný kód

Základní schéma běhu programu je stejné ve všech třech projektech:

1. Funkce `main` inicializuje MCU nastavením vektoru přerušení a spuštěním procesoru. Dále konfiguruje GPIO vstupy a výstupy, které jsou při běhu programu použity pro měření doby trvání výpočtů. Následně je inicializován TI-RTOS a spuštěna hlavní úloha `MainTask()`.

2. Hlavní úloha `MainTask()` volá funkci `InitNetwork()`, která inicializuje síťový procesor. Dále na základě MAC adresy zjistí, běží-li program na zařízení *Alice* nebo *Bob* a připojí zařízení k nakonfigurovanému přístupovému bodu.

3. Jakmile je zařízení připojeno k síti, spustí se úloha `ServerRoutine()`. Tato úloha otevře TCP soket na konfigurovaném portu a ve smyčce čeká na připojení klienta. Jakmile je klient připojen, volají se postupně funkce pro autentizaci klienta vůči serveru, autentizaci serveru vůči klientovi a vytvoření relačního klíče.

4. Také je spuštěna úloha `PushButtonHandler()`. Tato úloha registruje stisk tlačítka *SW2*, na jehož základě volá funkci `ButtonRoutine()`, což je klientský ekvivalent úlohy `ServerRoutine()`.

V bodě 2 výše uvedeného seznamu se provádí identifikace zařízení na základě jeho MAC adresy. Údaje unikátní pro jednotlivá zařízení *Alice* a *Bob* jsou uloženy ve struktuře `device`, která je deklarována v souboru `config.h`. Identifikátor zařízení, IP adresa a MAC adresa jsou ve struktuře uloženy vždy. Další parametry jsou specifické podle toho, který algoritmus implementace obsahuje. U *ECC* jsou to např. tajný a soukromý klíč uzlu, pro polynomiální metodu je to částečně vyřešený polynom $f(i, y)$ atp. V souboru `config.c` jsou definovány proměnné `Alice` a `Bob` typu `device` s vyplněnými parametry. Rozhodnutí, zda je zařízení *Alice* nebo *Bob* je založeno na porovnání MAC adresy zařízení a MAC adresy uložené v proměnných *Alice* a *Bob*. Následně je nastaven ukazatel `device *me` na tu či onu proměnnou.

Konfigurace konkrétního zařízení by mohla být uložena v externím souboru, který by byl uložen v sériové flash paměti zařízení a který by se lišil pro každé zařízení. Po startu programu by byl načten obsah souboru a hodnoty proměnných struktury `device me` by byly nastaveny z tohoto souboru.

Po stisku tlačítka *SW2* se bude zařízení snažit připojit ke druhému zařízení. Pokud se mu připojení podaří, proběhne vzájemné ověření a vytvoří se relační klíč. Žádná další interakce s uživatelem není možná.

3.2.2 Eliptické křivky

Implementace ověření pomocí *ECC* využívá knihovnu *mbed TLS*[6]. Z této knihovny jsou využity následující moduly:

- `bignum` – modul pro práci s velkými čísly. Program používá křivku `secp192r1`[7], *ECC* algoritmy tedy pracují s čísly alespoň 192 bitů dlouhými. V dalším textu bude pro tato čísla použito označení `MPI` (*Multi-Precision Integer*).
- `ecp` – modul definuje bod na eliptické křivce a implementuje funkce pro aritmetiku těchto bodů. Také definuje vybrané pojmenované křivky.
- `ecdsa` – modul implementuje algoritmy pro podepisování a ověření podpisu, také definuje pár soukromého a veřejného klíče, v dalším textu označován jako `keypair`. V následujícím textu budou algoritmy pro podpis a ověření podpisu reprezentovány funkcemi:
  - PODEPIŠ(R, r, d) – R jsou data, která mají být podepsána, r je výsledný podpis a d je soukromý klíč podepisující strany.
  - OVĚŘ(R, r, Q) – R jsou data, jejichž podpis bude ověřován, r je podpis, který má být ověřen a Q je veřejný klíč podepisující strany.

Jak bylo vysvětleno v kapitole 2.5.3, je při použití asymetrické kryptografie nutná účast důvěryhodné třetí strany, ať už ve formě certifikační autority nebo ve formě sítě důvěry. V této implementaci byl zvolen ten nejjednodušší možný přístup, kde generující server hraje roli CA. Server před generováním klíčů pro uzly vygeneruje sám sobě dvojici soukromého a veřejného klíče (d_{CA}, Q_{CA}). Pro každý uzel X vygeneruje (d_X, Q_X) a veřejný klíč Q_X podepíše svým soukromým klíčem d_{CA}: PODEPIŠ(Q_X, S_X, d_{CA}). Podpis S_X je pak uložen v paměti uzlu společně s veřejným klíčem certifikační autority Q_{CA} a vlastním `keypair`.

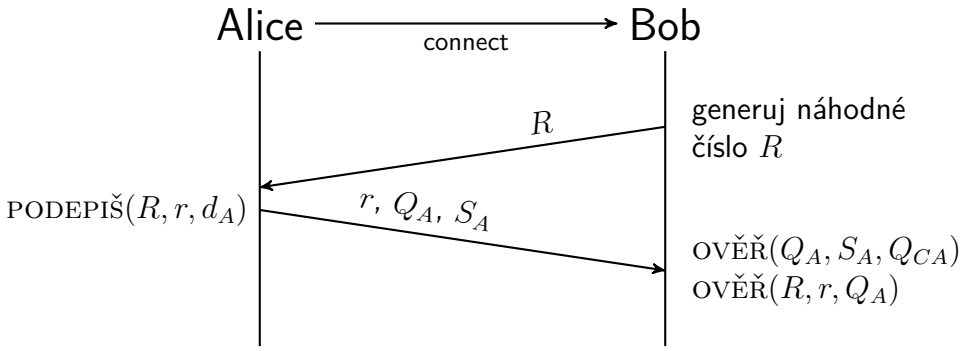
Aby se *Alice* autentizovala *Bobovi*, musí prokázat, že její `keypair` byl vygenerován certifikační autoritou. To provede tak, že odešle svůj veřejný klíč Q_A a jeho podpis S_A vytvořený certifikační autoritou. *Bob* ověří pomocí uloženého veřejného klíče Q_{CA}, že je podpis správný: OVĚŘ(Q_A, S_A, Q_{CA}). Tím je dokázano, že veřejný klíč Q_A byl vystaven certifikační autoritou. *Alice* dále musí prokázat, že vlastní k veřejnému klíči Q_A odpovídající soukromý

[6] https://tls.mbed.org/

[7] http://www.secg.org/sec2-v2.pdf

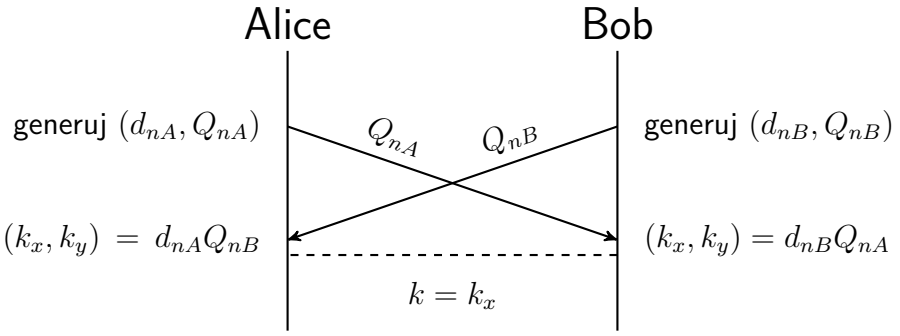
klíč d_A. *Bob* k tomuto účelu vygeneruje náhodné číslo R, které odešle *Alici*. *Alice* vytvoří pomocí svého soukromého klíče podpis PODEPIŠ(R, r, d_A), který odešle *Bobovi*. Ten pak musí ověřit, že podpis je validní s použitím již důvěryhodného veřejného klíče Q_A: OVĚŘ(R, r, Q_A). Diagram komunikace jednostranného ověření viz obrázek 3.5.

Oboustranného ověření je dosaženo opakovaným provedením tohoto postupu v opačném směru.



Obrázek 3.5: Diagram ověření klienta *Alice* vůči serveru *Bobovi* pomocí *ECC*.

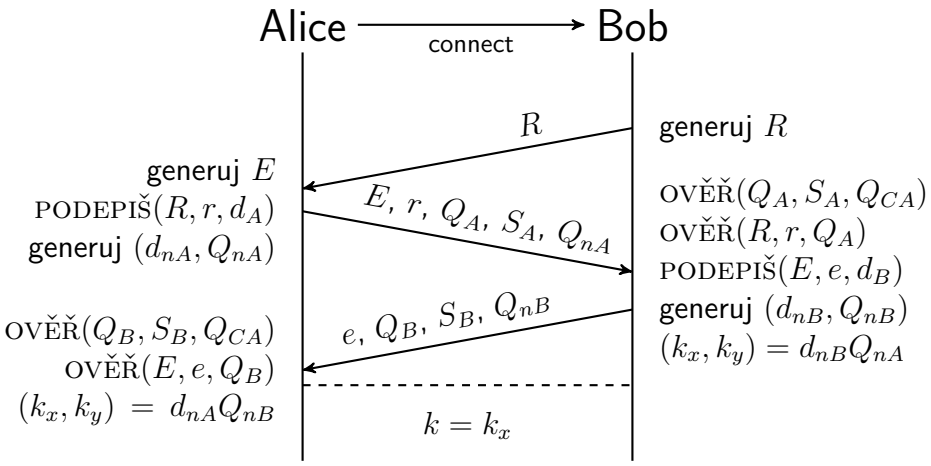
Relační klíč je po oboustranném ověření vytvořen algoritmem DHE. Obě strany vygenerují nový `keypair` (d_n, Q_n) a protistraně odešlou veřejný klíč Q_n. Obdržený veřejný klíč Q_n uzel vynásobí svým soukromým klíčem d_n. Výsledek tohoto násobení je bod na křivce, tedy hodnota x-ové a y-ové souřadnice, kde každá z těchto hodnot je 192 bitů dlouhé `MPI` číslo. Jako relační klíč pro AES šifrování je tedy možné použít 128 méně významných bitů hodnoty x-ové souřadnice. Diagram vytvoření relačního klíče viz obrázek 3.6.



Obrázek 3.6: Diagram vytvoření relačního klíče algoritmem DHE.

Implementace si dává za úkol pouze demonstrovat princip dané metody.

Pokud by se mělo jednat o reálně použitelný protokol, měly by být přítomny řídící zprávy potvrzující např. úspěšné či neúspěšné ověření, zda se bude ověřovat jednostranně nebo oboustranně atd. Algoritmus vytvoření relačního klíče není zabezpečen proti útoku MITM. Veřejný klíč Q_n by měl být alespoň podepsán soukromým klíčem uzlu, který byl ověřen při autentizaci, aby nemohl být nahrazen veřejným klíčem útočníka. Dále se nabízí optimalizace v podobě minimalizace počtu odeslaných zpráv, kde místo šesti zpráv (čtyři pro oboustranné ověření a dvě pro vypočtení relačního klíče) by celý proces mohl být proveden pomocí tří zpráv, viz obrázek 3.7.



Obrázek 3.7: Diagram s optimalizovaným počtem zpráv pro oboustranné ověření a výpočet relačního klíče.

Poznámky ke zdrojovému kódu

Klíče uzlu, podpis veřejného klíče uzlu a veřejný klíč CA jsou deklarovány ve struktuře `device`:

```
typedef struct
{
  int  id;             // alice nebo bob
  _u8  mac_byte;       // poslední byte MAC adresy
  _u32 ip;             // statická IP adresa
  _u8  d_bin[24];      // soukromý klíč
  _u8  Q_bin[49];      // veřejný klíč
  _u8  Q_ca_bin[49];   // veřejný klíč CA
  _u8  Q_sign_bin[55]; // veřejný klíč uzlu podepsaný CA
} device;
```

Data jsou v proměnných uložena binárně, tedy např:

```
_u8 d_bin[24] = {0x35,0x2a,0xea,0xdd,0xf3,0x05,0x31,...};
```

Příklad načtení binárního bufferu klíče do proměnné typu `mbedtls_ecp_keypair`, výpočet podpisu a ověření podpisu viz ukázky kódu 3.2, 3.3 a 3.4.

```
const mbedtls_ecp_group_id GROUP_ID = MBEDTLS_ECP_DP_SECP192R1;  // použitá křivka
_u8 d_bin[24] = {0x35,0x2a,0xea,0xdd,0xf3,0x05,0x31,...};        // soukromý klíč
_u8 Q_bin[49] = {0x04,0xaa,0x1c,0x84,0x2e,0x3d,0xc8,...};        // veřejný klíč

// mbedtls proměnná představuje pár sourkomého a veřejného klíče
mbedtls_ecp_keypair keypair;

// inicializace proměnné a načtení typu křivky
mbedtls_ecp_keypair_init(&keypair);
mbedtls_ecp_group_load(&keypair.grp, GROUP_ID);

// přečti data soukromého a veřejného klíče do keyipair proměnné
mbedtls_mpi_read_binary(&keypair.d, d_bin, sizeof(d_bin));
mbedtls_ecp_point_read_binary(&keypair.grp, &keypair.Q, Q_bin, sizeof(Q_bin));
```

Ukázka kódu 3.2: Kód pro načtení binárních dat do proměnné typu `mbedtls_ecp_keypair`.

```
#define SHA1_DIGEST_SIZE 20
_u8 digest[SHA1_DIGEST_SIZE];      // buffer s SHA1 hašem dat, které budou podepsána
_u8 sign[MBEDTLS_ECDSA_MAX_LEN];   // buffer s podpisem
size_t sign_len;                   // délka podpisu

// gen_random_bytes je funkce, která vyplní zadaný buffer náhodnými čísly
// int gen_random_bytes(void *p_rng, unsigned char *output, size_t output_len);
// - p_rng - vždy NULL
// - output - ukazatel na začátek bufferu
// - output_len - velikost bufferu
mbedtls_ecdsa_write_signature(&keypair, MBEDTLS_MD_SHA1, digest, SHA1_DIGEST_SIZE, sign, &
    sign_len, &gen_random_bytes, NULL);
```

Ukázka kódu 3.3: Vytvoření podpisu knihovnou *mbed TLS*.

```
#define SHA1_DIGEST_SIZE 20
size_t sign_len = 52;              // délka podpisu
_u8 digest[SHA1_DIGEST_SIZE];      // buffer s SHA1 hašem dat, jejichž podpis má být ověřen
_u8 sign[sign_len];                // buffer s podpisem k ověření
int ok;

// návratová hodnota je 0 v případě, že podpis je validní
```

```
8 ok = mbedtls_ecdsa_read_signature(&keypair, digest, SHA1_DIGEST_SIZE, sign, sign_len);
```

Ukázka kódu 3.4: Ověření podpisu knihovnou *mbed TLS*.

3.2.3 Polynomiální metoda

Pro ověření musí uzel dokázat, že zná párový klíč, který získá dosazením identifikátoru druhého uzlu do částečně vyřešeného polynomu stupně t, který má uložen v paměti. Výsledným klíčem zašifruje výzvu, zde náhodné číslo R, čímž dokáže, že klíč zná.

Před vlastním nasazením uzlů server vygeneruje polynom $f(x, y)$ stupně t v konečném tělesu $\mathrm{GF}(q)$

$$f(x,y) = \sum_{i,j=0}^{t} a_{ij} x^i y^j. \tag{3.1}$$

Po rozepsání rovnice 3.1 do tvaru

$$\begin{aligned} f(x,y) = {} & a_{0,0}x^0y^0 + a_{0,1}x^0y^1 + a_{0,2}x^0y^2 + \cdots + a_{0,t}x^0y^t \\ & + a_{1,0}x^1y^0 + a_{1,1}x^1y^1 + a_{1,2}x^1y^2 + \cdots + a_{1,t}x^1y^t \\ & \vdots \\ & + a_{t,0}x^ty^0 + a_{t,1}x^ty^1 + a_{t,2}x^ty^2 + \cdots + a_{t,t}x^ty^t \end{aligned}$$

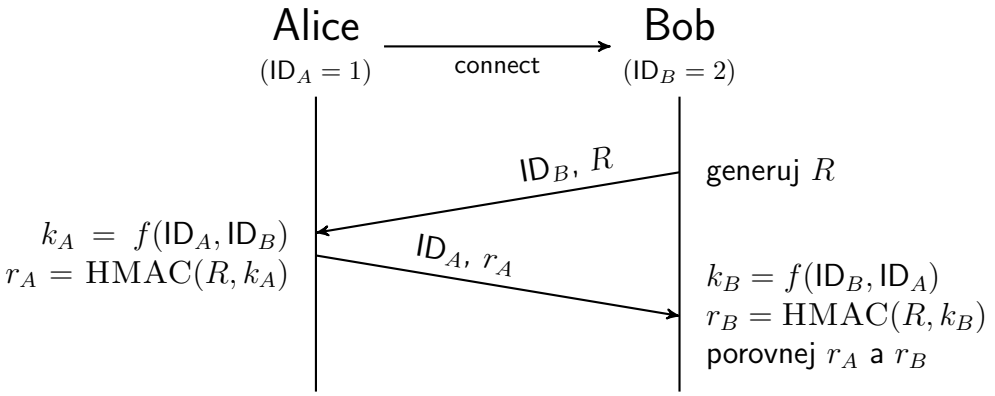
je zřejmé, že polynom lze reprezentovat maticí A typu $t \times t$, kde proměnné i a j z rovnice 3.1 odpovídají řádku a sloupci matice A. Aby polynom 3.1 splňoval podmínku, že $f(x, y) = f(y, x)$, musí být navíc matice A symetrická.

Každému uzlu sítě je následně přidělen identifikátor ID_x, který je dosazen do polynomu za proměnnou x. Polynom je částečně vypočten pro dosazenou proměnnou pro každý uzel a výsledek je uložen do paměti uzlu spolu s jeho identifikátorem. Po vytknutí činitelů y^j lze takový polynom zapsat jako

$$\begin{aligned} f(\mathrm{ID}_x, y) = {} & y^0(a_{0,0}\mathrm{ID}_x^0 + a_{1,0}\mathrm{ID}_x^1 + \cdots + a_{t,0}\mathrm{ID}_x^t) \\ & + y^1(a_{0,1}\mathrm{ID}_x^0 + a_{1,1}\mathrm{ID}_x^1 + \cdots + a_{t,1}\mathrm{ID}_x^t) \\ & \vdots \\ & + y^t(a_{0,t}\mathrm{ID}_x^0 + a_{1,t}\mathrm{ID}_x^1 + \cdots + a_{t,t}\mathrm{ID}_x^t). \end{aligned} \tag{3.2}$$

Z rovnice 3.2 vyplývá, že částečně vyřešený polynom zabere v paměti uzlu právě $(t + 1) \log_2 q$ bitů.

Má-li se *Alice* ověřit *Bobovi*, musí *Bob* odeslat *Alici* svůj identifikátor ID_B a náhodné číslo R, které bude sloužit jako výzva pro ověření znalosti párového klíče k. *Alice* vypočítá klíč k dosazením *Bobova* identifikátoru ID_B za proměnnou y do částečně vypočteného polynomu $f(\mathrm{ID}_A, y)$. Následně vypočítá haš r čísla R jako výsledek HMAC funkce $r = \mathrm{HMAC}(R, k)$. *Alice* odešle *Bobovi* jako odpověď vypočítaný haš r a svůj identifikátor ID_A, který *Bob* použije pro výpočet klíče k. Pomocí klíče k vytvoří svojí verzi haše r a tuto verzi porovná s verzí obdrženou od *Alice*. Pokud jsou oba haše stejné, je *Alice* ověřena. Diagram jednostranného ověření viz obrázek 3.8.



Obrázek 3.8: Diagram jednostranného ověření uzlu polynomiální metodou.

Podobně jako v předchozí kapitole 3.2.2 i zde je oboustranného ověření dosaženo opakováním uvedeného postupu v opačném směru. Tak jako *Bob* generoval náhodné číslo R a poslal ho *Alici* jako výzvu, *Alice* bude v opakovaném postupu generovat číslo E. Implementačně je zajištěno, že klíč k není počítán dvakrát a že identifikátory uzlů nejsou odlišné od identifikátorů obdržených v prvním ověření, přesto se zde nabízí prostor ke zlepšení v podobě oboustranného ověření pomocí pouze tří zpráv namísto čtyř.

Samotný relační klíč k_r je vypočítán jako jako výsledek HMAC funkce XOR operace čísel R a E, tedy $k_r = \mathrm{HMAC}(R \oplus E, k)$. Tento způsob vytvoření relačního klíče zabrání používání párového klíče k šifrování vlastní komunikace a omezí tak množství dat, která by útočník mohl použít ke kryptoanalýze. Neposkytuje bohužel dopřednou bezpečnost, jako např. algoritmus DHE.

Implementace této metody ověření využívá jednu důležitou optimalizaci. Ta je převzata z [15] a spočívá v rozdělení polynomu $f(x, y)$ v konečném

tělese GF(q) na několik polynomů $f_i'(x, y)$ v konečném tělese GF(q'), kde $q' < q$. Předpokládejme, že výsledek polynomu $f(x, y)$ představuje relační klíč, který má být použit k šifrování komunikace mezi dvěma uzly. Dále předpokládejme, že k šifrování bude použit algoritmus AES a minimální délka klíče tedy musí být alespoň 128 bitů. Koeficienty použitého polynomu musí tudíž být 128 bitová čísla, což nás nutí k použití algoritmů, jenž umí násobit, sčítat a provádět modulo operaci nad takto velkými čísly. Tyto algoritmy jsou samozřejmě pomalejší než standardní násobení nebo sčítání čísel velkých tak, že se vejdou do registrů procesoru. Zvolíme-li navíc q' tak, že $q' = 2^k + 1$, kde k je počet bitů, které podporuje procesor, není potřeba ani operace dělení nebo modulo [15]. V našem případě bylo zvoleno $k = 16$, tedy $q' = 2^{16} + 1 = 65537$. Máme-li k dispozici 8 různých polynomů v GF(q'), $q' = 2^{16} + 1$, můžeme je vypočítat stejným způsobem, jako bychom počítali jeden polynom v GF(q) a získáme osm 16-ti bitových podklíčů. Spojíme-li tyto podklíče za sebe, získáme jeden 128 bitový klíč, jenž je stejně bezpečný, jako klíč získaný neoptimalizovanou metodou [15].

Poznámky ke zdrojovému kódu

Částečně vyřešené polynomy jsou opět uloženy ve struktuře `device`, `R` je počet použitých polynomů, `T` je stupeň polynomu:

```
_u16 p[R][T] = {0x0C29,0x8B87,0x92F1,0x68CB,0xE2B6,0xE729,0x01B6,0x838A,0x0EDC,...};
```

Výpočet polynomů pak neprobíhá polynom po polynomu, ale exponent po exponentu. Díky tomu se y^j počítá jenom jednou a může být použito ve všech `R` polynomech, viz ukázku kódu 3.5.

```
// Vyřešené polynomy
_u16 p[R][T] = {0xE047,0x01F7,0x0622,0xA638,0xC963,...};
// Identifikátor protistrany
_u16 IDp = 2;

int poly, j;
_u16 exp = 1;
_u16 z[R];

// pro každý koeficient
for (j = 0; j < T; j++) {
  // v každém polynomu
  for (poly = 0; poly < R; poly++)
    // vynásob koeficient a y^j
```

```
15      z[poly] += p[poly][j] * exp;
16
17    // y^(j+1) = y^j y
18    exp *= IDp;
19  }
```

Ukázka kódu 3.5: Výpočet R polynomů.

3.2.4 Maticová metoda

Protokol ověření je téměř shodný s tím, který je použit v předchozí kapitole 3.2.3. Ověřovaný uzel musí prokázat, že zná párový klíč, kterým zašifruje výzvu, což je náhodné číslo R.

Před nasazením uzlů server vygeneruje matice G, D a A tak, jak je popsáno v kapitole 2.6.6. Matice G je typu $k \times N$, kde k určuje bezpečnost metody podobně, jako t určovalo stupeň polynomu a tedy bezpečnost polynomiální metody, a N je maximální počet uzlů v síti. Matice D je symetrická a typu $k \times k$. Matice A je výsledkem násobení matic $A = (D \cdot G)^T$ a je tedy typu $N \times k$.

Každému uzlu v síti je do paměti uložen jeden náhodný řádek matice A a první prvek odpovídajícího sloupce matice G. Je-li *Alici* přidělen i-tý řádek matice A, uloží se do paměti prvek $g_{1,i}$ matice G. Matice jsou v konečném tělese $\mathrm{GF}(q)$ a tedy údaje uložené v paměti uzlu zabírají $(k+1)\log_2 q$ bitů.

Pro ověření *Alice* vůči *Bobovi* musí *Bob* odeslat *Alici* svůj prvek g_B matice G spolu s náhodným číslem R. *Alice* z prvku g_B může vypočítat celý sloupec matice G podle rovnice 2.3. Párový klíč k pak *Alice* vypočítá skalárním násobením svého řádku matice A a *Bobova* sloupce matice G. Klíč k použije *Alice* k výpočtu haše r čísla R, $r = \mathrm{HMAC}(R, k)$, který *Bobovi* odešle spolu se svým prvkem g_A matice G. *Bob* podnikne stejné kroky pro výpočet svého párového klíče k, se kterým vytvoří vlastní verzi haše r a obě verze porovná. Jsou-li haše stejné, je *Alice* ověřena. Oboustranného ověření je dosaženo opakovaným voláním uvedené procedury v opačném směru, kde *Alice* pošle výzvu, náhodné číslo E, *Bobovi*.

Relační klíč k_r je vypočítán jako výsledek HMAC funkce XOR operace čísel R a E, tedy $k_r = \mathrm{HMAC}(R \oplus E, k)$. Tento způsob vytvoření relačního klíče zabrání používání párového klíče k šifrování vlastní komunikace

a omezí tak množství dat, která by útočník mohl použít ke kryptoanalýze. Neposkytuje bohužel dopřednou bezpečnost, jako např. algoritmus DHE.

Jediný rozdíl oproti protokolu polynomiální metody spočívá ve způsobu výpočtu párového klíče k a že si zařízení nevymění svoje identifikátory, ale prvek matice G. Z tohoto důvodu i zde existuje prostor pro zlepšení a to v podobě snížení počtu vyměněných zpráv.

Zároveň i zde je použita optimalizace z kapitoly 3.2.3. Namísto jedné matice Q typu $k \times N$ v konečném tělese GF(q) je vygenerováno několik matic Q_i typu $k \times N$ v konečném tělese GF(q'). Zvolíme-li $q' = 2^{16} + 1$, budeme generovat matice Q_i pro $i = \{1, 2, \ldots, 8\}$. Každou z těchto matic vynásobíme maticí D, také v konečném tělese GF(q'), a získáme matice $A_i = (D \cdot G_i)^T$. Matice K_i s 16 bitovými párovými klíči získáme násobením $K_i = (A_i \cdot G_i)$. Vynásobením řádku matice A_i a odpovídajícího sloupce matice Q'_i pro $i = \{1, 2, \ldots, 8\}$ tedy získáme osm 16-ti bitových klíčů. Tyto klíče pak spojíme za sebe a získáme jeden 128 bitový klíč, který je stejně bezpečný, jako bychom použili jednu matici Q v konečném tělése GF(q), kde $q = 2^{128} + 1$.

Poznámky ke zdrojovému kódu

Řádky matic A_i a odpovídající prvky matice G_i jsou uloženy ve struktuře `device`, `R` je počet použitých matic, `K` je zvolený stupeň bezpečnosti metody:

```
_u16 a[R][K] = {0xD139,0xD089,0x9850,0x0BC5,0x521B,0x57D7,0x2DB2,0xA35C,0xF091,...};
_u16 g[R] = {0x329F,0x0D8D,0x51DF,0xFB93,0xECDD,0xD7B3,0x010D,0x723B};
```

Násobení řádků matic A_i a obdržených sloupců matic G_i pak probíhá postupně pro každé i, viz ukázka kódu 3.6.

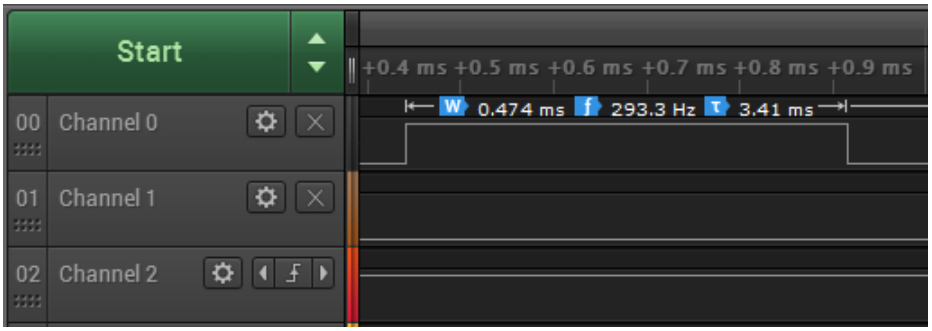
```
// řádky matic A_i
_u16 a[R][K] = {0xD139,0xD089,0x9850,0x0BC5,0x521B,0x57D7,0x2DB2,0xA35C,0xF091,...};
// prvky g_i protějšku
_u16 g[R] = {0x329F,0x0D8D,0x51DF,0xFB93,0xECDD,0xD7B3,0x010D,0x723B};

int i, j;
_u16 g, z[R];

// pro každý řádek matice A_i a prvku g_i
for (i = 0; i < R; i++) {
  g = g[i];
  // pro každý prvek v řádku matice A_i
  for (j = 0; j < K; j++) {
```

```
14      // vynásob a_i^j a g_i^[j]
15      z[i] += a[i][j] * g;
16
17      // g_i^[j+1] = g_i^[j] g_i^[j]
18      g *= g;
19    }
20  }
```

Ukázka kódu 3.6: Výpočet sloupců matice G_i a násobení s řádky matic A_i.

3.3 Měření a srovnání

Měření časové náročnosti probíhalo pomocí osmikanálového logického analyzátoru od společnosti *Saleae* s frekvencí snímání 500 kHz. Odchylka měření je tedy $\pm 2\,\mu s$. Piny analyzátoru byly připojeny na GPIO piny zařízení, na kterých byla programově měněna hrana z *low* na *high* a opačně. Naměřené vzorky byly zpracovány v nástroji *Saleae Logic*, viz obrázek 3.9. Z naměřených hodnot pak byl odečten čas mezi změnou z *low* na *high* a změnou z *high* na *low*.



Obrázek 3.9: Příklad měření v nástroji *Saleae Logic*.

Ke konfiguraci GPIO pinů byl použit nástroj *Pin Mux Tool*, který vygeneruje na základě uživatelského vstupu konfigurační soubor `pinmux.c`. Příklad konfigurace GPIO pinu 18 viz ukázku kódu 3.7.

```
1 // nastavení módu GPIO pinu
2 MAP_PinTypeGPIO(PIN_18, PIN_MODE_0, false);
3 // nastavení pinu jako výstupního
4 MAP_GPIODirModeSet(GPIOA3_BASE, 0x10, GPIO_DIR_MODE_OUT);
```

Ukázka kódu 3.7: Konfigurace GPIO pinu 18 v souboru `pinmux.c`.

Kód GPIO pinu není stejný jako jeho značení na desce. Správné číslo GPIO pinu je tedy potřeba odečíst z nástroje *Pin Mux Tool*. Příklad nastavení GPIO pinu 18 na hodnotu *high* viz ukázku kódu 3.8.

```
#define LOW 0
#define HIGH 1
#define PIN_18 28 // GPIO pin 18 má ve zdrojovém kódu hodnotu 28

unsigned int port;
unsigned char pin;

// získá číslo portu a pinu pro nastavení hladiny
GPIO_IF_GetPortNPin(gpio,&port,&pin);
// gpio pin nastaví na hodnotu high
GPIO_IF_Set(gpio,port,pin,HIGH);
```

Ukázka kódu 3.8: Nastavení GPIO pinu 18 na hodnotu *high*.

Měřen byl čas výpočtu jednotlivých algoritmů, ale také čas trvání celého procesu oboustranného ověření. Bylo provedeno mnoho měření, jejichž grafické výstupy nejsou v práci zahrnuty. V následujících odstavcích jsou uvedeny pouze samotné výsledky měření.

U metody využívající *ECC* byla měřena doba výpočtu algoritmů podpisu, ověření podpisu, generování páru klíčů (d, Q) a násobení soukromého a veřejného klíče. Bylo provedeno deset měření běhu ověření. Průměrný čas běhu algoritmů je zaznamenán v tabulce 3.1. Rozdíl v čase dvou různých ověření podpisu algoritmem ECDSA je pravděpodobně způsoben časem nutným k předpočítání údajů pro využití metody mocnění s klouzajícím okénkem, které používá knihovna *mbed TLS*.

Tabulka 3.1: Čas výpočtu ECC algoritmů na MCU *CC3200*.

Algoritmus	Průměrný čas [s]
Podpis čísla R	1,195
Ověření podpisu čísla R	7,341
Ověření podpisu veřejného klíče uzlu	4,926
Generování klíčů pro DHE	3,546
Násobení klíčů	3,591

Protože na sebe uzly v různých fázích protokolu musí čekat, trvá celý proces oboustranného ověření a vytvoření relačního klíče zhruba 34 sekund. To je naprosto nepřípustný čas pro uzel senzorické sítě a využití tohoto protokolu

by výrazně snížilo životnost uzlu. Implementace *TinyECC* z [14] dosahuje na 104 MHz procesoru mnohem lepších časů, což dokazuje důležitost optimalizace imlementace s přihlédnutím k použitému hardwaru. Srovnání časů dosažených implementací *TinyECC* na uzlu *Imote2* s procesorem na frekvenci 104 MHz a časů dosažených s knihovnou *mbed TLS* na uzlu *CC3200* obsahuje tabulka 3.2. Čas inicializace knihovny *mbed TLS* byl odhadnut na základě rozdílu mezi ověřováním podpisů v tabulce 3.1. Nutno podotknout, že *TinyECC* používá křivku `secp160`, kdežto s *mbed TLS* byla použita křivka `secp192`.

Tabulka 3.2: Srovnání časů výpočtů algoritmu ECDSA implementace *TinyECC* na uzlu *Imote2* na frekvenci 104 MHz [14] a knihovny *mbed TLS* na uzlu *CC3200*.

Algoritmus	*TinyECC* na *Imote2* @104 Mhz [ms]	*mbed TLS* na *CC3200* @80 MHz [ms]
Inicializace	43	2415
Vytvoření podpisu	45	1195
Ověření podpisu	56	4926

U polynomiální metody byl měřen čas výpočtu částečně vyřešeného polynomu po dosazení identifikátoru druhého uzlu. Celkový čas potřebný pro oboustranné ověření a vytvoření relačního klíče byl v průměru 800 ms. U maticové metody byl měřen čas výpočtu sloupců matic G_i z prvků g_i a násobení řádků matic A_i a vypočtených sloupců matic G_i. Časy pro různá t a k jsou uvedeny v tabulce 3.3. Celkový čas potřebný pro oboustranné ověření a vytvoření relačního klíče byl stejně jako v případě polynomiální metody v průměru 800 ms.

Tabulka 3.3: Čas výpočtu klíče pomocí polynomiální a maticové metody.

	Polynomiální metoda [µs]	Maticová metoda [µs]
$k = t = 10$	48	52
$k = t = 49$	226	238
$k = t = 99$	450	476

Rozdíl v času výpočtu polynomiální metody a maticové metody pro $k = t$ je způsobený přeuspořádáním operací při výpočtu polynomiální metody. Identifikátor uzlu v algoritmu polynomiální metody odpovídá prvku g_i v maticové

metodě. Rozdíl však je, že u polynomiální metody je mocnina identifikátoru stejná pro všechny dílčí polynomy, kdežto u maticové metody je nutné počítat mocninu prvku g_i pro každou dílčí matici zvlášť. Počet operací výpočtu párového klíče obou algoritmů lze zapsat jako:

$$\text{polynomiální metoda: } Rt(*) + Rt(+) + t(*),$$
$$\text{maticová metoda: } Rk(*) + Rk(+) + Rk(*).$$

Přesto jsou algoritmy výpočtu klíče obou metod rovnocenné, neboť asymptotická složitost výpočtu klíče u obou je $O(n)$. Paměťová náročnost je také srovnatelná. Pro polynomiální metodu musí být v paměti uloženo R částečně vyřešených polynomů a jeden identifikátor uzlu, tedy $(Rt+R+1)\log_2 q'$ bitů. Maticová metoda vyžaduje R řádek matic A_i a R prvků matic G_i, tedy $(Rk + R)\log_2 q'$ bitů. Rozdíl je tedy konstatní a to $\log_2 q'$ bitů ve prospěch maticové metody. Oba protokoly využívající zmíněné metody jsou si rovny i v počtu přenesených zpráv s tím rozdílem, že maticová metoda musí odeslat druhému uzlu R prvků matice G_i, kdežto polynomiální metoda odesílá jenom jedno stejně velké číslo, které představuje identifikátor uzlu. Celkové množství přenesených dat se tedy liší. Tabulka 3.4 uvádí konkrétní velikosti paměti, přenesených dat a času spotřebované oběma metodami pro $q' = 2^{16} + 1$, $R = 8$ a $k = t = 99$.

Při měření se také ukázalo, jak účinná může být hardwarová implementace kryptografického algoritmu, když generování dvaceti náhodných bytů funkcí `rand()` trvalo 118 µs, tedy více než čtyřikrát déle než hardwarově implementovaný kryptografický algoritmus HMAC se 128 bitovým klíčem, jehož výpočet trval pouze 26 µs.

Tabulka 3.4: Srovnání implementovaných protokolů využívajících symetrické klíče, TLS 1.2 s velikostí RSA klíče 2048 bitů a protokolu s použitím ECC s křivkou `secp192`.

	Polynomiální metoda	Maticová metoda	TLS 1.2	ECC
Paměť [B]	1602	1600	3020	177
Přenesená data [B]	88	144	2300	456
Počet zpráv	4	4	12	6
Čas běhu protokolu [ms]	800	800	1200	34000

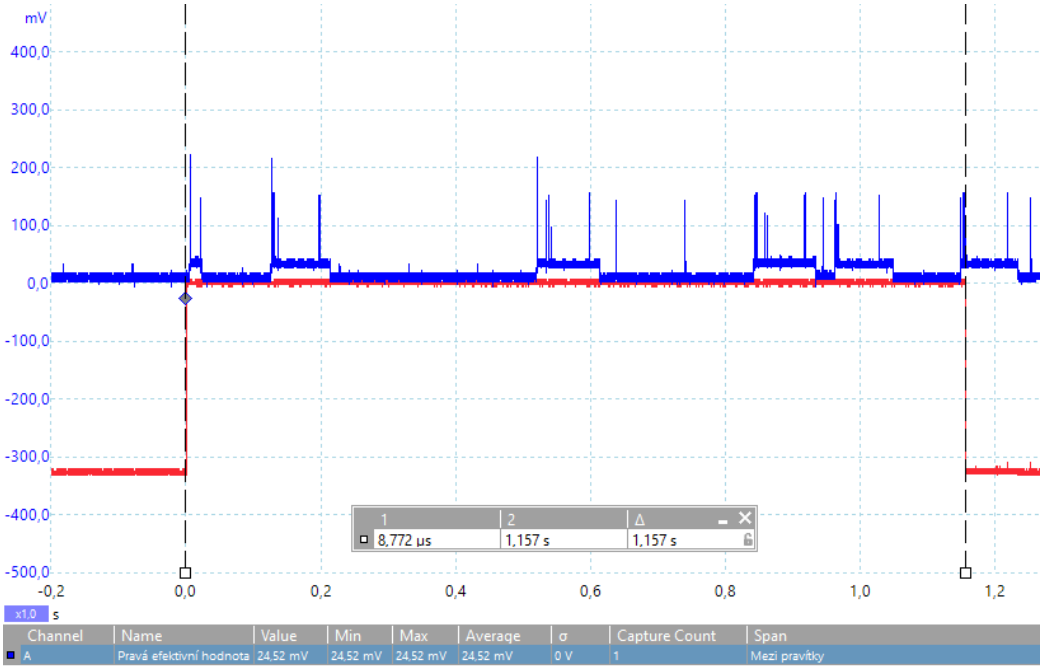
Knihovna *Simplelink* mikrokontroléru *CC3200* implementuje SSL a TLS protokol. Mikrokontrolér *CC3200* má navíc k dispozici koprocesory pro vý-

počet algoritmů RSA, AES a SHA1, čehož *Simplelink* implementace SSL a TLS protokolů využívá. V rámci projektu implementující ECC algoritmy byl proto napsán jednoduchý TLS server a TLS klient používající RSA certifikáty s 2048 bitů dlouhými klíči vygenerované programem *OpenSSL*. Měřen byl čas od začátku volání funkce `connect()` na straně klienta po návrat funkce `accept()` na straně serveru, tedy čas potřebný pro vytvoření TCP spojení a vyjednání TLS připojení. Použitý soubor šifer byl `TLS_RSA_WITH_AES_256_CBC_SHA` a protokol TLS 1.2. Čas nutný k založení TLS relace byl v průměru 1,2 s. Srovnání všech měřených protokolů obsahuje tabulka 3.4. Počet přenesených dat a počet zpráv u protokolu TLS 1.2 byl odhadnut na základě [7].

Měření spotřeby jednotlivých protokolů probíhalo ve školní laboratoři pomocí digitálního osciloskopu. Mezi rozpojené konce napájecího obvodu byl zapojen odpor o hodnotě 0,5 Ω a na koncích odporu bylo osciloskopem měřeno napětí. Pomocí vztahu $I = \frac{U}{R}$ je poté možné dopočítat elektrický proud, který obvodem protéká.

Byla měřena efektivní hodnota (RMS z angl. *Root Mean Square*) napětí U_{RMS} udaná osciloskopem po dobu ověřování implementovanými protokoly a protokolem TLS. Protokol využívající ECC nebyl do měření pro jeho enormní časovou náročnost zahrnut. Na obrázku 3.10 je výsledek měření odběru MCU digitálním osciloskopem *PicoScope 2206* při ověřovnání protokolem TLS. Naměřená data jsou zobrazena v nástroji *PicoScope 6*. Modrá křivka představuje odběr mikrokontroleru měřeného na odporu. Červená křivka je signál GPIO pinu. Ten je před zahájením ověřování programově nastavený na nízkou hladinu *low* a po dokončení ověření je nastaven zpět na vysokou hladinu *high*. Nástrojem *PicoScope 6* je pak měřena efektivní hodnota napětí MCU (modrá křivka) mezi přechodem červené křivky z *high* na *low* a opačně, viz spodní část obrázku. Důvodem zašuměných měření (modrá křivka) je nízká hodnota použitého odporu. Zvýšení hodnoty odporu vedlo k příliš velkému poklesu napětí na procesoru, což způsobovalo opakovaný restart při zapnutí napájení.

Tabulka 3.5 obsahuje srovnání naměřených napětí U_{RMS} a z nich odvozené proudy I_c. Dále je zde odhadnuta energie potřebná pro provedení 1000 ověřovacích cyklů za předpokladu konstantního času běhu protokolu, viz tabulka 3.4. Dále byla změřena efektivní hodnota napětí obvodu U_L udaná osciloskopem po dobu, kdy je procesor v zátěži, konkrétně v době, kdy počítal symetrický klíč obou metod. Bylo naměřeno napětí $U_L = 35{,}5\,\mathrm{mV}$. Budemeli počítat s časem nutným pro výpočet klíče s $t = k = 99$ z tabulky 3.3, je rozdíl ve spotřebě obou algoritmů po výpočtu 10^6 klíčů pouze 0,51 mA h.



Obrázek 3.10: Změřený odběr MCU digitálním osciloskopem *PicoScope 2206* v nástroji *PicoScope 6* při ověřovnání protokolem TLS.

Z výše uvedených čísel je zřejmé, že i velice dobře optimalizovaná implementace algoritmů *ECC*, jakou je *TinyECC*, je alespoň stonásobně pomalejší než implementované metody pro distribuci symetrického klíče. Dále se ukázalo, že protokoly využívající polynomiální a maticovou metodu výpočtu symetrického klíče jsou prakticky totožné. Malý rozdíl tvoří počet přenesených dat, kdy protokol používající maticovou metodu přenáší asi jedenapůlkrát více dat než protokol využívající metodu polynomiální. Nejzajímavější výsledek přináší srovnání těchto dvou protokolů a protokolu TLS, které jsou spotřebou energie i celkovým časem potřebným pro ověření velice vyrovnané, především díky kryptografickým koprocesorům MCU *CC3200*. Ukazuje se tedy, že protokoly ověření využívající asymetrickou kryptografii jsou v kombinaci s patřičnou hardwarovou akcelerací použitelné i pro nízkonákladová zařízení.

Tabulka 3.5: Naměřené hodnoty RMS napětí průběhu vytvoření relace a z nich odvozené údaje.

Protokol	Polynomiální metoda s $t = 99$	Maticová metoda s $k = 99$	TLS 1.2
U_{RMS} [mV]	25,41	26,16	24,52
I_c [mA]	50,82	52,32	49,04
Spotřeba [mA h]	11,3	11,6	16,3

3.4 Problémy a jejich řešení

V průběhu implementace se objevilo několik problémů, jejichž řešení nebylo zcela přímočaré. Tyto problémy proto budou v této kapitole popsány spolu s jejich řešením.

Velikost programu

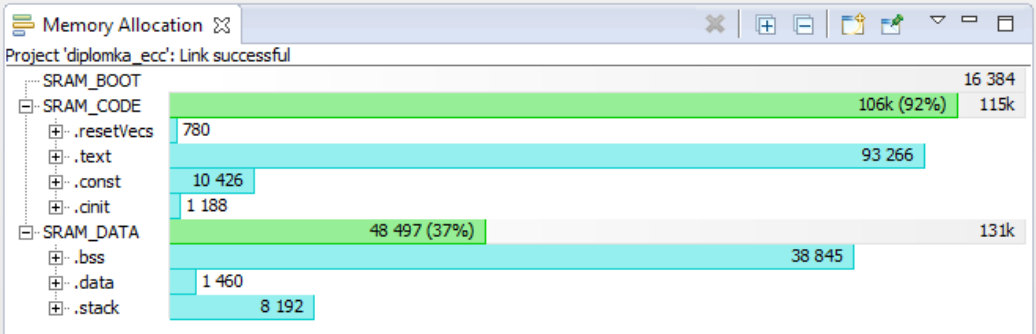
Paměť RAM musí být uživatelem staticky rozdělena na jednotlivé sekce před vlastním spuštěním programu. Toto rozdělení je definováno v souboru `cc3200v1p32.cmd`, který je v kořenovém adresáři každého projektu. Základní rozdělení na sekce pro *bootloader*, program a data je uvedeno v ukázce kódu 3.9. Paměť RAM začíná vždy na adrese `0x20000000`, ale prvních 16 KB je určeno pro *bootloader*. Pokud je překládaný program příliš velký, kompilátor vypíše chybu `#10099-D program will not fit into available memory.` s krátkým popisem a vysvětlením.

```
// Adresa začátku paměti.
#define RAM_BASE 0x20004000

MEMORY
{
   // Aplikace používá pamět' RAM pro program a pro data
   SRAM_BOOT (RWX) : origin = 0x20000000, length = 0x4000   // 16 KB
   SRAM_CODE (RWX) : origin = 0x20004000, length = 0x1C000
   SRAM_DATA (RWX) : origin = 0x20020000, length = 0x20000
}
```

Ukázka kódu 3.9: Definice rozdělení paměti RAM.

Skutečnou velikost použité paměti v jednotlivých sekcích lze zjistit ve

View Memory Allocation, viz obrázek 3.11. Podle údajů zjištěných ve *View Memory Allocation* je nutné zvětšit velikost sekce `SRAM_CODE`. V závislosti na zvětšení této sekce je nutné posunout `origin` sekce `SRAM_DATA` tak, aby se nepřekrývala s předcházející sekcí. Zároveň je nutné zmenšit její velikost tak, aby končila na adrese nejvýše `0x20040000`, tedy 256 KB od `RAM_BASE`.



Obrázek 3.11: Obsah *View Memory Allocation* pro projekt s ověřením algoritmy *ECC*.

Velikost zásobníku úlohy

Velikost zásobníku spuštěné úlohy v TI-RTOS je definována uživatelem při spuštění úlohy, viz konstanta `OSI_STACK_SIZE` v ukázce kódu 3.10.

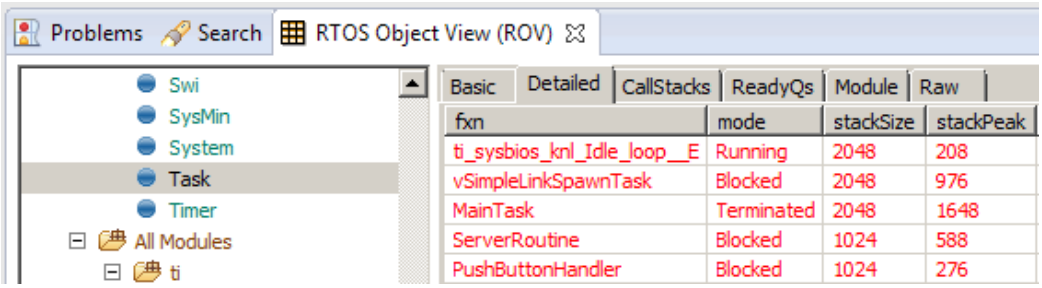
```
1 // Spuštění hlavní úlohy
2 osi_TaskCreate(MainTask, (signed char*)"MainTask", OSI_STACK_SIZE , NULL, TASK_PRIORITY+1,
      NULL );
```

Ukázka kódu 3.10: Spuštění úlohy v TI-RTOS.

Pokud dojde k přetečení zásobníku úlohy, bude program ukončen a v terminálu debugeru CCSv6 se vypíše:

```
1 ti.sysbios.knl.Task: line 373: E_stackOverflow: Task 0x200218e0 stack overflow.
2 xdc.runtime.Error.raise: terminating execution
```

Potřebnou velikost zásobníku lze zjistit ve *View RTOS Object Viewer (ROV)*. V položce *project.out* → *Viewable modules* → *Task* zobrazte detailní výpis. Ve sloupci *stackPeak* je uvedená maximální zaznamenaná velikost zásobníku úlohy, viz obrázek 3.12.



Obrázek 3.12: Zjistění velikosti zásobníku úloh v TI-RTOS.

Požadovaná velikost klíče HMAC funkce

Mikrokontrolér *CC3200* obsahuje kryptografický koprocesor, který implementuje několik haš funkcí a také jejich HMAC ekvivalenty. V práci byla použita funkce SHA-1 HMAC se 128 bitů dlouhým klíčem. Kryptografický koprocesor podporuje klíče dlouhé až 512 bitů. Pokud však má být použit klíč menší než 512 bitů, musí být takový klíč doplněný nulami na délku 512 bitů. Tato skutečnost je zmíněna pouze v komentáři zdrojového kódu funkce `SHAMD5HMACPPKeyGenerate()` v soboru `shamd5.c` knihovny `driverlib`.

4 Závěr

Cílem práce bylo implementovat a porovnat vybrané protokoly ověření založené na symetrické a asymetrické kryptografii z hlediska časové, respektive energetické, náročnosti s přihlednutím k použití protokolů v bezdrátových senzorických sítích.

V teoretické části byly vysvětleny základní požadavky na bezpečnou komunikaci v síti a popsány vlastnosti zařízení s omezenými prostředky, tedy zařízení nejčastěji použitá jako uzly bezdrátových senzorických sítí. Dále byly vysvětleny základní algoritmy symetrické a asymetrické kryptografie. Byl popsán standardní kryptografický protokol TLS a také vysvětleny některé metody pro správu a distribuci symetrických klíčů navržených pro použití v bezdrátových senzorických sítích.

Praktická část práce popisuje konkrétní mikrokontrolér použitý k implementaci a měření. Součástí praktické části je také stručný návod, jak na tomto mikrokontroléru úspěšně spustit programy s implementovanými protokoly. Především jsou zde však vysvětleny a popsány implementace protokolů ověření využívajících algoritmy, nástroje a postupy vysvětlené v teoretické části. Poslední kapitola je věnována vysvětlení způsobu měření a srovnání naměřených výsledků.

Práce ukázala, že vybrané protokoly pro správu symetrických klíčů jsou nenáročené na výpočetní výkon i paměť uzlu, obzvlášť ve srovnání s alternativami používajícími asymetrickou kryptografii. Implementované metody se ukázaly být algoritmicky rovnocenné a rozdíl v jejich energetické náročnosti je zanedbatelný. Zřejmá nenáročnost implementovaných protokolů je však vykoupena omezenou odolností sítě vůči kompromitaci podmnožiny uzlů sítě. Dále byla ukázána výhodnost použití kryptografických koprocesorů, díky kterým dokázal použitý mikrokontrolér vytvořit relaci mezi dvěma uzly standardním bezpečnostním protokolem TLS v průměru jenom jedenapůlkrát pomaleji než pomocí implementovaných protokolů využívající symetrické klíče.

Literatura

[1] Adams, C.: *Understanding PKI : concepts, standards, and deployment considerations*. Boston: Addison-Wesley, 2003, ISBN 9780672323911.

[2] Bidgoli, H.: *Handbook of information security*. Hoboken, N.J: John Wiley, 2006, ISBN 9780470051191.

[3] Blom, R.: An optimal class of symmetric key generation system. In *Advances in Cryptology*, EUROCRYPT, 1985, str. 335–338, ročník 8.

[4] Boyd, C.: *Protocols for Authentication and Key Establishment*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, ISBN 9783662095270.

[5] Chan, H.; Perrig, A.; Song, D.: Random key predistribution schemes for sensor networks. In *2003 Symposium on Security and Privacy, 2003.*, Květen 2003, s. 197–213, doi:10.1109/SECPRI.2003.1199337.

[6] Cusick, T.: *Stream ciphers and number theory*. Amsterdam New York: Elsevier, 1998, ISBN 9780080541846.

[7] Dierks, T.; Rescorla, E.: The Transport Layer Security (TLS) Protocol, Version 1.2. Technická zpráva, Network Working Group, Srpen 200, [Online], [cit. 2017-05-04].
URL https://tools.ietf.org/html/rfc5246

[8] Eschenauer, L.; Gligor, V. D.: A Key-Management Scheme for Distributed Sensor Networks. In *In Proceedings of the 9th ACM Conference on Computer and Communications Security*, ACM Press, 2002, s. 41–47, doi:10.1.1.19.9193.

[9] Gabidulin, E. M.: The theory of codes with maximum rank distance. *Problems of Information Transmission*, 2007, **21**, 1-12.

[10] Gordon, A.: *The official (ISC)2 guide to the SSCP CBK*. Indianapolis, IN: Sybex, 2016, ISBN 9781119278658.

[11] Johnson, B., J.; Kaliski: Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography. Technická zpráva, Network Working Group, Únor 2003.
URL https://tools.ietf.org/html/rfc3447

[12] Knudsen, L.: *The block cipher companion*. Heidelberg New York: Springer-Verlag Berlin Heidelberg, 2011, ISBN 9783642173424.

[13] Koblitz, N.: Elliptic curve cryptosystems. *Mathematics of computation*, 1987: s. 203–209.
URL http://www.ams.org/journals/mcom/1987-48-177/S0025-5718-1987-0866109-5/S0025-5718-1987-0866109-5.pdf

[14] Liu, A.; Ning, P.: TinyECC: A Configurable Library for Elliptic Curve Cryptography in Wireless Sensor Networks. In *2008 International Conference on Information Processing in Sensor Networks (ipsn 2008)*, April 2008, ISBN 978-0-7695-3157-1, s. 245–256, doi:10.1109/IPSN.2008.47.

[15] Liu, D.; Ning, P.: Establishing pairwise keys in distributed sensor networks. In *Proceedings of the 10th ACM Conference on Computer and Communications Security*, CCS '03, New York, NY, USA: ACM, 2003, ISBN 1-58113-738-9, s. 52–61.

[16] Patel, D.: *Information security : theory and practices*. New Delhi: Prentice-Hall of India, 2008, ISBN 9788120333512.

[17] Perrig, A.; Szewczyk, R.; Tygar, J.; aj.: SPINS: Security Protocols for Sensor Networks. *Wireless Networks*, 2002: s. 521–534, ISSN 1572-8196, doi:10.1023/A:1016598314198.

[18] Shannon, C. E.: Communication Theory of Secrecy Systems. *Bell System Technical Journal*, 1949: str. 656–715.

[19] Stallings, W.: *Cryptography and network security : principles and practice*. Upper Saddle River, N.J: Prentice Hall, 1999, ISBN 9780138690175.

[20] Čupek, V.: *Autentizace s využitím lehké kryptografie*. Disertační práce, Vysoké učení technické v Brně, 2016.
URL https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=136225

[21] U.S. National Security Agency: Commercial National Security Algorithm Suite and Quantum Computing FAQ. Leden 2016, [Online], [cit.

2017-05-02].
URL https://cryptome.org/2016/01/CNSA-Suite-and-Quantum-Computing-FAQ.pdf

[22] Vacca, J.: *Public key infrastructure : building trusted applications and Web services*. Boca Raton, Fla: Auerbach Publications, 2004, ISBN 9780849308222.

[23] Viega, J.: *Network security with OpenSSL*. Sebastopol, CA: O'Reilly, 2002, ISBN 9780596002701.

[24] Watro, R.; Kong, D.; Cuti, S.; aj.: TinyPK: Securing Sensor Networks with Public Key Technology. In *Proceedings of the 2Nd ACM Workshop on Security of Ad Hoc and Sensor Networks*, SASN '04, New York, NY, USA: ACM, 2004, ISBN 1-58113-972-1, s. 59–64, doi:10.1145/1029102.1029113.

[25] Wikipedia: Comparison of TLS implementations — Wikipedia, The Free Encyclopedia. 2017, [Online], [cit. 2017-05-04].
URL http://en.wikipedia.org/w/index.php?title=Comparison%20of%20TLS%20implementations&oldid=778225413

[26] Wikipedia: Elliptic curve cryptography — Wikipedia, The Free Encyclopedia. 2017, [Online], [cit. 2017-05-02].
URL http://en.wikipedia.org/w/index.php?title=Elliptic%20curve%20cryptography&oldid=772367584

[27] Wikipedia: Public-key cryptography — Wikipedia, The Free Encyclopedia. 2017, [Online], [cit. 2007-04-26].
URL http://en.wikipedia.org/w/index.php?title=Public-key%20cryptography&oldid=775123694

[28] Wikipedia: RSA (cryptosystem) — Wikipedia, The Free Encyclopedia. 2017, [Online], [cit. 2017-05-01].
URL http://en.wikipedia.org/w/index.php?title=RSA%20(cryptosystem)&oldid=777679673

[29] Xiao, Y.; Rayi, V. K.; Sun, B.; aj.: A survey of key management schemes in wireless sensor networks. *Computer Communications*, 2007: s. 2314–2341, ISSN 0140-3664, doi:https://doi.org/10.1016/j.comcom.2007.04.009, special issue on security on wireless ad hoc and sensor networks.

[30] Yan, Z.: *Trust modeling and management in digital environments : from social concept to system development*. Hershey, PA: Information Science Reference, 2010, ISBN 9781615206834.

[31] Yuan, T.; Zhang, S.; Zhong, Y.: A Matrix-Based Random Key Predistribution Scheme for Wireless Sensor Networks. In *7th IEEE International Conference on Computer and Information Technology (CIT 2007)*, Listopad 2014, s. 991–996, doi:10.1109/CIT.2007.17.

[32] Zhang, Y.; Liang, J.; Zheng, B.; aj.: A Hybrid Key Management Scheme for WSNs Based on PPBR and a Tree-Based Path Key Establishment Method. *Sensors*, 2016, doi:10.3390/s16040509.

Seznam zkratek

AP ..... Access Point

CA ..... Certifikační autorita

CCSv6 ..... Code Composer Studio v6

CPA ..... Chosen-plaintext attack

CRL ..... Certificate Revocation List

ECC ..... Eliptic Curve Cryptography

EPC ..... Electronic Product Code

GPIO ..... General-purpose input/output

KDC ..... Key Distribution Center

IDE ..... Integrated Development Environment

IoT ..... Internet of Things

KMS ..... Key Management Schemes

MAC ..... Message Authentication Code

MCU ..... Microcontroller Unit

MITM ..... Man in the middle

MRD ..... Maximum Rank Distnace

NIST ..... National Institute of Standards and Technology

P2P ..... Peer-to-peer

PKC ..... Public Key Cryptography

PKI ..... Public Key Infrastructure

QoS ..... Quality of Service

RFID ..... Radio Frequency Identification

RMS ..... Root Mean Square

SDK ..... Software Development Kit

SPN ..... Substitution-Permutation Network

SSL ..... Secure Sockets Layer

TLS ..... Transport Layer Security

UART ..... Universal asynchronous receiver/transmitter

WCN ..... Wireless Cellular Network

WLAN ..... Wireless Local Area Network

WMAN ..... Wireless Metropolitan Area Network

WPAN ..... Wireless Personal Area Network

WSN ..... Wireless Sensor Network

WWAN ..... Wireless Wide Area Network

Seznam obrázků

Seznam tabulek

Seznam ukázek kódu