VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

SOFTWAROVÁ A HARDWAROVÁ INJEKTÁŽ CHYB VE VÝPOČTECH CPU A MCU

SOFTWARE AND HARDWARE BASED FAULT INJECTION ATTACKS AGAINST THE CPU AND MCU

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. MAREK LÖRINC**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. MARTIN PEREŠÍNI**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2021/2022

Zadání diplomové práce

Student: **Lörinc Marek, Bc.**
Program: Informační technologie
Obor: Inteligentní zařízení
Název: **Softwarová a hardwarová injektáž chyb ve výpočtech CPU a MCU**
Software and Hardware Based Fault Injection Attacks against the CPU and MCU
Kategorie: Bezpečnost
Zadání:

1. Prostudujte problematiku softwarové a hardwarové injekce chyb ve výpočtech CPU a MCU. Zaměřte se zejména na chyby způsobené změnou napětí.
2. Přečtěte si a seznamte se alespoň se dvěma konkrétními studiemi (z literatury), které se zabývají problematikou změny napětí.
3. Na základě studií proveďte bezpečnostní analýzu a vytvořte protokol, který podrobně popisuje replikaci útoků v praxi.
4. Prakticky ověřte vybrané útoky na reálném HW. Proveďte útoky alespoň na dvou různých procesorech Intel x86 nebo MCU na bázi ARM.
5. Diskutujte o výsledcích dosažených replikací útoků a o vlastnostech obrany proti takovým útokům. Navrhněte rozšíření a zlepšení obrany proti útokům na změnu napětí.

Literatura:

- https://plundervolt.com/
- http://voltjockey.com/
- https://arxiv.org/abs/1912.04870
- https://zt-chen.github.io/voltpillager/
- https://platypusattack.com
- https://media.ccc.de/v/36c3-10859-trustzone-m_eh_breaking_armv8-m_s_security

Při obhajobě semestrální části projektu je požadováno:

- 1-3

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Perešíni Martin, Ing.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 18. května 2022
Datum schválení: 3. listopadu 2021

Abstrakt

Práca sa zaoberá útokmi, ktoré vyvolávajú chyby vo výpočtoch CPU a MCU. K vyvolaniu chyby sa používa krátka zmena napätia CPU alebo MCU. Teoretická časť práce sa zaoberá popisom, ako tieto chyby vyvolať a zneužiť. V tejto časti je popísaná aj najznámejšia ochrana voči hardvérovým útokom, to je prostredie dôveryhodného vykonávania. Vyvolať chybu v tomto prostredí je primárnym cieľom útokov vyvolávajúcich chyby. Praktická časť sa zaoberá replikáciou útokov PlunderVolt a VoltPillager na procesory Intel s aktivovaným prostredím dôveryhodného vykonávania SGX. Bolo vykonaných niekoľko experimentov na vyvolanie chyby v šifrovaní RSA a AES v rámci enklávy SGX. Na tieto chyby boli použité známe analytické metódy, ktorými sa úspešne podarilo získať šifrovací kľuč. Praktická časť sa zaoberá aj replikáciou útoku na mikrokontroléry ARM s aktívnym prostredím dôveryhodného vykonávania TrustZone-M.

Abstract

The thesis deals with attacks that cause faults in CPU and MCU calculations. A short voltage change in CPU or MCU is used to trigger the error. The theoretical part of the thesis deals with the description of how to cause and exploit these errors. This section also describes the most well-known protection against hardware attacks, which is a trusted execution environment (TEE). Inject a fault to TEE is the primary target of fault attacks. The practical part deals with the replication of PlunderVolt and VoltPillager attacks on Intel processors with an activated TEE SGX. Several experiments were performed to trigger faults in RSA and AES encryption within the SGX enclave. To obtain the encryption key from these errors, known analysis methods were used. The practical part also deals with the replication of the attack on ARM microcontrollers with an active TEE TrustZone-M.

Kľúčové slová

útoky vyvolávajúce chyby, hardvérové injektovanie chyby, softvérové injektovanie chyby, PlunderVolt, VoltPillager, chyby zmenou napätia, TrustZone-M(eh), hardvér, Intel SGX, ARM Trustzone

Keywords

fault attacks, software fault injection, hardware fault injection, PlunderVolt, VoltPillager, voltage fault injection, TrustZone-M(eh), hardware, Intel SGX, ARM Trustzone

Citácia

LÖRINC, Marek. *Softwarová a hardwarová injektáž chyb ve výpočtech CPU a MCU*. Brno, 2022. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Martin Perešíni

Softwarová a hardwarová injektáž chyb ve výpočtech CPU a MCU

Prehlásenie

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne pod vedením pána Ing. Martina Perešíniho. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Marek Lörinc
13. mája 2022

Poďakovanie

Rád by som poďakoval vedúcemu svojej práce Ing. Martinovi Perešínimu za odbornú pomoc poskytnutú pri realizácii tejto práce.

Obsah

1 Úvod **4**
1.1 Štruktúra práce 4

2 Útoky vyvolávajúce chyby **5**
2.1 Hardvérové injektovanie chyby 6
2.2 Softvérové injektovanie chyby 8
2.2.1 Manipulácia s rozhraním DVFS 8
2.2.2 Vyvolanie rušivých chýb na pamäti 9
2.3 Charakteristika modelov chyby 9
2.3.1 Model chyby - precízne prevrátenie bitov 9
2.3.2 Model s jednou/viacerými chybami 10
2.3.3 Náhodný/Deterministický model chyby 10
2.3.4 Model chyby - natavenie/resetovanie viacero bitov 10
2.3.5 Model chyby - preskočenie inštrukcie 11

3 Trusted Computing **12**
3.1 Trusted Platform Module 13
3.2 Trusted Execution Environment 13
3.2.1 Intel SGX 15
3.2.2 ARM TrustZone 17

4 Kryptografia **21**
4.1 Šifrovanie RSA 21
4.1.1 Generovanie kľúča 22
4.1.2 Distribúcia kľúča 22
4.1.3 Šifrovanie a dešifrovanie 22
4.1.4 Príklad použitia 23
4.1.5 Využitie čínskej vety o zvyškoch 23
4.2 Šifrovanie AES 24
4.2.1 Šifrovanie 24
4.2.2 Dešifrovanie 29

5 Diferenciálne chybové útoky **30**
5.1 Útok Bellcore 30
5.2 Diferenciálna chybová analýza použitá na šifrovanie AES 31
5.2.1 Prvý krok útoku vyvolávajúceho chyby 32
5.2.2 Analýza prvého kroku útoku vyvolávajúceho chyby 33
5.2.3 Druhý krok útoku vyvolávajúceho chyby 34

5.2.4 Analýza druhého kroku útoku vyvolávajúceho chyby 35
5.2.5 Útok na iné bajty 36

6 Teória útokov využívajúcich zmenu napätia **37**
6.1 PlunderVolt 37
6.1.1 Testovacie nastavenia 38
6.1.2 Vyvolanie chyby násobenia v enkláve 40
6.1.3 Extrakcia kľúčov z enklávy pomocou injektovania chýb 41
6.1.4 Reakcia Intelu 43
6.2 VoltPillager 43
6.2.1 Rozhrania na riadenie napätia CPU 44
6.2.2 Vyvolanie chyby násobenia 45
6.2.3 Extrakcia kľúča z dešifrovania/podpisu RSA-CRT v SGX 46
6.2.4 Porovnanie so softvérovými útokmi 46
6.3 TrustZone-M(eh) 47
6.3.1 Pripojenie útočného HW k MCU 47
6.3.2 Útok na MCU SAM-L11 48

7 Protokol replikácie útokov **50**
7.1 Replikácia SW útoku 50
7.1.1 Potrebné vybavenie na útok 50
7.1.2 SW zmena napätia 51
7.1.3 Overenie náchylnosti na zmenu napätia 51
7.1.4 Aktivácia TEE 51
7.1.5 Vyvolanie chyby v TEE a jej analýza 52
7.2 Replikácia HW útoku 52
7.2.1 Potrebné vybavenie na útok 53
7.2.2 Zostavenie útočného hardvéru 53
7.2.3 Princíp zmeny napätia 53
7.2.4 Vytvorenie a nahratie firmvéru na zmenu napätia 53
7.2.5 Pripojenie utočného HW k základnej doske/pinu MCU 54
7.2.6 Overenie náchylnosti na zmenu napätia 55
7.2.7 Vyvolanie chyby v TEE a jej analýza 55

8 Implementácia **56**
8.1 Replikácia útoku PlunderVolt 56
8.1.1 Potrebné vybavenie 56
8.1.2 Zmena napätia 57
8.1.3 Overenie náchylnosti na zmenu napätia 57
8.1.4 Aktivácia TEE 60
8.1.5 Vyvolanie chyby v TEE 60
8.2 Replikácia útoku VoltPillager 62
8.2.1 Potrebné vybavenie na útok a zostavenie útočného hardvéru 62
8.2.2 Princíp zmeny napätia 63
8.2.3 Nahratie firmvéru na zmenu napätia 64
8.2.4 Pripojenie útočného HW k základnej doske/pinu MCU 64
8.2.5 Overenie náchylnosti na zmenu napätia 67
8.2.6 Vyvolanie chyby v TEE a jej analýza 69

8.3 Replikácia útoku TrustZone-M(eh) 70
8.3.1 Potrebné vybavenie na útok 70
8.3.2 Zostavenie útočného hardvéru 71
8.3.3 Princíp zmeny napätia 71
8.3.4 Vytvorenie a nahratie firmvéru na zmenu napätia 72
8.3.5 Pripojenie útočného HW k základnej doske/pinu MCU 72
8.3.6 Overenie náchylnosti na zmenu napätia 73
8.3.7 Aktivácia TEE 74
8.3.8 Vyvolanie chyby v TEE 75

9 Diskusia **80**
9.1 PlunderVolt 80
9.2 VoltPillager 82
9.3 TrustZone-M(eh) 83

10 Záver **85**

Literatúra **86**

A Obsah priloženého pamäťového média **92**

B Substitučná tabuľka S-box **94**

C Doska VoltPillager vrátane rozšírení **95**

D Možnosti zapojenia schémy na vykonanie útoku TrustZone-M(eh) **96**

Kapitola 1

Úvod

Injektovanie chýb vo výpočtoch MCU a CPU je jedna z najväčších hrozieb, ktorou môže útočník získať citlivé dáta. Dokonca ak sa zameriame na napäťové injektovanie chýb, tak je táto metóda oproti iným jednoduchá a cenovo dostupná. V prípade softvérového injektovania chýb, dokonca bez nákladov na špeciálne vybavenie.

Na ochranu proti akýmkoľvek útokom, primárne však hardvérovým, sa používa prostredie dôveryhodného vykonávania (napríklad Intel SGX, ARM TrustZone), ktoré by malo zabezpečiť dôveru aplikačných dát za akýchkoľvek podmienok pomocou šifrovania. Avšak to sa ukázalo ako nedostatočné zabezpečenie, pretože tieto chyby môže útočník využiť na získanie súkromného kľúča v šifrovaní napríklad pomocou diferenciálnej chybovej analýzy. Dokonca pri niektorých mikrokotroléroch sa úspešne podarilo obísť celé prostredie dôveryhodného vykonávania. To umožnilo získať napríklad PIN od hardvérovej peňaženky TREZOR One.

V minulosti bola často kritizovaná reálna uskutočniteľnosť týchto útokov a ak aj je možné uskutočniť útok, vybavenie musí byť veľmi drahé. To však už dnes neplatí, čo dokazuje aj táto práca, pretože sa úspešne podarilo zreplikovať niekoľko útokov a navyše za nízku cenu útočného hardvéru.

1.1 Štruktúra práce

V kapitole 2 je vysvetlené, čo sú to útoky vyvolávajúce chyby, ich druhy, možná realizácia a najznámejšie druhy chýb, aké sa vyvolávajú. V kapitole 3 je čitateľ oboznámený s technológiami a návrhmi, ktorých cieľom je zvýšiť bezpečnosť výpočtovej techniky prostredníctvom hardvérových vylepšení a súvisiacich softvérových úprav. Základom týchto techník je šifrovanie. Preto aj v nadchádzajúcej kapitole 4 sú prebrané najznámejšie šifrovacie algoritmy, ktoré sa používajú. Cieľom injektovania chyby je najčastejšie vyvolať chybu v tomto šifrovaní. Kapitola 5 ozrejmí čitateľovi, ako sa dá týchto vyvolaných chýb využiť pomocou rôznych známych analýz. Kapitola 6 popisuje 1 softvérový útok a 2 hardvérové útoky, ktoré na injektovanie chyby využívajú zmenu napätia. Tieto útoky boli aj zrealizované a ich implementácia je popísaná v kapitole 8. Implementácia sa riadi protokolom vytvoreným v kapitole 7, ktorý popisuje obecný postup, ako zreplikovať útok vyvolávajúci chybu zmenou napätia. Kapitola 9 popisuje výsledky dosiahnuté replikáciou útokov a návrhy na zlepšenie obrany proti danému útoku.

Kapitola 2

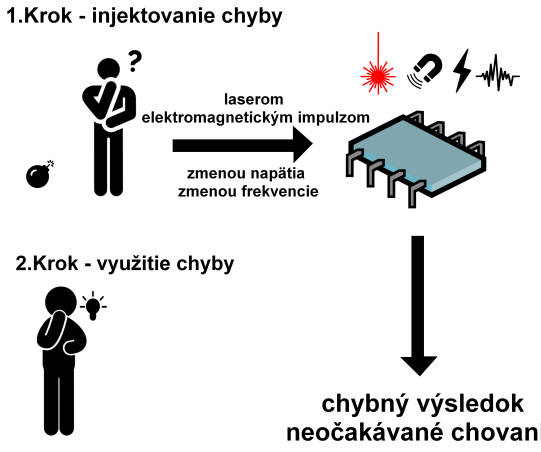
Útoky vyvolávajúce chyby

Útok vyvolávajúci chyby je útok na fyzické elektronické zariadenie ako napríklad čipová karta, hardvérový bezpečnostný modul[1] alebo CPU. Útok spočíva v **zaťažovaní** zariadenia externými prostriedkami (napríklad napätím, svetlom) s cieľom **generovať chyby**. Tieto chyby vedú k **zlyhaniu zabezpečenia** systému (obnovenie kľúča, akceptovanie falošného podpisu, obnovenie PIN kódu) [12].

Úspešný útok pozostáva z dvoch krokov (obrázok 2.1):

- injektovanie chyby
- využitie chyby

Prvý krok spočíva v zavedení chyby vo vhodnom čase počas procesu. Injektovanie chyby môže byť **hardvérové** alebo **softvérové**. Druhý krok spočíva vo využití chybného výsledku alebo neočakávaného správania. Využívanie chýb závisí od návrhu a implementácie softvéru. V prípade algoritmu bude záležať aj na jeho špecifikácii, keďže využitie chyby bude väčšinou kombinované s **kryptoanalýzou**. V závislosti od typu vykonanej analýzy sa bude musieť injektovanie chyby vykonať v **presnom okamihu** alebo približne v danom časovom období [12].



Obr. 2.1: Proces útoku vyvolávajúceho chyby

[1] fyzické výpočtové zariadenie, ktoré chráni a spravuje digitálne kľúče, vykonáva funkcie šifrovania a dešifrovania digitálnych podpisov, silnú autentifikáciu a ďalšie kryptografické funkcie [5]

2.1 Hardvérové injektovanie chyby

Hardvérovo riadené techniky injektovania chýb využívajú **samostatný hardvér** na injektovanie externých porúch. Cieľom je vyvolať fyzické zaťaženie hardvéru a tým vyvolať chybu v softvéri obete [56]. Existuje niekoľko metód na hardvérové injektovanie chyby, ktoré využívajú rôzne nástroje. Nástroje na injektovanie chýb možno porovnávať na základe ich vlastností, ktoré sú zhrnuté v tabuľkách 2.1 a 2.2.

U každého nástroja sú podstatné nasledujúce vlastnosti [10]:

- **Účinok nástroja**. V tomto prípade môže byť účinok globálny alebo lokálny. Lokálna chyba naznačuje, že útočník mohol presne obmedziť účinok chyby na konkrétne miesto v porovnaní s globálnou poruchou, kde útočník nemôže skutočne kontrolovať, kde je chyba vložená.

- **Požiadavky na cieľové zariadenie**. Rozbalenie znamená, že vonkajšia vrstva zariadenia musí byť otvorená, aby bolo možné získať prístup k čipu vo vnútri. Takže bude potrebné zariadenie otvoriť buď chemickými alebo mechanickými prostriedkami.

- **Poškodenie** sa vzťahuje na stav zariadenia po injektovaní chyby. Pri niektorých nástrojoch na injektovanie chýb môže byť cieľové zariadenie zničené. Preto je možné výber nástrojov upraviť v závislosti od dostupnosti a dôležitosti cieľového zariadenia.

- **Detail návrhu zariadenia**. Niektoré z nástrojov vyžadujú znalosť zariadenia, aby sa chyba správne injektovala, zatiaľ čo v ostatných prípadoch sú potrebné čiastočné alebo žiadne znalosti.

- **Presnosť nástroja** na injektovanie porúch. Parameter času sa vzťahuje na presnosť časovania, kedy je porucha injektovaná. Poloha udáva ako presne je potrebné určiť miesto chyby v cieľovom zariadení. Cena injektovacieho zariadenia v tomto prípade veľmi súvisí s presnosťou. Aby sa dosiahla vyššia presnosť, zariadenie na injektovanie porúch bude drahšie.

- **Technické a odborné zručnosti** potrebné na obsluhu zariadenia. V tomto prípade niektoré špecializované nástroje ako laser alebo iónový lúč, môžu byť nebezpečnejšie a môžu spôsobiť potenciálne poškodenie. Preto na bezpečnú obsluhu zariadenia je potrebná vyššia úroveň zručností.

Metóda	**Efekt**	**Rozbalenie**	**Detail návrhu**	**Poškodenie**
Zmena napätia	Globálny	Nie	Čiastočne potrebný	Nie
Zmena frekvencie	Globálny	Nie	Potrebný	Nie
Zahrievanie	Globálny	Nie	Potrebný	Možné
EM impulz	Lokálny	Nie	Čiastočne potrebný	Možné
Svetelný impulz	Lokálny	Áno	Potrebný	Možné
Laserový lúč	Lokálny	Áno	Potrebný	Možné
Svetelné žiarenie	Lokálny	Áno	Nepotrebný	Áno
Iónový lúč	Lokálny	Áno	Potrebný	Áno

Tabuľka 2.1: Najpoužívanejšie metódy na injektovanie chyby a ich vlastnosti 1 [10]

Metóda	Presnosť - Čas	Presnosť - Pozícia	Cena	Zručnosti
Zmena napätia	Mierna	Nízka	Nízka	Stredné
Zmena frekvencie	Vysoká	Nízka	Nízka	Stredné
Zahrievanie	Žiadna	Nízka	Nízka	Nízke
EM impulz	Mierna	Mierna	Nízka	Stredné
Svetelný impulz	Mierna	Mierna	Stredná	Stredné
Laserový lúč	Vysoká	Vysoká	Vysoká	Vysoké
Svetelné žiarenie	Mierna	Nízka	Nízka	Stredné
Iónový lúč	Vysoká	Vysoká	Vysoká	Vysoké

Tabuľka 2.2: Najpoužívanejšie metódy na injektovanie chyby a ich vlastnosti 2 [10]

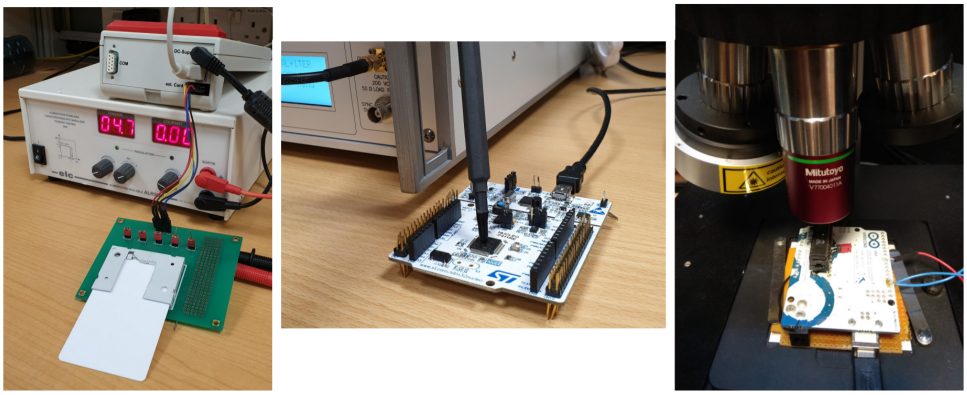
Útok zmenou napätia

Útok zmenou napätia zahŕňa manipuláciu so zdrojom napájania cieľových zariadení. Jednoduchým príkladom môže byť **zníženie napájacieho napätia**, ktoré zavedie chyby časovania v dôsledku oneskorenia šírenia. Ukázalo sa, že toto nedostatočné napájanie spôsobuje chyby v zariadeniach pri kryptografických operáciách. To však neposkytuje dobrú časovú presnosť, čo sťažuje zameranie sa na konkrétnu inštrukciu [11].

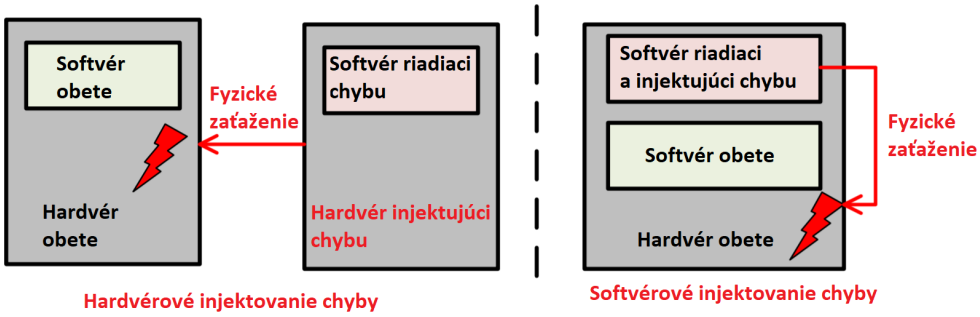
Schopnosť zamerať sa na konkrétnu inštrukciu sa dá dosiahnuť **krátkou zmenou napätia** napájania na konkrétnom mieste v špecifickom časovom okne. Do externého napájania možno vložiť kladné aj záporné zmeny napätia, kde zmena je veľmi krátka (typicky v rozsahu od ns po μs) a snaží sa vyvolať chyby na konkrétnych inštrukciách [57].

Úspešný útok zmenou napätia závisí od mnohých parametrov, ktoré môže útočník ovládať [29]:

- menovité napájanie
- veľkosť zmeny napätia
- šírka impulzu zmeny napätia
- oneskorenie (v súvislosti s cieľovou operáciou)
- teplota CPU/MCU



Obr. 2.2: Útok zmenou napätia na smart kartu (vľavo), elektromagnetickým impulzom(stred) na mikrokontrolér a laserom na mikrokontrolér (vpravo) [10]



Obr. 2.3: Porovnanie hardvérového a softvérového injektovania chyby [56]

2.2 Softvérové injektovanie chyby

Softvérové techniky na injektovanie chyby sú riadené pomocou **škodlivého softvéru** (t. j. softvér na injektovanie chýb a riadiaci softvér), ktorý beží na **rovnakej** hardvérovej platforme ako softvér obete, ktorý má byť napadnutý. Tento škodlivý softvér mení fyzické prevádzkové podmienky cieľového hardvéru. Zatiaľ čo techniky riadené hardvérom vyžadujú fyzický prístup k cieľovému systému, softvérové injektovanie chyby umožňuje aj vzdialené útoky [56].

Spočiatku boli tieto útoky zaujímavé v scenároch, kde útočník je neprivilegovaný alebo dokonca má pridelenú vlastnú oblasť. Avšak s technológiami trusted computing, ako sú: Intel SGX, ARM TrustZone a AMD SEV, privilegovaní útočníci musia byť tiež považovaní za súčasť zodpovedajúcich modelov hrozieb. Stále však exitujú útoky, ktoré si aj s tým dokázali poradiť (viď kapitola 6).

Najpoužívanejšie techniky na softvérové injektovanie chyby sú:

- Manipulácia s rozhraním DVFS
- Vyvolanie rušivých chýb na pamäti

2.2.1 Manipulácia s rozhraním DVFS

V moderných systémoch je frekvenčné škálovanie dynamickým napätím (Dynamic Voltage Frequency Scaling - DVFS) primárne používanou technikou riadenia energie, ktorá reguluje prevádzkové napätie a frekvenciu mikroprocesora na základe pracovnej záťaži. V typickej schéme DVFS ovládače na úrovni jadra riadia frekvenciu a napätie procesora prostredníctvom **regulátorov napätia**, ktoré sú na samostatnom čipe.

Jeden z najznámejších softvérových útokov [52] preukázal, že útočník dokáže zneužiť rozhranie medzi softvérovými ovládačmi a hardvérovým regulátorom na vyvolanie porúch vo viacjadrových procesoroch. Tu útočník používa škodlivý ovládač na úrovni jadra, bežiaci na jadre procesora na nastavenie prevádzkového napätia a frekvencie iného jadra. Táto metóda umožňuje útočníkovi porušiť časové obmedzenia nastavenia jadra obete prostredníctvom pretaktovania a nedostatočného napájania na určité časové obdobie. Útočník ovláda dočasné umiestnenie s koncovými bodmi pretaktovania resp obdobie podvoltovania. Intenzita chyby je určená **frekvenciou pretaktovania** a **hodnotou podvoltovania**.

Táto metóda nevyžaduje žiadny ďalší hardvér na injektovanie chyby ani fyzický prístup k cieľovému zariadeniu.

2.2.2 Vyvolanie rušivých chýb na pamäti

V tejto metóde injektovania chyby, útočník vkladá poruchy do buniek pamäte využitím problémov spoľahlivosti moderných pamätí, ako sú pamäťové čipy DRAM a Flash. Postupné zmenšovanie vo výrobnej technológii umožnilo výrobcom pamätí výrazne znížiť cenu za bit umiestnením menších pamäťových buniek bližšie k sebe. Avšak to tiež zvyšuje elektrické rušenie medzi pamäťovými bunkami. Pristupovanie k pamäťovej bunke elektricky narušuje okolité pamäťové bunky. Narušená pamäťová bunka stráca svoju hodnotu a dochádza k poruche pamäte, keď množstvo elektrického rušenia je za hranicou povoleného šumu danej bunky [56].

Útočník môže spôsobiť poruchy pamäte prostredníctvom neprivilegovaného programu na injektovanie porúch. Tento program opakovane pristupuje k množine pamäťových buniek (t.j. útočné pamäťové bunky) na vyvolanie porúch rušenia v množine napadnutých pamäťových buniek uchovávajúce bezpečnostne citlivé dáta. To umožňuje útočníkovi narušiť pamäťový priestor bezpečnostne citlivého programu z pamäťového priestoru ovládaného útočníkom. Rušivé chyby na pamäti sa väčšinou využívajú u DRAM a NAND Flash pamäťových čipoch [17].

2.3 Charakteristika modelov chyby

Útoky využívajúce chyby sú založené na manipulácii so zariadením takým spôsobom, aby zariadenie fungovalo neštandardne. Z reakcie zariadenia, čo môže byť chybný výsledok, chybové hlásenie, alebo nejaká forma bezpečnostného resetu (vrátane zničenia zariadenia), sa chce útočník dozvedieť niečo o tajomstvách ukrytých v zariadení. Keďže kryptografické algoritmy musia byť verejné, aby im používatelia mohli dôverovať, nič také ako utajenie neexistuje. Preto útočník môže určiť, aké premenné sa používajú a aké hodnoty závisia na tajnom kľúči. To umožňuje určiť, aký druh chyby vyvolá určitú reakciu, ktorú môže útočník pozorovať. Napríklad, ak sa jeden bit v tajnom kľúči preklopil počas útoku a zariadenie túto chybu nezistí, výstupom je chybný výsledok so **špecifickým vzorom**. Porovnaním tohto chybného výsledku so správnym by útočník mohol byť schopný odvodiť jeden bit tajného kľúča. útočník sa môže zamerať aj na tok operácií, tak, že určité operácie sa opakujú alebo vynechávajú. Aby dosiahol a využil želaný efekt, musí mať vedomosti o tom, ako určitý fyzický útok ovplyvní logický tok napadnutého algoritmu. Len potom bude možné určiť pravdepodobnosť úspechu a určiť tajné údaje z chybného výstupu [42].

Popis útoku vyvolávajúceho chyby musí špecifikovať model chyby. Model objasňuje schopnosti útočníka a musí zahŕňať parametre, ako je typ chyby, načasovanie, umiestnenie, presnosť injektovania chyby a počet porúch. Modely prvého rádu predpokladajú, že útočník je schopný vyvolať iba jednu chybu počas vykonávania algoritmu, zatiaľ čo modely druhého rádu predpokladajú, že je možné injektovať viac ako jednu chybu [29].

2.3.1 Model chyby - precízne prevrátenie bitov

Zástancovia útokov vyvolávajúcich chyby sa snažia ukázať efektivitu svojich útokov tým, že toho predpokladajú čo najmenej (napríklad miesto chyby sa považuje za neznáme, čo sa neskôr obnoví pomocou dôkladnej analýzy). Z hľadiska diferenciálneho chybového útoku

(viď kapitola 5), najsilnejší model prehadzuje **jeden konkrétny bit** (kolo šifry ako aj umiestnenie bitu volí útočník). Zvážme vplyv zmeny jedného vstupného bitu operácie `AND`: $y = x_0 \wedge x_1$. Ak je možné presne x_0 prevrátiť, potom posúdením, či sa výstup zmenil alebo nie, je možné získať x_1 (žiadna zmena výstupných prostriedkov $x_1 = 0$, inak $x_1 = 1$). Týmto spôsobom je možné napadnúť akékoľvek hradlo `AND`. Je tiež zaujímavé, že tento model, prelomí protiopatrenia ako detekcia alebo infekcia. Aj keď je šifra 'chránená' detekciou alebo infekciou (čo účinne bráni získať chybný výstup), útočník je schopný skontrolovať, či preklopením jedného bitu operácie `AND` sa zmení výstup alebo nie [10].

2.3.2 Model s jednou/viacerými chybami

Väčšina publikovaných prác o útokoch vyvolávajúcich chyby predpokladá model s jednou chybou. Podľa tohto modelu útočník môže injektovať chyby maximálne raz počas jedného vykonania šifry (to môže mať za následok ovplyvnenie viacerých miest). Takže ak sa vložia dve sady chýb, povedzme, v prvom a poslednom kole šifrovania, bude sa to považovať za porušenie modelu [10].

Generické modelovanie útokov s viacerými chybami sa stáva oveľa náročnejším, pretože teoreticky akákoľvek sada hodnôt parametrov pre vyvolanie prvej chyby môže byť kombinovaná s akoukoľvek sadou hodnôt parametrov pre vyvolanie druhej chyby. Samozrejme, môžu byť zavedené obmedzenia, napríklad rovnaký typ chyby môže byť vyvolaný dvakrát. Aj pri týchto obmedzeniach útočníkov, ich schopnosti sa stávajú silnejšími. Napríklad útočník môže teraz vyvolať chyby aj v premennej a aj v procedúre testujúcej túto premennú. Tým sa zmarí mnoho protiopatrení navrhnutých tak, aby odolali útokom s jednou chybou [29].

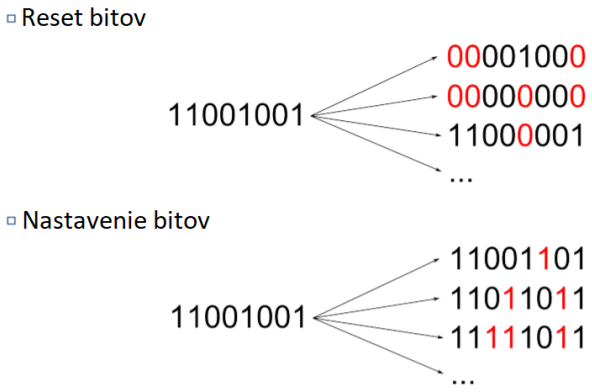
2.3.3 Náhodný/Deterministický model chyby

Model náhodnej chyby je v literatúre najčastejšie používaný [46]. Tu môže útočník ovládať v ktorom kole môžu byť chyby injektované, ale nemôže ovládať hodnotu zmenenú injektovaním chyby. V podstate tu vložená chyba má za následok prevrátenie jedného/viacerých bitov hodnoty operandu. Vo všeobecnosti môže útočník ovládať intenzitu, miesto a trvanie vonkajšieho rušenia, ale nemôže kontrolovať presnosť injektovania chyby (predpokladá sa, že vplyv injektovania chyby nie je známy). V určitých prípadoch sa dá predpokladať, že aj presné miesto chyby ako neznáme (zložitosť útoku sa vynásobí počtom miest chýb [26]). Záležiac na konkrétnom útoku môže byť cieľom pre injektovanie chyby slovo (bajt/nibble) alebo séria bitov. Okrem prevracania bitov, môžu byť použité rôzne modely, kde sú konkrétne bity nastavené na 1 alebo prestavované na 0 [14]. Pre bajtovo orientované šifry ako AES, sa najčastejšie predpokladá náhodná chyba bajtu.

2.3.4 Model chyby - natavenie/resetovanie viacero bitov

Predpoklady k možnosti využitia tohto modelu sú nasledovné [30]:

1. Nastavenie/resetovanie viacero bitov môže byť zavedené injektovaním chyby do daného zariadenia
2. Neznámy otvorený text môže byť šifrovaný viackrát
3. Rôzne chybné alebo správne výstupy je možné porovnať párovo bez odhalenia ich hodnôt



Obr. 2.4: Príklad nastavenia a resetovania viacero bitov

Model s nastavením/resetovaním viacero bitov bol pozorovaný počas injektovania chyby pomocou elektromagnetického impulzu [38] alebo aj pomocou laseru [50], kde sa uvádza, že počas injektovania laserovej chyby do SRAM je model chyby s preklopením bitov nepoužiteľný. Iba chyby pomocou modelu s natavením/resetovaním viacero bitov sú možné. Príklad resetovania a nastavenia viacero bitov je zobrazený na obrázku 2.4.

2.3.5 Model chyby - preskočenie inštrukcie

Preskočenie inštrukcie je chyba, ktorá má za následok preskočenie, čo znamená **nevykonanie** jednej inštrukcie programu za behu (ako keby tok programu preskočil chybný pokyn). Niekoľko prác študovalo preskočenie inštrukcie pomocou EM impulzu. Štúdie sa zaoberajú preskočením jednej inštrukcie [39], ale aj preskočením niekoľko inštrukcií po sebe [49]. Niekoľko prác sa zaoberá aj preskakovaním inštrukcií pomocou lasera, kde je jedna inštrukcia preskočená s vysokou presnosťou a vysokou úspešnosťou. To je následné použité na vykonanie úspešného diferenciálneho chybového útoku na AES [25].

Vhodným kandidátom na využitie preskočenia inštrukcie môže byť nasledujúci kód[2]:

```
bool firmware_is_valid = validate_firmware();
if(!firmware_is_valid)
    abort();
boot();
```

Výpis 2.1: Kód v jazyku C vhodný na demonštráciu preskočenia inštrukcie

Ak sa podarí preskočiť inštrukciu podmienky, potenciálny nebezpečný firmvér môže byť nabootovaný ako bezpečný.

[2]Prevzaté z https://media.ccc.de/v/36c3-10859-trustzone-m_eh_breaking_armv8-m_s_security#t=907

Kapitola 3

Trusted Computing

Počítačová bezpečnosť je v dnešnej dobe informačných technológií nesmierne dôležitá. Väčšina používateľov počítačov sa už dnes stretla s vírusom, spamom, phishingom alebo iným typom malvéru alebo dokonca s ohrozením súkromia a ukradnutím dôverných informácií. Preto už v roku 1980 boli armádou definované kritéria hodnotenia spoľahlivosti PC systémov (TCSEC). Tieto kritéria sa však primárne zaoberajú bezpečnosťou **operačných systémov**, avšak ako sa neskôr ukázalo, dôležitú úlohu zohráva aj **hardvér**. I keď operačný systém považuje hardvér za dôveryhodný, pretože neexituje alternatívny spôsob testovania a overovania správnosti hardvéru, neznamená to, že hardvér nemôže byť napadnutý.

Kvôli tomu vznikla v roku 2003 skupina **Trusted Computing Group (TCG)**, ktorá bola založená s cieľom vyvinúť, definovať a voľne propagovať špecifikácie pre Trusted Computing. Zvýšenie bezpečnosti by malo byť zaistené pomocou tranzitívnych vlastností dôvery. Spoločnosť TCG rozšírila svoj rozsah aj mimo počítačov na iné zariadenia a systémy, ako je úložisko, mobilné zariadenia, servery a periférne zariadenia.

V počítačovej bezpečnosti je koncept dôveryhodného počítania (**Trusted Computing – TC**) možné považovať za taký počítačový systém, ktorého entity majú určitú úroveň záruky, že daná časť, prípadne celok počítačového systému sa chová očakávaným spôsobom. Za entitu je možné považovať človeka počítačového systému, alebo program vykonávajúci na vzdialenom stroji, pričom stupeň záruky zabezpečenia môže pokrývať všetky aspekty systému, alebo iba jeho časť [8].

TC musí zabezpečiť [8]:

- **Ochranu úložiska**. Spoločnosť TCG predstavila platformu **Trusted Computing Platform (TCP)**, čo poskytuje funkciu dôveryhodného disku. Táto funkcia šifruje všetky dáta priamo na disku a rýchlosť šifrovania zodpovedá priepustnosti rozhrania disku, takže proces je v podstate pri bežnej prevádzke neviditeľný pre užívateľa. Čiže ak je dôveryhodný disk ukradnutý, prerobený alebo vyradený z prevádzky, zostáva chránený.

- **Bezpečné Online transakcie**. Na ochranu údajov zákazníkov a zamestnancov pred internetovými útokmi softvér `Personal Information Manager (PIM)`, zabezpečený hardvérovým čipom v TCP, izoluje kontaktné informácie, heslá, bankové prístupové kódy a čísla kreditných kárt. So šifrovacími kľúčmi umiestnenými lokálne v TCP sa kópie automaticky prenesú do manažerá kľúčov `Key Transfer Manager Server` poskytujúci ochranu aj obnoviteľnosť informacie.

- **Ochrana dát a siete pred vírusmi a malvérom**. V tomto prípade spoliehanie sa na antivírus a osobný frewall na prenosných PC nie je prijateľné pre zabezpečenie firemnej siete. Oprávnený používateľ môže získať prístup k sieti z externej stránky, aby ste si jednoducho skontroloval e-maily. Ak počítač používateľa má napríklad vírus, ten sa môže šíriť do siete. S využitím výhod TCP, možno odhaliť tieto klamlivé koncové body. TCP špecifikácia vytvára úroveň dôveryhodnosti v stave koncového bodu a tiež zabezpečuje prítomnosť, stav a verziu softvéru povinných aplikácií.

- **Správa digitálnych práv**. Trusted Computing by umožnil spoločnostiam vytvoriť systém digitálnej správy práv, čo by bolo veľmi ťažké obísť. Príkladom je sťahovanie hudobného súboru. Dalo by sa použiť diaľkové overenie, že hudobný súbor sa odmietne prehrať s výnimkou použitia špecifického hudobného prehrávača, ktorý presadzuje pravidlá nahrávacej spoločnosti. Zapečatené úložisko by zabránilo používateľovi otvoriť súbor pomocou iného prehrávača. Hudba by bola prehrávaná v zaclonenej pamäti, čo by užívateľovi bránilo vytváraniu neobmedzenej kópie súboru, kým beží prehrávanie a bezpečné vstupy/výstupy (I/O) by zabránili zachyteniu toho, čo je odoslané do zvukového systému.

3.1 Trusted Platform Module

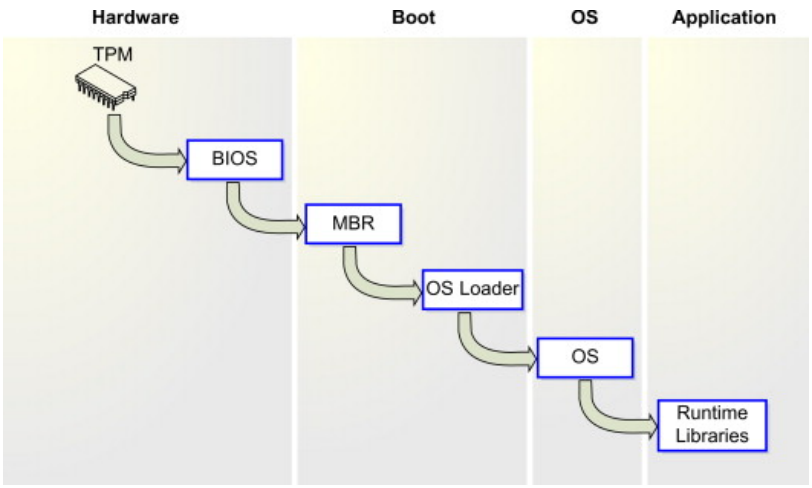
Prvé zariadenie definované TCG je modul dôveryhodného počítania (**Trusted Platform Module** – **TPM**), ktorý je zapuzdrený v rámci TCP pripojením jediného čipu na základnú dosku alebo zabudovaním funkcionality v rámci iných súčiastok. TPM je zvyčajne implementovaný ako mikrokontrolér na základnej doske, ktorý ukladá **heslá**, **digitálne kľúče** a **certifikáty** poskytujúce jedinečnú identifikáciu.

TPM v podstate vytvára **reťazec dôvery** (`chain of trust`, obrázok 3.1), v ktorom TPM vystupuje ako `koreň` overovania integrity viacerých komponentov výpočtového prostredia, ktoré sú potrebné na vytvorenie dôveryhodnej spúšťacej cesty. Každá vrstva (komponent) zodpovedná za overenie ďalšej vrstvy po nej, takže stroj ako celok sa považuje za dôveryhodný iba vtedy, ak je overený celý reťazec. Len jeden nefunkčný článok spôsobí, že reťazec dôvery nebude overený [35].

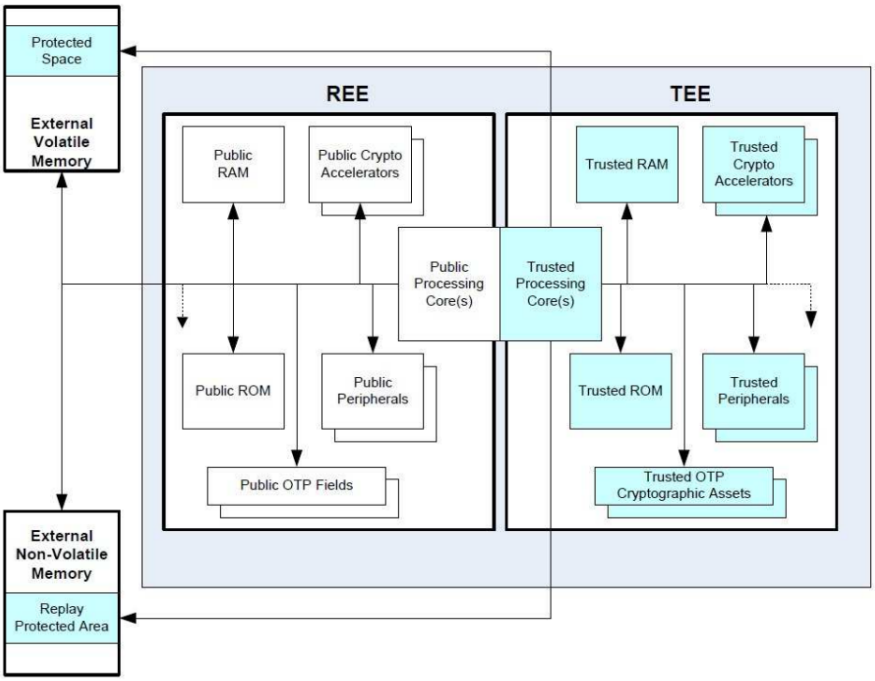
3.2 Trusted Execution Environment

Moduly TPM sú však obmedzené v porovnaní s modernými technológiami dôvery: nepovoľujú ľubovoľné spúšťanie aplikácií, ani zabezpečený vstup/výstup (I/O) nie sú realizovateľné bez dodatočných procesov, ako sú virtuálne stroje s podporou TPM. Jedným z riešení je prostredie dôveryhodného vykonávania (**Trusted Execution Environment - TEE**), ktoré zdieľa hardvér s nedôveryhodným REE (**Rich Execution Enviroment - REE**[1]), napríklad Android, ale spúšťa sa nezávisle s hardvérovo vynútenou **izoláciou**. Umožňuje kritickým aplikáciám spúšťať sa so silnou dôvernosťou a zárukou integrity spolu s potenciálne narušeným REE a zároveň poskytuje funkcie podobné TPM, ako je **zabezpečené úložisko** a **overené spustenie**. TEE má v skutočnosti svoj vlastný **CPU**, **pamäte** a **periférie**. Jediný spôsob komunikácie s externým prostredím je pomocou aplikačného rozhrania TEE [51]. Popísaná architektúra TEE je zobrazená na obrázku 3.2.

[1] prostredie, ktoré poskytuje a riadi širší operačný systém



Obr. 3.1: Ilustrácia reťazca dôvery. Z obrázka je zrejmé, že komponentom môže byť hardvér aj softvér [35].



Obr. 3.2: HW architektúra TEE [6]

TEE sa používajú na spúšťanie citlivých aplikácií, ako je porovnávanie odtlačkov prstov s hardvérovo vynútenou izoláciou prostredníctvom dôveryhodného hardvérového prvku, napríklad CPU. Dôveryhodné aplikácie (**Trusted applications - TA**) sa nachádzajú vo svojich vlastných pamäťových oblastiach a takáto izolácia sa použije na zabránenie neoprávnenému prístupu z priestoru REE. TA môžu alokovať zdieľaný pamäťový priestor s dôveryhodnými aplikáciami alebo odhaľovať funkcie definované vývojármi, ktoré sú sprostredkované vysoko privilegovaným bezpečným monitorovaním.

Najznámejšie implementácie TEE sú **ARM TRUSTZONE** a **Intel SGX**. Intel SGX hľadá uplatnenie na serverových a väčších prenosných zariadeniach, napríklad PC alebo notebooky s procesorom Intel. ARM TRUSTZONE sa primárne používa na zariadeniach s architektúrou ARM. To sú rôzne mikrokotroléry, jednodoskové PC, vrátane Raspberry PI alebo aj systémy na čipe (System on Chip - SoC), kde majú prevažné zastúpenie smartfóny [51].

3.2.1 Intel SGX

Intel SGX prináša rozšírenie architektúry Intel procesorov o sadu nových inštrukcií, ktoré poskytujú vývojárom aplikácií vytvoriť bezpečný a hardvérom izolovaný kontajner nazývaný **enkláva**. Enkláva chráni **dôvernosť** a **integritu** obsahu aplikácií pred všetkým ostatným softvérom na platforme, vrátane potenciálne nedôveryhodného OS. Keďže hlavná pamäť platformy môže byť pod kontrolou útočníka, SGX zaisťuje, že údaje enklávy sú **zašifrované** a integrita je ochránená ešte predtým, ako údaje opustia hranice CPU. Enklávy tiež poskytujú bezpečné ukladanie tým, že umožňujú zapečatenie údajov tak, že ich môže dešifrovať iba špecifická enkláva.

Na rozdiel od TPM, všetky výpočty SGX vykonáva **CPU**, čo vedie k výraznému zvýšeniu výkonu. Typickým prípadom použitia SGX je **izolácia** a **ochrana komponentov** aplikácie citlivých na bezpečnosť. Minimalizácia veľkosti týchto dôveryhodných komponentov znižuje riziko bezpečnostných zraniteľností a znižuje záťaž pre overovateľa počas vzdialeného overovania [31].

Atestácia

S cieľom zvýšiť dôveru, že daný softvér beží bezpečne v rámci enklávy na aktualizovanej platforme Intel SGX a úrovni zabezpečenia, Intel SGX poskytuje atestačný mechanizmus. Intel poskytuje dva typy atestácie enklávy v Intel SGX, **lokálna** a **vzdialená** [4]**.**

Počas lokálnej atestácie, enkláva požiada hardvér o vygenerovanie poverenia, známeho ako `report`, a odoslanie `reportu` do inej enklávy na **rovnakej platforme**, ktorá ho môže overiť. `Report` o enkláve obsahuje nasledovné údaje [54]:

- meranie kódu a dát v enkláve
- hash verejného kľúča u nezávislého dodávateľa certifikátu (ISV)
- používateľské údaje
- iné informácie o stave súvisiace s bezpečnosťou
- podpisový blok nad vyššie uvedenými údajmi

Pre vzdialenú atestáciu môže aplikácia poslať `report` enklávy do špeciálnej enklávy (**Quoting Enclave - QE**), aby sa vytvoril typ poverenia, ktorý odráža stav enklávy a

platformy. Toto poverenie sa nazýva `QUOTE` a je podpísané súkromným kľúčom EPID [4]. QE je enkláva poskytovaná spoločnosťou Intel, ktorá dokáže spracovať `report` enklávy a previesť `report` na `QUOTE`. Iba QE má prístup ku kľúču Intel EPID. `QUOTE` je dátová štruktúra používaná na diaľkovú atestáciu a jej hlavný obsah je rovnaký ako obsah `reportu`. Štruktúra `QUOTE` obsahuje nasledujúce údaje [54]:

- meranie kódu a dát v enkláve
- hash verejného kľúča u nezávislého dodávateľa certifikátu (ISV)
- ID produktu
- bezpečnostné číslo verzie enklávy
- atribúty enklávy
- používateľské údaje
- atribúty enklávy
- podpisový blok nad vyššie uvedenými údajmi

Implementácia

Na implementáciu tohto konceptu SGX predstavuje novú sadu inštrukcií pre architektúru x86. To umožňuje systému BIOS alokovať oblasť pamäte obmedzenú na použitie procesorom (`Processor Reserved Memory - PRM`). Dáta enklávy a kód sú umiestnené vo vyrovnávacej pamäti `Enclave Page Cache (EPC)`, podmnožine PRM. Aby sa zabezpečila jej dôvernosť, EPC je šifrovaná pomocou `Memory Encryption Engine (MEE)`, čo je prvok CPU, ktorý zaisťuje, že dáta enklávy bežia iba v rámci limitov CPU. Šifrovací kľúč je **náhodne generovaný** pomocou CPU a je zmenený pri každom cykle a nikdy neprekračuje svoje hranice. Neoprávnené požiadavky na pamäť enklávy CPU blokuje a zaobchádza s nimi ako s neexistujúcimi adresami pamäte. Teda iba enkláva má prístup k vlastným informáciám.

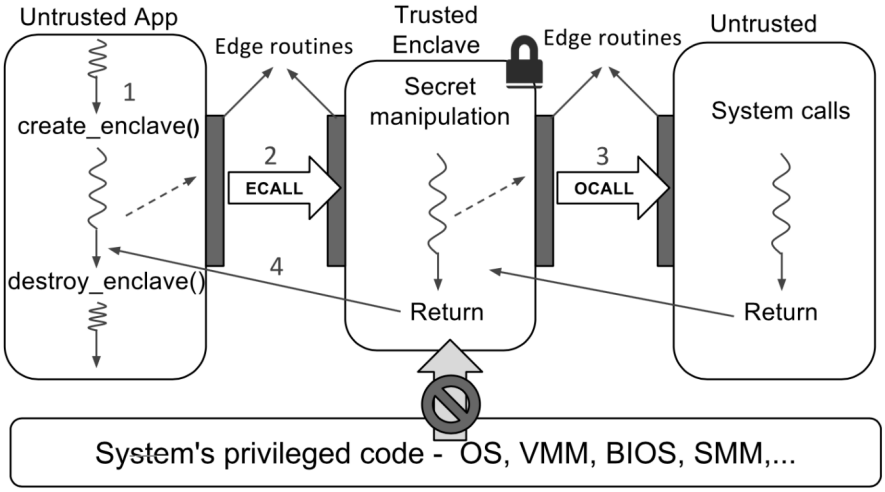
Systémový kód, ktorý riadi fyzickú pamäť počítača, používa aj inštrukcie SGX na správu EPC. Avšak takýto spôsob sa považuje za nespoľahlivý v modeli hrozby SGX, preto sa vykonávajú bezpečnostné kontroly na zabezpečenie oprávnenosti všetkých operácií zahŕňajúce EPC. Na túto kontrolu CPU používa štruktúru `Enclave Page Cache Map (EPCM)`, kde je matica so vstupom pre každú stránku EPC. Tá obsahuje o nej tri základné informácie [19]:

- Či sa stránka používa
- Do ktorej enklávy stránka patrí
- Typ stránky

To umožňuje načítať niekoľko enkláv naraz, bez rušenia medzi nimi.

Aby sa zabezpečilo, že budú môcť bežať iba autentizované enklávy, Intel poskytuje popísaný atestačný **mechanizmus podpisovania enklávy**. Akonáhle je enkláva inštanciovaná, CPU vykoná svoje meranie a uloží ho do registra `MRENCLAVE`. Jeho hodnota je potom konfrontovaná so štruktúrou podpisu. To znamená, že ak sú identické, enkláva bude úspešne načítaná. Register na uloženie hashu autorského kľúča je tiež vyhradený: `MRSIGNER` [31].

Prístup do enklávy sa vykonáva cez obmedzené a dobre definované rozhranie pozostávajúce z dvoch typov funkcií: **ECALL** je funkcia, ktorú môže nedôveryhodná aplikácia



Obr. 3.3: Postup spustenia aplikácii Intel SGX [19]

používať na volanie vykonania v rámci enkláv a **OCALL** je funckia, ktorú môže enkláva použiť na prístup k nedôveryhodným prvkom. Tieto funkcie musia byť definované v špeciálnom súbore, pomocou jazyka `Enclave Definition Language (EDL)`. Ten obsahuje deklaráciu funkcie, podobnú jazyku C, ale so špeciálnymi atribútmi na určenie smeru, veľkosti a typu údajov, ktoré prekročia hranice enklávy. Z tohto súboru, špeciálny nástroj (Edger8er) generuje rutiny(`edge routine`), ktoré bezpečne manipulujú s príslušnými parametrami [31].

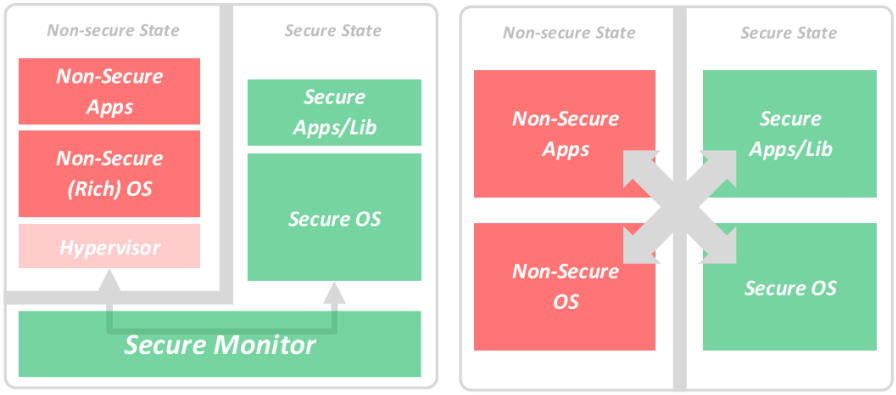
Postup spustenia aplikácií Intel SGX

Na obrázku 3.3 je zobrazený postup spustenia aplikácie využívajúcej Intel SGX. Ako prvé, nedôveryhodná aplikácia **vytvorí enklávu (1)** a akonáhle potrebuje manipulovať s citlivými údajmi, použije volanie **ECALL** na presunutie spustenia **do enklávy (2)**. Avšak enklávy nemajú priamy prístup k systémovým zdrojom. Taktiež nemôžu spúšťať funkcie z dynamických knižníc. Ak je niečo z toho potrebné, enkláva požije **OCALL (3)**. Keď enkláva už nie je potrebná, zničí sa a jej **údaje sa odstránia (4)**.

Bezpečnosť údajov je zaručená kým sú dáta umiestnené v rámci enklávy a je stratená, keď je enkláva zničená. Je možné použiť aj mechanizmus na zapečatenie, ak je potrebné trvalé uloženie. Pri ochrane údajov pomocou kryptografie, je potrebné venovať osobitnú pozornosť úložisku tajných kľúčov. Architektúra SGX však umožňuje CPU vygenerovať jedinečný **128-bitový kľúč AES-GCM**, čo je kľúč na zapečatenie, pre konkrétnu enklávu bežiacu na konkrétnej platforme na šifrovanie údajov. Kľúč na zapečatenie nie je potrebné uložiť, pretože ho generuje CPU za behu a nikdy neopustí hranice procesora. Vývojár sa môže rozhodnúť pripojiť kľúč na zapečatenie do registra `MRENCLAVE` alebo `MRSIGNER`, aby len tá istá enkláva alebo akákoľvek enkláva od toho istého autora mohla odpečatiť údaje. Pečatiaci kľúč je tiež pripojený k jedinečnému trvalému kľúču CPU. To znamená, že iba CPU, ktorý zapečatil dáta, ich bude môcť odpečatiť [19].

3.2.2 ARM TrustZone

TrustZone je hardvérový bezpečnostný mechanizmus zavedený od roku 2004 do aplikačných procesorov Arm (architektúra ARMv6, Cortex-A). V roku 2016 bola TrustZone upravená



Obr. 3.4: Technológia TrustZone pre ARM Cortex-A (vľavo) a pre ARM Cortex-M (vpravo) [44]

tak, aby pokrývala novu generáciu mikrokontrolérov Arm (Cortex-M). TrustZone zahŕňa bezpečnostné rozšírenia pre systémy na čipe (SoC) pokrývajúce **procesor**, **pamäť** a **periférie** [24]. Bezpečnosť TrustZone je založená na myšlienke rozdelenia celého hardvéru a softvéru systému na čipe (System on Chip - SoC) do dvoch svetov: **bezpečný svet** a **normálny svet**. Bezpečným svetom je označené všetko čo beží, ak je stav procesora označený ako bezpečný. Normálny svet je označené všetko čo beží, keď je procesor v nezabezpečenom stave. Hardvérové bariéry sú vytvorené, aby zabránili komponentom normálneho sveta v prístupe k bezpečným zdrojom. Bezpečný svet nie je obmedzený. Konkrétne, systém bráni normálnemu svetu v prístupe k [41]:

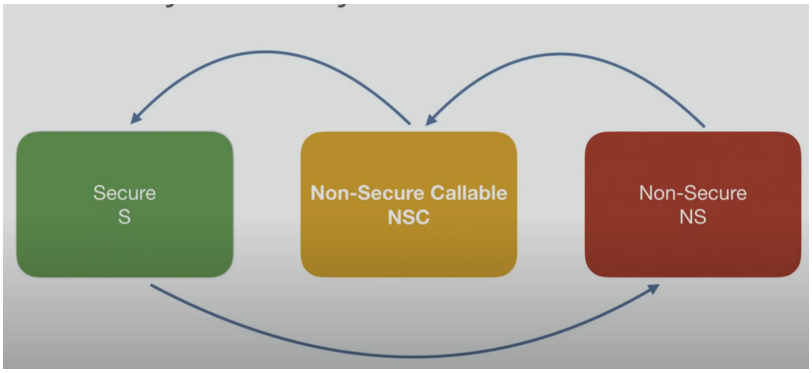
- Oblasti fyzickej pamäte označenej ako bezpečná
- Kontrole systému, ktorá sa vzťahuje na bezpečný svet
- Prepínaniu stavov mimo schválených mechanizmov

TrustZone pre aplikačné procesory

TrustZone pre aplikačné procesory bol vyvinutý pre zvýšenie bezpečnosti pre procesory rady Cortex-A. Ako už je spomenuté v sekcii 3.2.2, najdôležitejšia architektonická zmena na úrovni procesora spočíva v zavedení dvoch svetov: bezpečný svet a normálny svet. Obrázok 3.4 ilustruje tento koncept. V danom okamihu pracuje výlučne procesor v jednom z týchto svetov. Svet, v ktorom procesor momentálne pracuje, je určený hodnotu nového 33. bitu procesora, známeho aj ako **nezabezpečený bit** (`Non-Secure`). Hodnota tohto bitu sa načíta z registra `Secure Configuration Register (SCR)` a šíri sa po celom systéme až po pamäť a periférne zbernice.

TrustZone predstavuje extra režim procesora, ktorý je zodpovedný za zachovanie stavu procesora vždy, keď dôjde k zmene sveta. Tento režim procesora sa nazýva **režim monitora** a zaisťuje, že aktuálny stav sveta, z ktorého procesor odchádza, je uložený a stav sveta, do ktorého vstupuje, je obnovený. Procesor je možné uviesť do tohto režimu pomocou novej privilegovanej inštrukcie `Secure Monitor Call - SMC`, alebo vhodnou konfiguráciou hardvérovej výnimky alebo pomocou prerušenia (IRQ, FIQ).

Aby sa posilnila hardvérová izolácia medzi svetmi, procesor rozšíril verzie špeciálnych registrov, ako aj niektoré systémové registre (prístupné cez koprocesor 15 na Armv7-A a



Obr. 3.5: TrustZone-M bezpečnostné stavy pamäte[2]

pomocou inštrukcií MSR a MRS na Armv8-A). V normálnom svete, bezpečnostne kritické systémové registre a bity jadra procesora sú buď úplne skryté alebo podmienené súborom prístupových oprávnení kontrolovaných softvérom bezpečného sveta [44].

TrustZone-M

Z vysoko-úrovňovej perspektívy, TrustZone-M pre mikrokontroléry ARM Cortex-M je podobný variantu v procesoroch Cortex-A. V oboch prevedeniach môže procesor pracovať v zabezpečenom alebo nezabezpečenom stave. Existujú však dôležité rozdiely, pretože Cortex-M bol optimalizovaný pre rýchlejšie prepínanie kontextu a deterministické vykonávanie. Výsledkom je, že základné mechanizmy TrustZone-M sa výrazne líšia od pôvodnej špecifikácie TrustZone. V TrustZone-M je stav vykonávania založený na **mapovaní pamäte** a prepínanie medzi svetmi prebieha **automaticky** pri vybavovaní výnimiek kódu [44].

TrustZone-M **neobsahuje režim monitorovania**, čo znižuje latenciu prepínania medzi svetmi, čoho výsledkom sú efektívnejšie prechody. Na premostenie softvéru medzi oboma svetmi podporuje TrustZone-M viacero bezpečných vstupných bodov. Pre tento účel sada inštrukcií architektúry (`Instruction Set Architecture - ISA`) bola rozšírená o tri nové inštrukcie vrátane **zabezpečenia brány** (`Secure Gateway - SG`). S výnimkou ukazovateľov zásobníka a niekoľkých špeciálnych registrov, v architektúre Cortex-M, je väčšina registrov **zdieľaná** medzi bezpečnými a nezabezpečenými stavmi.

Čo sa týka pamäte, fyzický adresový priestor je rozdelený na **zabezpečené** a **nezabezpečené** sekcie. Navyše, v TrustZone-M je zabezpečený pamäťový priestor rozdelený na dva typy: **bezpečný** a **nezabezpečený** (`Non-Secure Callable - NSC`). NSC je špeciálne bezpečné pamäťové miesto, ktoré sa používa na uchovávanie inštrukcií SG. To je vstupným bodom každého explicitného prechodu medzi nezabezpečenými a zabezpečenými stavmi. Čiže pri prechode z nezabezpečeného stavu **nie je** možný priamy prechod do zabezpečeného stavu, ale použije sa ešte medzistav NSC. Naopak zo zabepečeného stavu je možné prejsť priamo do nezabezpečeného, ako je aj zobrazené na obrázku 3.5 [45].

Stav pamäte môže byť nakonfigurovaný pomocou jednotky `Secure Attribution Unit (SAU)` a/alebo jednotky `Implementation Defined Attribution Unit (IDAU)`. Čiže tieto jednotky určujú, aký stav má adresa. Prísnejší stav vyhráva. To znamená, že ak napríklad SAU má bezpečný stav a IDAU nezabezpečený, adresa bude mať **zabezpečený stav**.

[2]Prevzaté z https://media.ccc.de/v/36c3-10859-trustzone-m_eh_breaking_armv8-m_s_security

Niektoré mikrokontroléry, majú iba **jednu** z týchto jednotiek. Napríklad MCU `SAM L11` má iba jednotku IDAU, čiže tá priamo určuje stav adresy.

Jednotka `TrustZone-aware Memory Protection Unit (MPU)` umožňuje, že každý svet môže mať lokálnu sadu oprávnení na prístup do pamäte poskytnutím iného rozhrania MPU pre každý svet. Riadič prerušení `Nested Vectored Interrupt Controller (NVIC)` umožňuje, aby boli prerušenia konfigurované ako bezpečné alebo nezabezpečené. Ak stav prichádzajúceho prerušenia sa rovná aktuálne vykonávanému stavu, sekvencia výnimiek je podobná ako u predchádzajúcich procesorov rady M. Hlavný rozdiel nastáva, keď dôjde k nezabezpečenému prerušeniu počas vykonávania zabezpečeného kódu. V tomto prípade procesor automaticky vloží všetky zabezpečené informácie do zabezpečeného zásobníka a vymaže obsah z registrov [45].

Kapitola 4

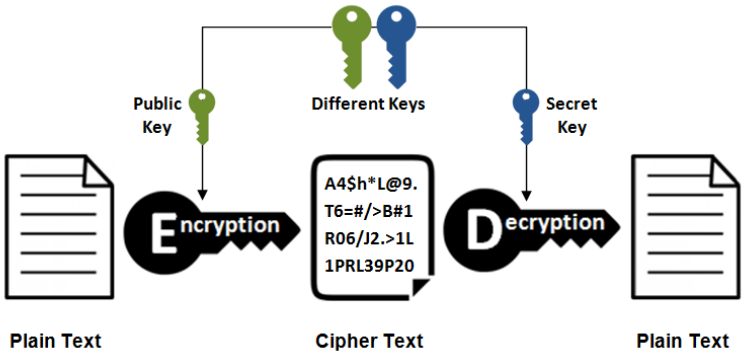
Kryptografia

Táto práca sa primárne zaoberá útokmi, ktoré vyvolávajú chyby počas vykonávania kódu v prostredí dôveryhodného vykonávania **Intel SGX** a **ARM TRUSTZONE**. Aby komunikácia nebola odhalená, tieto prostredia ju šifrujú najčastejšie pomocou šifrovacích algoritmov **AES(Advanced Encryption Standard)** a **RSA(Rivest–Shamir–Adleman)**. Cieľom tejto kapitoly je vysvetliť, ako šifrovacie algoritmy RSA a AES fungujú. Až potom v ďalšej kapitole môže byť vysvetlené, aké chyby sa v týchto šifrovacích algoritmoch vyvolávajú a ako týchto chýb zneužiť, pretože to priamo súvisí s ich funkčnosťou.

4.1 Šifrovanie RSA

RSA (Rivest–Shamir–Adleman) je šifra s **verejným kľúčom**(asymetrické šifrovanie, obrázok 4.1), ktorá sa často používa na bezpečný prenos údajov. Pri šifrovaní s verejným kľúčom je šifrovací kľúč verejný a **odlišný** od dešifrovacieho kľúča, ktorý je utajený (súkromný). Používateľ RSA vytvorí a zverejní verejný kľúč na základe dvoch veľkých **prvočísel** spolu s pomocnou hodnotou. Prvočísla sú utajené. Správy môže zašifrovať ktokoľvek prostredníctvom verejného kľúča, ale dekódovať ich môže iba niekto, kto pozná prvočísla [48].

Bezpečnosť RSA je postavená na predpoklade, že rozložiť veľké číslo na súčin prvočísel (faktorizácia) je veľmi náročná úloha. Z čísla $n = pq$ je teda v rozumnom čase prakticky nemožné zistiť činitele p a q, pretože nie je známy žiadny algoritmus faktorizácie, ktorý



Obr. 4.1: Asymetrické šifrovanie [21]

by pracoval v polynomiálnom čase voči veľkosti binárneho zápisu čísla n. Naproti tomu násobenie dvoch veľkých čísel je elementárnou úlohou [3].

Algoritmus RSA zahŕňa štyri kroky: generovanie kľúča, distribúcia kľúča, šifrovanie a dešifrovanie.

4.1.1 Generovanie kľúča

Kľúč pre RSA je generovaný nasledovným spôsobom [36]:

1. Zvolia sa 2 rôzne veľké a náhodné prvočísla p a q.
2. Vypočíta sa ich súčin $n = pq$.
3. Spočíta sa hodnota **Eulerovej funkcie** $\varphi(n)$, ktorá určuje počet všetkých prirodzených čísel takých, že $1 \leq k \leq n$ a ich najväčší spoločný deliteľ $NSD(k, n) = 1$. Z toho je zrejmé, že pre prvočíslo p je $\varphi(p) = p - 1$. K výpočtu hodnoty Eulerovej funkcie pre obecný argument n sa využíva multiplikatívnosti, ktorú je možné využiť pri 2 nesúdeliteľných číslach. Kedže prvočísla p a q sú nesúdelitelné, Eulerova funkcia sa vypočíta ako:

$$\varphi(n) = \varphi(pq) = \varphi(p) * \varphi(q) = (p - 1) * (q - 1)$$

4. Zvolí sa celé číslo e(verejný exponent) menšie ako $\varphi(n)$ a ktoré je s $\varphi(n)$ nesúdeliteľné.
5. Zvolí sa číslo d (privátny exponent) tak, aby platilo

$$e * d \mod \varphi(n) = 1$$

Privátnym kľúčom sa stáva dvojica čísel (n, d) a verejným kľúčom dvojica (n, e).

4.1.2 Distribúcia kľúča

Predpokladajme, že Bob chce poslať informácie Alice. Ak sa rozhodnú použiť RSA, Bob musí poznať Alicin verejný kľúč na zašifrovanie správy a Alica musí na dešifrovanie správy použiť svoj súkromný kľúč.

Aby mohol Bob posielať zašifrované správy, Alica odošle svoj verejný kľúč (n, e) Bobovi spoľahlivou, ale nie nevyhnutne tajnou cestou. Alicin súkromný kľúč d nie je nikdy distribuovaný.

4.1.3 Šifrovanie a dešifrovanie

Po vypočítaní všetkých potrebných premenných na generovanie kľúča je možné zašifrovať a dešifrovať správu pomocou algoritmu. To je samozrejme dané skutočnosťou, že bol vytvorený verejný kľúč, ktorý pozostáva z n a e. Vzorec pre zašifrovanie správy m je nasledujúci:

$$c = m^e \mod n$$

Pre dešifrovanie šifrovaného textu c sa používa vzorec [7]:

$$m = c^d \mod n$$

4.1.4 Príklad použitia

V tomto scenári vystupujú dvaja herci: Alica a Bob. Alica chce poslať správu Bobovi a chce ju zašifrovať pomocou šifrovania RSA. Bob vyberie dve prvočísla, $p = 101$ a $q = 113$. Pomocou týchto dvoch čísel sa vypočíta premenná n ako $n = p * q = 11413$. Ďalším krokom je výpočet hodnoty $\varphi(n)$ tak ,že:

$$\varphi(n) = (p-1) * (q-1) = 100 * 112 = 11200$$

Bob vyberie číslo exponentu, povedzme napríklad $e = 3$. Zvyčajne by bolo rozumné vybrať väčší exponent z dôvodu bezpečnosti. V tomto prípade ale použijeme malý počet z dôvodu jednoduchosti. Pomocou tohto exponentu je Bob schopný vygenerovať súkromný kľúč, takže:

$$d = e - 1 \mod 11200 = 6597$$

Bob zverejní premenné verejného kľúča, n a e, ktoré možno ďalej použiť na šifrovanie správ vo forme čistého textu. Proces generovania kľúča je dokončený a proces prenosu správy pokračuje.

Nasledujúcim krokom je šifrovanie. Uvažujme, že Alica chce odoslať správu s jedným znakom "U". Na prevod znaku na celé číslo využijeme ASCII tabuľku, kde znak "U" má hodnotu 85. Alica chce odoslať šifrovaný text, preto vypočíta hodnotu c ako:

$$c = 85^3 \mod 11413 = 9236$$

Túto hodnotu odošle Bobovi. Pri väčšom počte znakov sa vypočíta hodnota c pre každý znak.

Bob očividne pozná hodnotu tajného kľúča(6597), čoho využije na dešifrovanie správy prijatej od Alici nasledovne [7]:

$$9236^{6597} \mod 11413 = 85 = \text{U}$$

4.1.5 Využitie čínskej vety o zvyškoch

Pre zvýšenie efektivity veľa knižníc využíva na dešifrovanie nasledujúcu optimalizáciu založenú na čínskej vete o zvyškoch.

Keďže príjemca pozná tajné prvočísla p a q, môže vypočítať nasledujúce modulárne komponenty [55]:

1. $d_p = d \mod (p-1)$ a $d_q = d \mod (q-1)$
2. $C_p = C \mod p$ a $C_q = C \mod q$
3. $M_p = C_p^{d_p} \mod p$ a $M_q = C_q^{d_q} \mod q$

To zníži čas výpočtu keďže d_p,$d_q < d$ a C_p,$C_q < C$. V skutočnosti je ich veľkosť približne polovičná a preto v ideálnom prípade dosiahneme približne 4 násobné zrýchlenie. Konečná správa je potom vypočítaná ako [55]:

$$M = [M_p(q^{-1} \mod p)q + M_q(p^{-1} \mod q)p] \mod n$$

4.2 Šifrovanie AES

Algoritmus AES je konkrétna realizácia všeobecného algoritmu Rijndael, pomenovaného podľa jeho tvorcov Joana Daemena a Vincenta Rijmena [20].

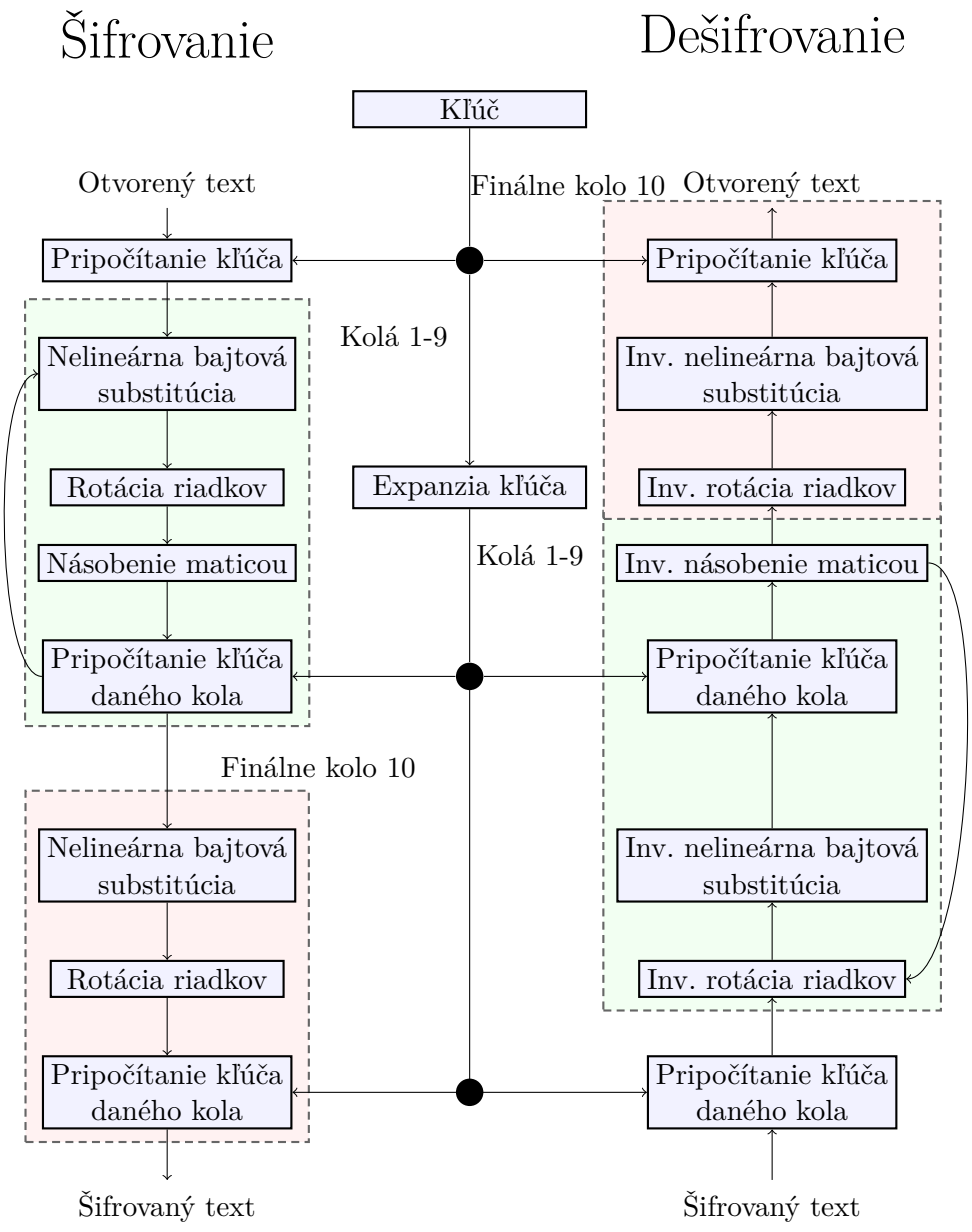
AES šifra je rýchla v softvéri aj v hardvéri (napríklad s využitím AES-NI) a na rozdiel od svojho predchodcu DES nepoužíva Feistelovu sieť. Namiesto toho využíva tzv. **SP-sieť**, teda substitučno-permutačnú. SP-sieť určuje počet matematických operácií, ktoré sa vykonávajú v algoritmoch blokovej šifry [37].

Vstup a **výstup** algoritmu AES tvorí **dátový blok** s dĺžkou 128 bitov (16 bajtov) reprezentujúca otvorený text. Týchto 16 bajtov je reprezentovaných v **4x4 matici**, s ktorou AES pracuje ako s maticou bajtov. Dĺžka **šifrovacieho kľúča** je pri algoritme AES volená z trojice možných hodnôt 128, 192 a 256 bitov. Iné dĺžky vstupu, výstupu a šifrovacieho kľúča nie sú štandardom povolené. Veľkosť kľúča rozhodne o **počte kôl** šifrovacieho algoritmu AES. To znamená, že pre 128-bitový kľúč prebehne 10 kôl šifrovania, pre 192-bitový kľúč 12 kôl a pre 256-bitový kľúč 14 kôl [1].

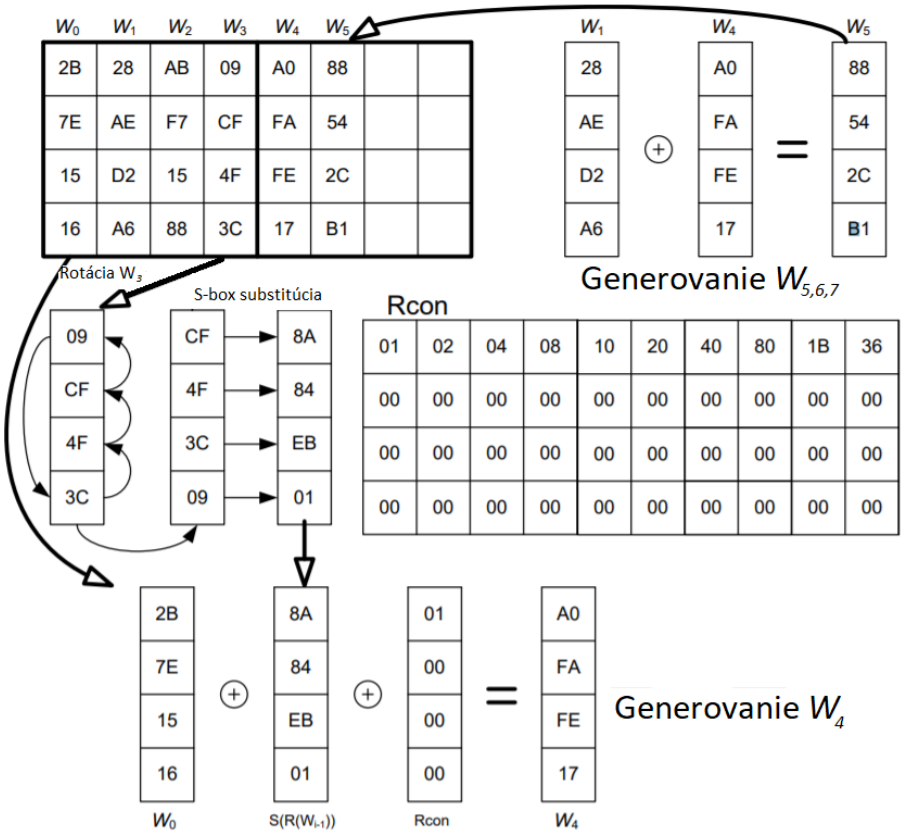
4.2.1 Šifrovanie

Proces sifrovania štandardom AES je zobrazený na ľavej strane obrázka 4.2 a je možné ho rozdeliť do troch základných fáz: **počiatočná fáza**, fáze, kde sú **vyhotovené kolá** a fáza **finálneho kola**. Pred samotným procesom šifrovania dôjde k expanzii šifrovacieho kľúča z pôvodnej veľkosti na veľkosť potrebnú pre celý proces šifrovania (všetky kolá). Počiatočná fáza šifrovania obsahuje iba jednu operáciu a to "Pripočítanie kľúča", kde je pripočítaný pôvodný šifrovací kľúč k poľu `stav`(v tomto prípade sa jedná o otvorený text). V jednotlivých kolách sú vykonávané operácie postupne a zo základných štyroch operácií realizovaných počas algoritmu AES sú iba operácie "Pripočítanie kľúča" parametrizované vstupným šifrovacím kľúčom. Všetky ostatné operácie sú pri šifrovaní a dešifrovaní reverzibilné a nezaručujú bezpečnosť, iba konfúziu a difúziu (zmätenie a rozptýlenie). Vo finálnom kole je vynechaná operácia "Násobenie maticou" a to z dôvodu ľahkej inverzie procesu dešifrovania. Proces šifrovania je možné zhrnúť do nasledujúcich bodov [34]:

- Expanzia kľúča
- Počiatočná fáza
  - Pripočítanie šifrovacieho kľúča
- Kolá
  - Nelineárna bajtová substitúcia
  - Rotácia riadkov
  - Násobenie maticou
  - Pripočítanie kľúča daného kola
- Finálne kolo
  - Nelineárna bajtová substitúcia
  - Rotácia riadkov
  - Pripočítanie kľúča daného kola



Obr. 4.2: Štruktúra šifrovania a dešifrovania algoritmom AES(128 bitový kľúč, 10 kôl)



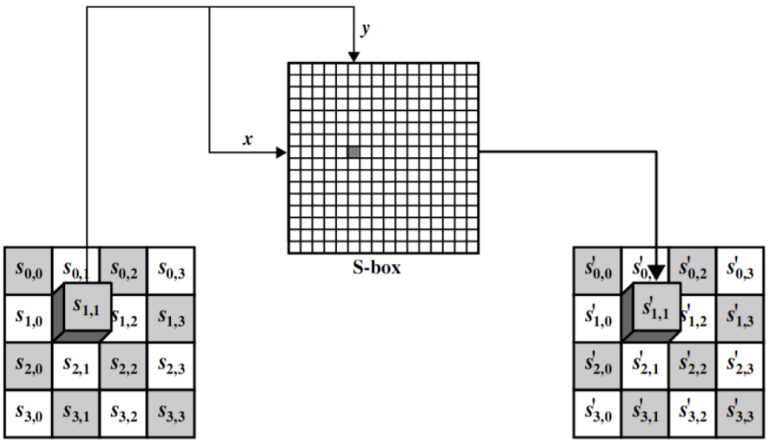
Obr. 4.3: Expanzia kľúča [34]

Expanzia kľúča

Algoritmus AES je založený na expanzii (rozšírení) kľúča AES na šifrovanie a dešifrovanie údajov. Každé kolo má nový kľúč. Rutina na expanziu kľúča vytvára **podkľúče** pre každé kolo slovo po slove, kde slovo je pole štyroch bajtov. Rutina vytvára $4 * (N_r + 1)$ slov. Kde N_r je celkový počet kôl [43]. Postup pre štandard AES-128 je zobrazený na obrázku 4.3 a je nasledovný. Šifrovací kľúč(počiatočný kľúč) sa používa na vytvorenie prvých štyroch slov. Veľkosť kľúča je 16 bajtov. Prvé štyri bajty sú reprezentované ako w_0, ďalšie štyri bajty v druhom stĺpci predstavuje w_1, nasledujúce štyri bajty vo w_2 a vo štvrtom stĺpci w_3 posledné štyri bajty.

Z obrázku je zrejmé, že kľúče w_5, w_6 a w_7, zapísané vo forme štyroch bajtových slov, sú generované jednoducho pomocou operácie XOR. Kľúč w_4 je vytvorený ďaleko zložitejšie za pomoci niekoľkých funkcií. Tieto funkcie sú aplikované na slovo w_3 šifrovacieho kľúča. Vykonávané funkcie sú nasledovné [34]:

- Cyklický posun bajtov slova w_3 o jednu pozíciu doprava.
- Substitúcia posunutých bajtov podľa príslušnej tabuľky pre S-box.
- Výsledok je sčítaný so slovom w_0 operáciou XOR.
- Výsledok predchádzajúcich troch krokov je sčítaný logickou operáciou XOR s konštantou **Rcon**. Táto konštanta je definovaná pre každé kolo. Každá hodnota konštanty



Obr. 4.4: Nelineárna bajtová substitúcia [1]

EA	04	65	85
83	45	5D	96
5C	33	98	B0
F0	2D	AD	C5

87	F2	4D	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6

Obr. 4.5: Reálny príklad nelineárnej bajtovej substitúcie v algoritme AES

Rcon je reprezentovaná štyrmi bajtmi. Tri najnižšie bajty sú vždy nulové. Na obrázku 4.3 sú uvedené hodnoty konštanty **Rcon** pre 10 kôl.

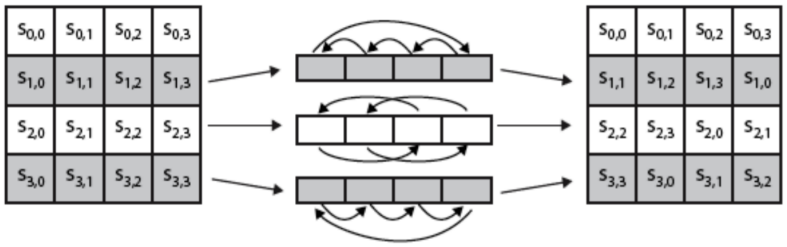
Na obrázku 4.3 je znázornená expanzia pre kľúč prvého kola. Ďalšie kľúče sú expandované identicky. Čiže vždy sa generujú 4 slová, kde prvé sa generuje zložitejšou cestou zahŕňajúcu posun, substitúciu, konštantu Rcon a operáciu XOR. V ostatných slovách sa používa iba operácia XOR.

Nelineárna bajtová substitúcia

Prvá fáza každého kola začína nelineárnou bajtovou substitúciou. Táto fáza závisí od substitučnej **S-box tabuľky**(Tabuľka B.1), podľa ktorej sa postupne nahradí každý bajt `stavu`. Napríklad v AES, ak máme hexa 53 v `stave`, treba ho vymeniť za hexa ED. ED je vytvorený z priesečníka 5 a 3 [1].

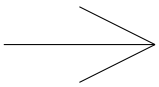
Rotácia riadkov

Hlavnou myšlienkou tohto kroku je cyklicky posúvať bajty stavu doľava v každom riadu okrem prvého riadku. Bajty v prvom riadku sa nemenia. Druhý riadok sa posúva kruhovo doľava o jeden bajt. Tretí riadok je cyklicky posunutý o dva bajty doľava. Posledný riadok je cyklicky posunutý o tri bajty doľava. Veľkosť nového stavu sa nemení a zostáva rovnaká ako pôvodná veľkosť 16 bajtov, avšak posunula sa pozícia bajtov, ako je znázornené na obrázku 4.6.



Obr. 4.6: Rotácia riadkov [1]

87	F2	4D	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6



87	F2	4D	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95

Obr. 4.7: Reálny príklad rotácii riadkov v algoritme AES

Násobenie maticou

Ďalším zásadným krokom je premiešanie stĺpcov pomocou násobenia maticou. Násobenie sa vykonáva v každom stave. Každý bajt jedného riadku transformačnej matici je vynásobený každou hodnotou (bajtom) v stĺpci matice `stavu`. Inak povedané, použije sa bežné **násobenie dvoch matíc** o veľkosti 4x4 s tým rozdielom, že namiesto sčítania vynásobených hodnôt, použijeme operáciu **XOR** [1]. Maticová reprezentácia výpočtu je nasledovná:

$$\begin{bmatrix} S'_{0,0} & S'_{0,1} & S'_{0,2} & S'_{0,3} \\ S'_{1,0} & S'_{1,1} & S'_{1,2} & S'_{1,3} \\ S'_{2,0} & S'_{2,1} & S'_{2,2} & S'_{2,3} \\ S'_{3,0} & S'_{3,1} & S'_{3,2} & S'_{3,3} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,0} & S_{1,1} & S_{1,2} & S_{1,3} \\ S_{2,0} & S_{2,1} & S_{2,2} & S_{2,3} \\ S_{3,0} & S_{3,1} & S_{3,2} & S_{3,3} \end{bmatrix}$$

Z toho vyplýva, že napríklad hodnota $S'_{0,0}$ je vypočítaná ako:

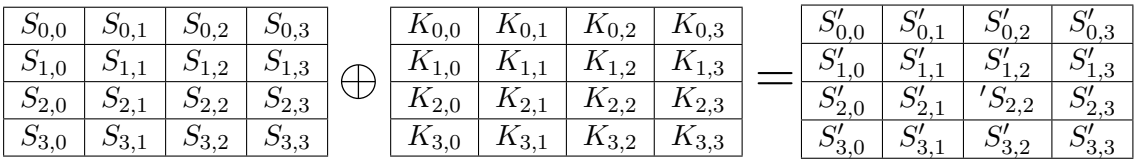
$$S'_{0,0} = (\{02\} \bullet S_{0,0}) \oplus (\{03\} \bullet S_{1,0}) \oplus S_{2,0} \oplus S_{3,0}$$

Pripočítanie kľúča daného kola

Pripočítanie kľúča daného kola je **najdôležitejšou** fázou v AES algoritmu. Kľúč aj vstupné údaje (tiež označované ako `stav`) sú usporiadané v 4x4 matici bajtov. Tento krok má schopnosť poskytnúť oveľa väčšiu bezpečnosť pri šifrovaní údajov, pretože táto operácia je založená na vytváraní vzťahu medzi kľúčom a šifrovým textom. Šifrovaný text pochádza z predchádzajúcej fázy. V tejto fáze sa tiež používa podkľúč a kombinuje so `stavom`. Hlavný kľúč sa používa na odvodenie podkľúča v každom kole pomocou Rijndaelovho plánu kľúčov. Veľkosť podkľúča a `stavu` je rovnaká. Celá fáza pripočítanie kľúča daného kola spočíva v tom, že každý bajt podkľúču je kombinovaný s každým bajtom `stavu` s použitím bitovej operácie XOR [32].

Čiže napríklad hodnota $S'_{0,0}$ v novo vzniknutom `stave` S' bude vypočítaná ako:

$$S'_{0,0} = S_{0,0} \oplus K_{0,0}$$



Obr. 4.8: Pripočítanie kľúča daného kola

4.2.2 Dešifrovanie

Postup dešifrovania je odvodený od šifrovania a je zobrazený na pravej polovici obrázka 4.2. Základným rozdielom je inverzia niektorých operácií a zmena poradia jednotlivých operácií. Použitý je dešifrovací kľúč, ktorý je totožný so šifrovacím, ale je vyčítaný v opačnom poradí. Inverzná schéma dešifrovania je nasledujúca [34]:

- Expanzia kľúča
- Počiatočná fáza
  - Pripočítanie šifrovacieho kľúča
- Kolá
  - Inverzná rotácia riadkov
  - Inverzná nelineárna bajtová substitúcia
  - Pripočítanie kľúča daného kola
  - Inverzné násobenie maticou
- Finálne kolo
  - Inverzná rotácia riadkov
  - Inverzná nelineárna bajtová substitúcia
  - Pripočítanie kľúča daného kola

Popis jednotlivých funkcií je v podstate identický s popisom funkcií behom šifrovania s tým rozdielom, že operácie pracujú inverzne.

Kapitola 5

Diferenciálne chybové útoky

Ako už popisuje kapitola 2, vonkajší šum, ako je elektromagnetické žiarenie, kolísanie napájacieho napätia alebo frekvencie môže spôsobiť chybu v elektronických zariadeniach. Tieto vlastnosti elektronických zariadení využívajú útočníci na úmyselné vloženie chyby do zariadenia, na ktorom beží kryptografický algoritmus. Potom pomocou analýzy chybného výstupu môže útočník odhaliť **tajný kľúč**. Tento typ útoku je známy ako diferenciálny chybový útok, ktorý bol pôvodne predstavený v útoku **Bellcore** [15]. Ich útok je založený na algebraických vlastnostiach modulárnej aritmetiky, a preto je použiteľný iba na kryptosystémy s verejným kľúčom, ako je napríklad RSA.

E. Biham a A.Shamir [13] rozšírili tento útok na rôzne kryptosystémy s tajným kľúčom ako DES a nazvali ho **diferenciálna chybová analýza** (DFA), ktorá je založená na kombinácii diferenciálnej kryptoanalýzy a analýzy chýb. Okrem toho aplikovali diferenciálu kryptoanalýzu na šifrovanie Data Encryption Standard (DES).

Diferenciálna chybová analýza sa stala najbežnejšie používanou metódou analýzy chýb pri útokoch na symetrické blokové šifry. Táto metóda sa volí ako prvá, pokiaľ ide o testovanie odolnosti voči chybám v nových kryptografických algoritmoch z dôvodu jej jednoduchosti a schopnosti obnovy tajného kľúča s nízkym počtom chybných šifrovaní [16].

Útok pomocou DFA pozostáva z injektovnia chyby do **prechodného stavu** šifry, zvyčajne počas jedného z posledných kôl. Chyba sa potom ďalej **šíri**, čo vedie k tomu, že zašifrovaný text je chybný. Následne sa analyzuje rozdiel medzi pôvodným a chybným zašifrovaným textom, čo poskytne útočníkovi informácie o tajnom kľúči. DFA využíva vlastností nelineárnej operácie, ktorá sa bežne používa v kryptosystémoch [16].

Ukázalo sa, že väčšina blokových kryptosystémov je zraniteľná voči DFA a neexistujú žiadne šifry, ktoré by v súčasnosti mohli zabrániť tejto analýze. Pomocou DFA sa podarilo uskutočniť úspešný útok na šifry ako AES [2], DES [13], PRESENT [9].

5.1 Útok Bellcore

Útok je použiteľný ako pre klasické šifrovanie RSA, čo zahŕňa jedno modulárne umocňovanie, tak aj pre **RSA-CRT**, čo pre zvýšenie efektivity používa čínsku vetu o zvyškoch(viac v sekcii 4.1.5). V prvom prípade útok vyžaduje niekoľko chybných podpisov, zatiaľ čo v druhom prípade postačí iba **jeden** chybný podpis. V tomto prípade je popísaná varianta RSA-CRT.

Tento útok je založený na niekoľko predpokladoch [29]:

- RSA implementácia využíva čínskej vety o zvyškoch

- Útočník môže zaviesť chybu buď v s_p alebo v s_q
- Útočník môže zaznamenať chybný podpis
- Útočník pozná správny podpis alebo počiatočnú správu m.

Predpokladajme, že konečný podpis s je vypočítaný Gaussovou rekombinačnou metódou s použitím súkromných kľúčov p, q:

$$s = CRT(s_p, s_q) = (s_p q(q^{-1} \mod p)) + (s_q p(p^{-1} \mod q)) \mod n$$

Predpokladajme, že chyba sa objavila vo výpočte s_p a chybný podpis s' je:

$$s' = CRT(s'_p, s_q) = (s'_q(q^{-1} \mod p)) + (s_q p(p^{-1} \mod q)) \mod n$$

Rozdiel medzi podpismi môže byť vypočítaný ako:

$$\begin{aligned}
\Delta &= s - s' \\
&= (s_p q(q^{-1} \mod p)) + (s_q p(p^{-1} \mod q)) - (s'_q(q^{-1} \mod p)) - (s_q p(p^{-1} \mod q)) \\
&= (s_p q(q^{-1} \mod p)) - (s'_q(q^{-1} \mod p)) \\
&= (s_p - s'_q)q(q^{-1} \mod p) \mod n
\end{aligned}$$

Preto najväčší spoločný deliteľ (NSD) medzi Δ a n je rovný q:

$$NSD(m, \Delta) = NSD(pq, (s_p - s'_p)q(q^{-1} \mod p)) = q$$

Arjen Lenstra spozoroval, že útok proti CRT-RSA možno vykonať s počiatočnou správou m [33]. Ak je chyba vyvolaná počas výpočtu s_p, potom pre správny podpis s a chybný s' platí nasledujúci vzťah:

$$\begin{aligned}
s^e &= (s')^e \mod q \\
s^e &\neq (s')^e \mod p
\end{aligned}$$

kde e je verejný exponent
Rozdiel $(s')^e - m$ môže byť získaný ako:

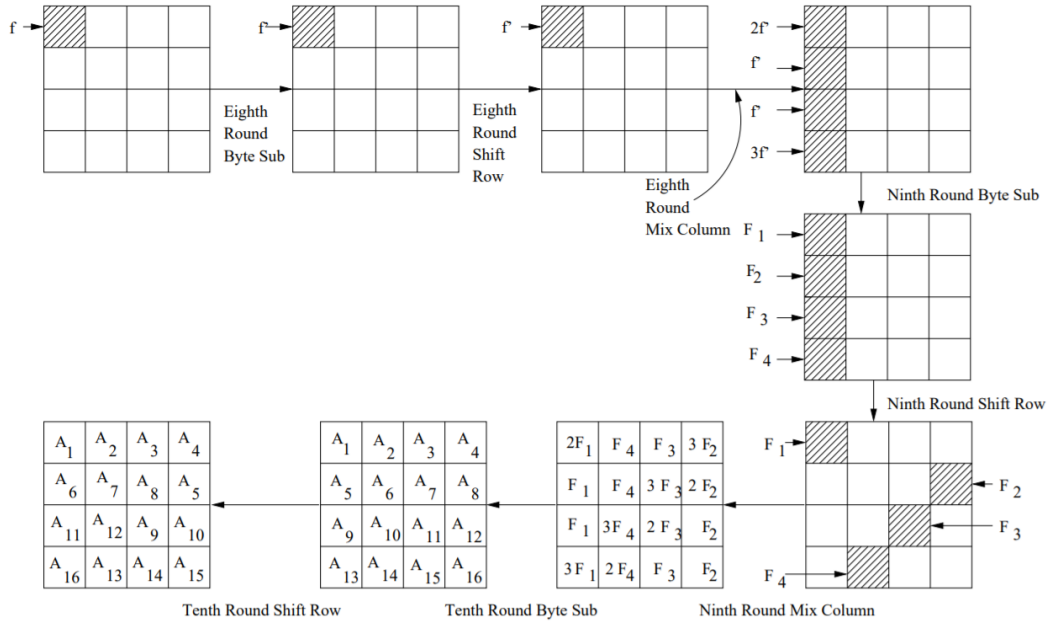
$$\begin{aligned}
(s')^e - m &= (s'_p)^e q(q^{-1} \mod p) + (s'_q)^e p(p^{-1} \mod q) - m \\
&= (s'_p)^e q(q^{-1} \mod p) + (s'_q)^e p(p^{-1} \mod q) - \\
&\quad - (s_p)^e q(q^{-1} \mod p) - (s_q)^e p(p^{-1} \mod q) \\
&= (s'_p)^e q(q^{-1} \mod p) - (s_p)^e q(q^{-1} \mod p) \\
&= ((s'_p)^e - (s_p)^e)q(q^{-1} \mod p)
\end{aligned}$$

Potom NSD pre modulus n a rozdiel $(s')^e - m$ je rovný q:

$$NSD(n, (s')^e - m) = NSD(pq, ((s'_p)^e - (s_p)^e)q(q^{-1} \mod p)) = q$$

5.2 Diferenciálna chybová analýza použitá na šifrovanie AES

V tejto časti bude definovaná stratégia vykonania analýzy chýb. Predpokladá sa, že útočník vyvolá chybu v bajte vstupu do **ôsmeho kola**. Tiež sa predpokladá že chyba zodpovedá náhodnému modelu chyby popísanému v sekcii 2.3.3, kde sa tento bajt stáva náhodnou a neznámou hodnotou. Popísaná technika diferenciálnej chybovej analýzy na AES je prevzatá od Tunstalla a spol. [53].



Obr. 5.1: Šírenie chyby vyvolanej v ôsmom kole šifry AES

5.2.1 Prvý krok útoku vyvolávajúceho chyby

Ak je v bajte stavovej matice vyvolaná chyba, ktorá je potom vstupom do ôsmeho kola, operácia **násobenie maticou** (MixColumn) na konci kola rozšíri túto chybu do celého stĺpca stavu. Operácia **rotácia riadkov** (ShiftRow) na začiatku nasledujúceho kola potom tieto bajty posunie, aby obsadili rôzne stĺpce. Ďalšia operácia **násobenie maticou** potom prenesie chybu do zostávajúcich dvanásť bajtov.

Tento proces je znázornený na obrázku 5.1, kde je ukázané rozšírenie chyby bajtu vyvolanej na vstup ôsmeho kola. Následne je zobrazený XOR rozdiel stavových matíc dvoch výsledkov, kde jeden je bez chyby a druhý chybný. Toto sa použije ako základ pre diferenciálnu chybovú analýzu.

Ak bude chyba vyvolaná vo vstupe do ôsmeho kola a zváži sa stav rozdielov po operácii **rotácia riadkov** deviateho kola, potom je možné získať nasledujúcu sadu rovníc, ktoré zahŕňajú hodnoty z kľúčových bajtov k_1, k_8, k_{11} a k_{14}, čím sa získa výraz pre 32 bitov $\mathbf{K_{10}}$.

$$
\begin{aligned}
2\delta_1 &= S^{-1}(x1 \oplus k1) \oplus S^{-1}(x'_1 \oplus k_1) \\
\delta_1 &= S^{-1}(x_{14} \oplus k_{14}) \oplus S^{-1}(x'_{14} \oplus k_{14}) \\
\delta_1 &= S^{-1}(x_{11} \oplus k_{11}) \oplus S^{-1}(x'_{11} \oplus k_{11}) \\
3\delta_1 &= S^{-1}(x_8 \oplus k_8) \oplus S^{-1}(x'_8 \oplus k_8)
\end{aligned}
$$

Kde δ_1, k_1, k_8, k_{11} a k_{14} sú neznáme hodnoty $\in \{0, ..., 255\}$

Vyššie uvedený systém rovníc možno použiť na zníženie možností pre tých 32 bitov kľúča. Útočník vyberie hodnotu pre δ_1 a určí, ktoré hodnoty k_1, k_8, k_{11} a k_{14} spĺňajú rovnice pomocou štyroch nezávislých dôkladných vyhľadávaní. Každá rovnica vráti 0, 2,

alebo 4 hypotézy. Ak niektorú zo štyroch rovníc nemožno splniť, potom môžu byť akékoľvek hypotézy pre túto hodnotu δ_1 zavrhnuté.

Rovnakú techniku je možné použiť na obnovenie informácií na zostávajúcich bajtoch posledného podkľúča. To znamená, že informácie na zostávajúcich kľúčových bajtoch možno odvodiť pomocou nasledujúcej sady rovníc. Na získanie informácií o k_2, k_5, k_{12} a k_{15} útočník môže použiť:

$$\begin{aligned} 3\delta_2 &= S^{-1}(x_5 \oplus k_5) \oplus S^{-1}(x'_5 \oplus k_5) \\ 2\delta_2 &= S^{-1}(x_2 \oplus k_2) \oplus S^{-1}(x'_2 \oplus k_2) \\ \delta_2 &= S^{-1}(x_{15} \oplus k_{15}) \oplus S^{-1}(x'_{15} \oplus k_{15}) \\ \delta_2 &= S^{-1}(x_{12} \oplus k_{12}) \oplus S^{-1}(x'_{12} \oplus k_{12}) \end{aligned}$$

Na získanie informácií o k_3, k_6, k_9 a k_{16} môže útočník použiť nasledujúce rovnice:

$$\begin{aligned} \delta_3 &= S^{-1}(x_9 \oplus k_9) \oplus S^{-1}(x'_9 \oplus k_9) \\ 3\delta_3 &= S^{-1}(x_6 \oplus k_6) \oplus S^{-1}(x'_6 \oplus k_6) \\ 2\delta_3 &= S^{-1}(x_3 \oplus k_3) \oplus S^{-1}(x'_3 \oplus k_3) \\ \delta_3 &= S^{-1}(x_{16} \oplus k_{16}) \oplus S^{-1}(x'_{16} \oplus k_{16}) \end{aligned}$$

Nakoniec, na získanie informácií o k_4, k_7, k_{10} a k_{13} môže útočník použiť nasledujúce rovnice:

$$\begin{aligned} \delta_4 &= S^{-1}(x_{13} \oplus k_{13}) \oplus S^{-1}(x'_{13} \oplus k_{13}) \\ \delta_4 &= S^{-1}(x_{10} \oplus k_{10}) \oplus S^{-1}(x'_{10} \oplus k_{10}) \\ 3\delta_4 &= S^{-1}(x_7 \oplus k_7) \oplus S^{-1}(x'_7 \oplus k_7) \\ 2\delta_4 &= S^{-1}(x_4 \oplus k_4) \oplus S^{-1}(x'_4 \oplus k_4) \end{aligned}$$

Je vhodné poznamenať, že rovnice majú rovnakú štruktúru, a preto majú riešenia podobný charakter. Očakáva sa, že vyhodnotenie každej sady rovníc vráti 2^8 jedinečných hypotéz pre príslušné kľúčové bajty. Preto útočník očakáva, že bude mať 2^{32} kľúčových hypotéz použitého tajného kľúča.

5.2.2 Analýza prvého kroku útoku vyvolávajúceho chyby

Prvý krok útoku používa štyri sady rovníc na zmenšenie kľúčového priestoru AES. V tejto sekcii sa určí očakávaný počet kľúčových hypotéz, ktoré bude mať útočník v každej fáze útoku.

S cieľom analyzovať počet platných hypotéz v prvej fáze útoku, sa analyzuje prvá sada rovníc uvedená v časti 5.2.1, kde $\delta_1 \in \{1, ..., 255\}$. Ak je δ_1 rovná nule, potom by sa dalo povedať, že očakávaná chyba nebola injektovaná. Čiže ak je δ_1 nula, tak to znamená, že $x_1 = x'_1$ a všetkých 256 kľúčových hypotéz je možných. Ako prvé sa uvažuje, že prvá rovnica je v tejto sade:

$$2\delta_1 = S^{-1}(x1 \oplus k1) \oplus S^{-1}(x'_1 \oplus k_1)$$

Hodnoty x_1 a x'_1 sú známe zo správnych a chybných šifrových textov. Pre danú hodnotu $2\delta_1$ bude 0, 2 alebo 4 platné kľúčové hypotézy. Pravdepodobnosť hypotézy pre všetky $\delta_1 \in \{1, ..., 255\}$ je približne jedna, a teda vznikne 256 kľúčových hypotéz, keď sú zvážené všetky možné hodnoty $\delta_1 \in \{1, ..., 255\}$.

To isté možno povedať o každej zo štyroch rovníc v sade uvedenej vyššie. Avšak za danú hodnotu δ_1 sa očakáva, že každá zo štyroch rovníc poskytne približne jednu hypotézu pre kľúčový bajt. Tieto hodnoty poskytnú jednu hypotézu pre štvoricu kľúčových bajtov $\{k_1, k_8, k_{11}, k_{14}\}$. Vzhľadom na to, že útočník bude musieť vziať do úvahy všetky hodnoty v množine $\{1, ..., 255\}$ vznikne 256 možných hodnôt pre štvorice $\{k_1, k_8, k_{11}, k_{14}\}$. Potom, čo útočník analyzoval štyri rovnice definované v časti 5.2.1 očakáva 2^{32} kľúčových hypotéz.

5.2.3 Druhý krok útoku vyvolávajúceho chyby

Pre ešte väčšie zredukovanie kľúčových hypotéz, sa použije vzťah medzi kľúčom z deviateho kola a kľúčom z desiateho kola.

Do úvahy berieme AES algoritmus plánovania kľúča, kľúč deviateho kola, $\mathbf{K_9}$, generovanie kľúča desiateho kola, $\mathbf{K_{10}}$. Plán kľúča je invertovateľný a $\mathbf{K_9}$ možno vyjadriť v členoch prvkov $\mathbf{K_{10}}$ Hodnota $\mathbf{K_9}$ môže byť vyjadrená ako:

$$\begin{pmatrix} k_1 \oplus S(k_{14} \oplus k_{10}) \oplus h_{10} & k_5 \oplus k_1 & k_9 \oplus k_5 & k_{13} \oplus k_9 \\ k_2 \oplus S(k_{15} \oplus k_{11}) & k_6 \oplus k_2 & k_{10} \oplus k_6 & k_{14} \oplus k_{10} \\ k_3 \oplus S(k_{16} \oplus k_{12}) & k_7 \oplus k_3 & k_{11} \oplus k_7 & k_{15} \oplus k_{11} \\ k_4 \oplus S(k_{13} \oplus k_9) & k_8 \oplus k_4 & k_{12} \oplus k_8 & k_{16} \oplus k_{12} \end{pmatrix}$$

Môžeme spozorovať, že chybné hodnoty v prvom stĺpci stavovej matice na výstupe v ôsmom kole operácie **násobenie maticou** sú $(2f', f', f', 3f')$, kde f' je nenulová ľubovoľná hodnota v $\mathbb{F}_{2^8}$. Pomocou operácie **inverzné násobenie maticou**(InverseMixColumn) a pomocou vzájomných vzťahov medzi chybnými hodnotami je možné definovať nasledujúcu rovnicu:

$$\begin{aligned} 2f' = {} & S^{-1}(14(S^{-1}(x_1 \oplus k_1) \oplus k'_1) \oplus 11(S^{-1}(x_{14} \oplus k_{14}) \oplus k'_2) \oplus \\ & 13(S^{-1}(x_{11} \oplus k_{11}) \oplus k'_3) \oplus 9(S^{-1}(x_8 \oplus k_8) \oplus k'_4)) \oplus \\ & S^{-1}(14(S^{-1}(x'_1 \oplus k_1) \oplus k'_1) \oplus 11(S^{-1}(x'_{14} \oplus k_{14}) \oplus k'_2) \oplus \\ & 13(S^{-1}(x'_{11} \oplus k_{11}) \oplus k'_3) \oplus 9(S^{-1}(x'_8 \oplus k_8) \oplus k'_4)) \\ = {} & S^{-1}(14(S^{-1}(x_1 \oplus k_1) \oplus ((k_1 \oplus S(k_{14} \oplus k_{10}) \oplus h_{10}))) \oplus \\ & 11(S^{-1}(x_{14} \oplus k_{14}) \oplus (k_2 \oplus S(k_{15} \oplus k_{11}))) \oplus \\ & 13(S^{-1}(x_{11} \oplus k_{11}) \oplus (k_3 \oplus S(k_{16} \oplus k_{12}))) \oplus \\ & 9(S^{-1}(x_8 \oplus k_8) \oplus (k_4 \oplus S(k_{13} \oplus k_9)))) \oplus \\ & S^{-1}(14(S^{-1}(x'_1 \oplus k_1) \oplus ((k_1 \oplus S(k_{14} \oplus k_{10}) \oplus h_{10}))) \oplus \\ & 11(S^{-1}(x'_{14} \oplus k_{14}) \oplus (k_2 \oplus S(k_{15} \oplus k_{11}))) \oplus \\ & 13(S^{-1}(x'_{11} \oplus k_{11}) \oplus (k_3 \oplus S(k_{16} \oplus k_{12}))) \oplus \\ & 9(S^{-1}(x'_8 \oplus k_8) \oplus (k_4 \oplus S(k_{13} \oplus k_9)))) \end{aligned}$$

Podobne je možné zadefinovať nasledujúce rovnice:

$$\begin{aligned} f' = S^{-1}(9(S^{-1}(x_{13} \oplus k_{13}) \oplus (k_4 \oplus k_9)) \oplus 14(S^{-1}(x_{10} \oplus k_{10}) \oplus (k_{10} \oplus k_{14})))\oplus \\ 11(S^{-1}(x_7 \oplus k_7) \oplus (k_{15} \oplus k_{11})) \oplus 13(S^{-1}(x_4 \oplus k_4) \oplus (k_{16} \oplus k_{12})))\oplus \\ S^{-1}(9(S^{-1}(x'_{13} \oplus k_{13}) \oplus (k_4 \oplus k_9)) \oplus 14(S^{-1}(x'_{10} \oplus k_{10}) \oplus (k_{10} \oplus k_{14})))\oplus \\ 11(S^{-1}(x'_7 \oplus k_7) \oplus (k_{15} \oplus k_{11})) \oplus 13(S^{-1}(x'_4 \oplus k_4) \oplus (k_{16} \oplus k_{12}))) \end{aligned}$$

$$\begin{aligned} f' = S^{-1}(13(S^{-1}(x_9 \oplus k_9) \oplus (k_9 \oplus k_5)) \oplus 9(S^{-1}(x_6 \oplus k_6) \oplus (k_{10} \oplus k_6)))\oplus \\ 14(S^{-1}(x_3 \oplus k_3) \oplus (k_{11} \oplus k_7)) \oplus 11(S^{-1}(x_{16} \oplus k_{16}) \oplus (k_{12} \oplus k_8)))\oplus \\ S^{-1}(13(S^{-1}(x'_9 \oplus k_9) \oplus (k_9 \oplus k_5)) \oplus 9(S^{-1}(x'_6 \oplus k_6) \oplus (k_{10} \oplus k_6)))\oplus \\ 14(S^{-1}(x'_3 \oplus k_3) \oplus (k_{11} \oplus k_7)) \oplus 11(S^{-1}(x'_{16} \oplus k_{16}) \oplus (k_{12} \oplus k_8))) \end{aligned}$$

$$\begin{aligned} 3f' = S^{-1}(11(S^{-1}(x_5 \oplus k_5) \oplus (k_5 \oplus k_1)) \oplus 13(S^{-1}(x_2 \oplus k_2) \oplus (k_6 \oplus k_2)))\oplus \\ 9(S^{-1}(x_{15} \oplus k_{15}) \oplus (k_7 \oplus k_3)) \oplus 8(S^{-1}(x_{12} \oplus k_{12}) \oplus (k_8 \oplus k_4)))\oplus \\ S^{-1}(11(S^{-1}(x'_5 \oplus k_5) \oplus (k_5 \oplus k_1)) \oplus 13(S^{-1}(x'_2 \oplus k_2) \oplus (k_6 \oplus k_2)))\oplus \\ 9(S^{-1}(x'_{15} \oplus k_{15}) \oplus (k_7 \oplus k_3)) \oplus 8(S^{-1}(x'_{12} \oplus k_{12}) \oplus (k_8 \oplus k_4))) \end{aligned}$$

Druhá fáza útoku je spojená s prvou fázou a používa sa na ďalšie zníženie počtu kľúčových hypotéz.

5.2.4 Analýza druhého kroku útoku vyvolávajúceho chyby

Očakávaný počet hypotéz vytvorených v druhom kroku útoku vychádza z podobného zdôvodnenia ako analýza prvého kroku uvedeného v časti 5.2.2. Ak vezmeme do úvahy druhú rovnicu definovanú v časti 5.2.3, možno ju prepísať ako:

$$f' = A \oplus B$$

kde premenné A a B sú definované ako:

$$\begin{aligned} A \ = S^{-1}(9(S^{-1}(x_{13} \oplus k_{13}) \oplus (k_4 \oplus k_9)) \oplus 14(S^{-1}(x_{10} \oplus k_{10}) \oplus (k_{10} \oplus k_{14})))\oplus \\ 11(S^{-1}(x_7 \oplus k_7) \oplus (k_{15} \oplus k_{11})) \oplus 13(S^{-1}(x_4 \oplus k_4) \oplus (k_{16} \oplus k_{12}))) \\ B = S^{-1}(9(S^{-1}(x'_{13} \oplus k_{13}) \oplus (k_4 \oplus k_9)) \oplus 14(S^{-1}(x'_{10} \oplus k_{10}) \oplus (k_{10} \oplus k_{14})))\oplus \\ 11(S^{-1}(x'_7 \oplus k_7) \oplus (k_{15} \oplus k_{11})) \oplus 13(S^{-1}(x'_4 \oplus k_4) \oplus (k_{16} \oplus k_{12}))) \end{aligned}$$

A a B môžeme považovať za náhodné hodnoty v $\mathbb{F}_{2^8}$. Pre dané hodnoty f' rozdiel medzi A a B sa bude rovnať f' s pravdepodobnosťou $\frac{1}{2^8}$. Použitím rovnakého uvažovania pravdepodobnosť, že všetky štyri rovnice boli platné je $(\frac{1}{2^8})^4 = \frac{1}{2^{32}}$

Je potrebné uvažovať o všetkých možných hodnotách f', to je $\{0, ..., 255\}$. Daná kľúčová hypotéza bude preto platná pre niektorú ľubovoľnú hodnotu f' s pravdepodobnosťou $2^8 * \frac{1}{2^{32}} = \frac{1}{2^{24}}$. V prvom kroku útoku sa očakáva, že útok vráti 2^{32} hypotéz, pričom o každej sa stále uvažuje na konci druhého kroku s pravdepodobnosťou $\frac{1}{2^{24}}$. Dalo by sa preto očakávať, že druhý krok útoku vyprodukuje 2^8 možných kľúčových hypotéz.

5.2.5 Útok na iné bajty

V predchádzajúcich častiach je popísaný útok, kde je diferenciálna chybová analýza založená na poznaní, že v prvom bajte stavovej matice bola vyvolaná chyba. Avšak je možné poznamenať, že analýza vracia veľmi malý počet hypotéz. Preto je možné uskutočniť 16 nezávislých analýz za predpokladu, že chyba je vyvolaná vo všetkých 16 bajtoch stavu na začiatku ôsmeho kola. Útočník môže očakávať, že to vytvorí $2^4 * 2^8 = 2^{12}$ platných kľúčových hypotéz, čo je stále triviálne vyhľadávanie.

Kapitola 6

Teória útokov využívajúcich zmenu napätia

Táto kapitola popisuje princíp najznámejších útokov, ktoré využívajú zmenu napätia. Okrem rozdelenia útokov na hardvérové a softvérové (viď kapitola 2), sa tieto útoky delia podľa toho, na akú **architektúru** procesorov boli použité a to buď Intel(x86) alebo ARM. Útoky nie sú prenosné z nasledujúcich dôvodov [27]:

- Architektúra ARM umožňuje nastavenie napätie jadra a frekvencie prakticky **bez obmedzenia**. To znamená, že útočník si môže ľubovoľne vybrať kombináciu frekvencie a napätia, čo umožňuje použiť extrémne nebezpečné nastavenia napätia/frekvencie výlučne zo softvéru na vykonanie útoku. Naopak architektúra x86 ponúka iba pevný, preddefinovaný zoznam P stavov, ktoré sú testované, aby boli bezpečné pre bežné prevádzkové podmienky výrobcu pred uvedením na trh.
- Architektúra ARM umožňuje, efektívne fungovanie každého jadra vo svojom **vlastnom** stave P. Na x86 fungujú všetky fyzické jadrá v rámci toho istého stavu P, čo znamená, že rovnaké nastavenie napätia sa bude vzťahovať na útočníka aj jadro obete a preto chyby nemožno jednoducho obmedziť na dané jadro ako na architektúre ARM.
- Platformy založené na x86 architektúre využívajú rozsiahle **bezpečnostné merania** a implementujú **obranu**, aby bolo možné detekovať, zabrániť a zotaviť sa z chýb hardvéru za behu. Takéto záchranné mechanizmy prakticky neexistujú na ARM, čo výrazne zvyšuje spoľahlivosť zavedenia chyby a možné scenáre zneužitia.
- Keďže správa napájania je jedným z kľúčových faktorov na mobilných zariadeniach (ARM) príslušné hardvérové mechanizmy sú rozsiahle **zdokumentované** a nástroje sú **ľahko dostupné**. Pre architektúru x86 prakticky **neexistuje** žiadna oficiálna dokumentácia týkajúca sa nízkoúrovňovej správy napájania, čiže aj jednoduché testy zvyčajne zahŕňajú aj nákladné **reverzné inžinierstvo** mikroarchitektonických prvkov, ktoré môžu byť odlišné na inej generácii procesorov.

6.1 PlunderVolt

Všetky informácie o útoku PlunderVolt sú získané zo štúdie tohto útoku [40].

Moderné CPU sú veľmi dobre optimalizované, aby výkon a účinnosť boli maximalizované pri zachovaní správnej funkčnosti pre špecifikované pracovné podmienky. V skutočnosti

Názov útoku	Typ útoku	Architektúra
PlunderVolt	Sóftvérový	Intel
VoltPillager	Hardvérový	Intel
TrustZone-M(eh)	Hardvérový	ARM

Tabuľka 6.1: Známe útoky využívajúce zmenu napätia a ich základné vlastnosti

väčšina moderných procesorov nemôže bežať natrvalo na ich maximálnu frekvenciu, pretože by sa spotrebovalo veľké množstvo energie, čo produkuje príliš veľa tepla. Preto procesory udržujú taktovaciu frekvenciu a napájacie napätie čo najnižšie – **dynamicky** sa zväčšujú iba v prípade potreby. Vyššie frekvencie vyžadujú pre správnu funkciu procesora vyššie napätie, preto by sa nemali meniť nezávisle. Okrem toho existujú aj iné typy spotreby energie, čo ovplyvňuje vhodný výber dvojice frekvencia napätie pre špecifické situácie.

Zníženie napájacieho napätia bolo tiež dôležité vo vývoji najnovších DRAM. Napájacie napätie sa postupne znižuje, čo má za následok menší náboj v kondenzátoroch uchovávajúcich jednotlivé bity – toto viedlo k známemu efektu **Rowhammer** [28]. Veľké množstvo výskumov na daný efekt, prinieslo množstvo praktických útokov napríklad pre eskaláciu privilégií, zavádzanie chýb do kryptografických primitív, alebo čítanie nedostupných pamäťových miest. Preto vedecká komunita a priemysel vynaložil značné úsilie na odstránenie efektu Rowhammer. Dosiahlo to bod, kde Intel nakoniec považuje hlavnú pamäť ako **nedôveryhodné úložisko** a plne **šifruje** a overuje všetku pamäť v rámci enklávy Intel SGX [23].

Pri útoku Plundervolt, útočník zneužije privilegovaný softvér a pomocou nezdokumentovaného rozhrania Intelu na škálovanie napätia sa naruší integrita výpočtov v enkláve Intel SGX. Plundervolt starostlivo **kontroluje napájacie napätie** procesora počas výpočtu enklávy, čo spôsobuje predvídateľné chyby procesora. V dôsledku toho ani technológia šifrovania/overenia pamäte Intel SGX nemôže chrániť pred útokom Plundervolt.

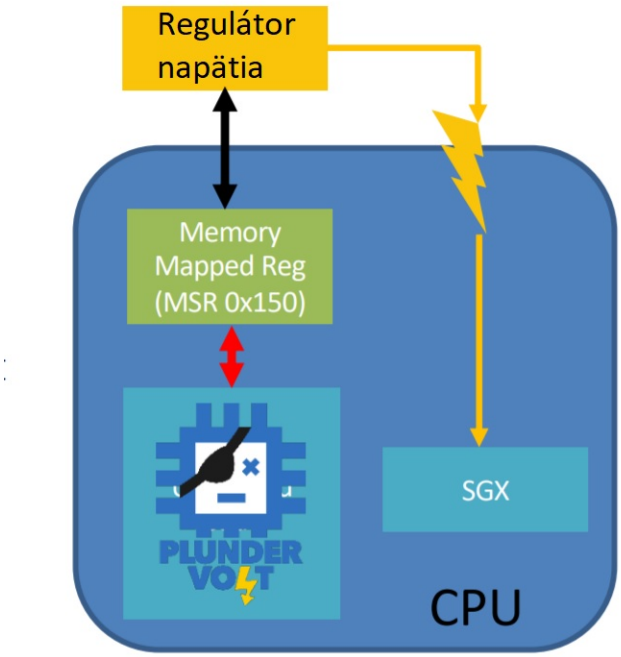
6.1.1 Testovacie nastavenia

Predpokladá sa štandardný Intel SGX model útočníka, kde útočník má plnú kontrolu nad všetkým softvérom bežiacim mimo enklávy (vrátane privilegovaného systémového softvéru, ako je operačný systém a BIOS). Rozhodujúca je schopnosť útočníka čítať a zápisovať do **registra MSR**, napríklad prostredníctvom škodlivého modulu jadra `ring 0` alebo útočného frameworku ako je `SGXStep`.

Škálovanie napätia na procesoroch Intel

Pomocou reverzného inžinierstva bola potvrdená existencia nezdokumentovaného registra MSR, ktorý sa používa na úpravu prevádzkového napätia na procesoroch Intel.

Obrázok 6.2 ukazuje, ako môže byť 64-bitová hodnota v registri **MSR 0x150** rozdelená na index a **napäťový ofset**. Špecifikovaním platného indexu si systémový softvér môže vybrať, na ktoré komponenty CPU by mala byť zmena napätia aplikovaná. Jadro CPU a vyrovnávacia pamäť **zdieľajú** rovnaké napätie a vyššie napätie bude použité na jadro aj vyrovnávaciu pamäť. Požadovaný napäťový ofset je zakódovaný ako 11-bitové celé číslo so znamienkom vzťahujúce sa na základné prevádzkové napätie jadra. Táto hodnota je vyjadrená v jednotkách $1/1024V$ (približne $1mV$), čím umožňuje maximálnu zmenu napätia o $1V$. Potom, čo softvér úspešne odoslal požiadavku na škálovanie napätia, trvá určitý čas,



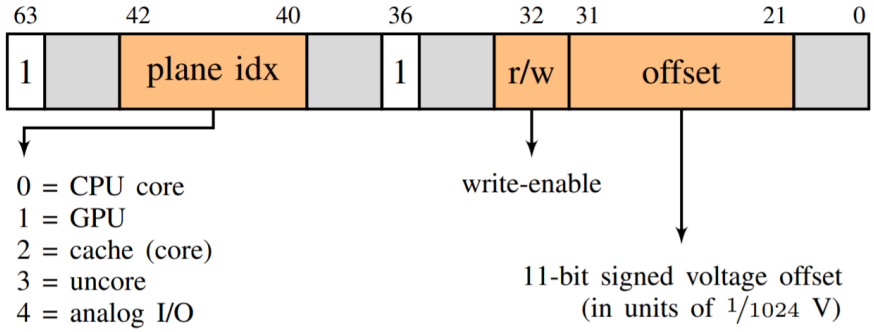
Obr. 6.1: Injektovanie chyby do procesora Intel s podporou SGX pomocou útoku PlunderVolt

kým dôjde k fyzickej zmenej napätia. Aktuálne prevádzkové napätie môže byť vyžiadané z dokumentovaného registra MSR 0x198.

Nastavenie napätia a frekvencie

Aby bol spoľahlivo nájdený **pár frekvencia/napätie** spôsobujúci chybu, je potrebné nakonfigurovať CPU tak, aby bežalo na **fixnej frekvencii**. Frekvencia sa nastaví napríklad pomocou príkazu *cpupower*. Podvoltovanie sa aplikuje zapísaním do registra MSR 0x150 (napríklad pomocou modulu msr jadra Linuxu) presne pred vstupom obete do enklávy pomocou **ECALL** cez nedôveryhodný hostiteľský program. Po návrate z enklávy hostiteľský program okamžite naspäť nastaví stabilné prevádzkové napätie. Okrem modulu jadra msr sa môže útočník spoliehať aj na presnejšie metódy podvoltovania, napríklad ak by mala byť minimalizovaná latencia konfigurácie. Preto bol framework na kontrolu vykonávania enklávy SGX-Step rozšírený o funkcionality x86 prerušenie a volanie brány. Tým pádom je možné vykonávať privilegované pokyny na čítanie a zápis do registra MSR priamo pred vstupom obete do enklávy.

Jednou z výziev pre úspešný útok Plundervolt je nastavenie parametru podvoltovania tak, aby procesor generoval **nesprávne výsledky** pre určité inštrukcie, pričom stále umožňuje **správne fungovanie** ostatných inštrukcií. To znamená, že prílišné podvoltovanie vedie k zlyhaniu a **zamrznutiu** systému, zatiaľ čo príliš malé podvoltovanie **nevytvára** žiadne chyby. Preto je potrebné nájsť správnu hodnotu podvoltovania opatrným znižovaním napätia jadra(napríklad o 1 mV na krok) až kým nenastane porucha, ale zároveň sa nezrúti systém. V praxi výskumníci útoku PlunderVolt zistili, že stačí podvoltovanie na



Obr. 6.2: Štruktúra nezdokumentovaného registra MSR s adresou 0x150 [40]

krátke časové úseky o $-100mV$ až $-260mV$, v závislosti od konkrétneho CPU, frekvencie a teploty.

6.1.2 Vyvolanie chyby násobenia v enkláve

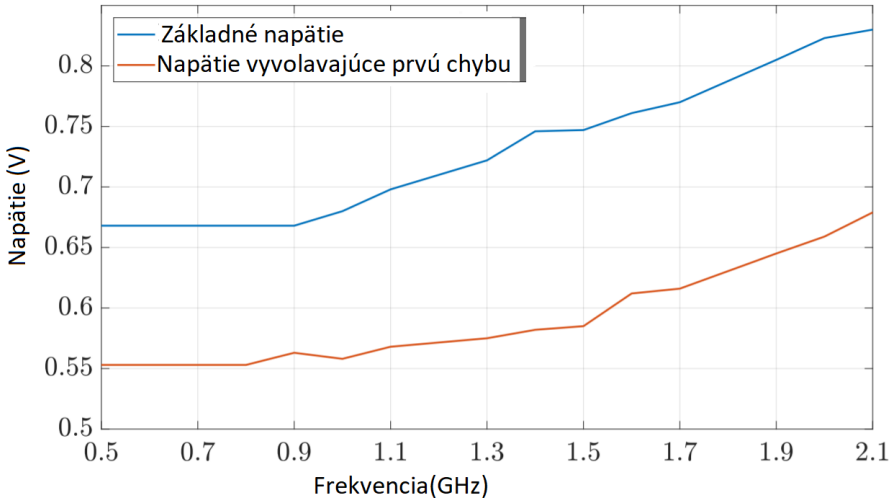
Prvým krokom k praktickému injektovaniu chýb do SGX enkláv, bolo analyzovať x86 inštrukcie v **izolácii**. Aj keď sa nepodarilo vyvolať chybu u jednoduchých aritmetických inštrukcií (napríklad sčítanie a odčítanie) alebo logických inštrukcií (napríklad posuny a OR/XOR/AND), pomocou **násobenia** sa chybu podarilo vyvolať. Môže to byť spôsobené tým, že násobičky majú zvyčajne dlhšiu kritickú cestu v porovnaní so sčítačkami alebo inými jednoduchými operáciami. Alebo dôvodom môže byť aj, že násobenie býva pravdepodobne najagresívnejšie optimalizované kvôli ich častému používaniu v zdrojových kódoch.

Zvážme nasledujúcu implementáciu, ktorá spúšťa jednoduché násobenie (daný kód sa skompiluje do jazyka symbolických inštrukcií s inštrukciou *imul*) v cykle vo vnútri obsluhy ECALL:

```
uint64_t multiplier = 0x1122334455667788;
uint64_t var = 0xdeadbeef * multiplier;
while(var == 0xdeadbeef * multiplier){
    var = 0xdeadbeef;
    var *= multiplier;
}
var ^= 0xdeadbeef * multiplier;
```

Výpis 6.1: Kód v jazyku C demonštrujúci chybu násobenia

Z kódu je očividné, že program by nemal **nikdy skončiť**, pretože podmienka cyklu je vždy splnená. Avšak experimenty ukázali, že podvoltovanie CPU presne pred vstupom do enklávy spôsobí prevrátenie bitov (viď 2.3.1) v premennej *var*, typicky v bajte 3 (počítajúc od najmenej významného bajtu ako bajt 0). To umožňuje ukončenie programu enklávy. Na chybný výstup je následne použitá logická operácia XOR s požadovanou hodnotou, aby sa zvýraznili iba chybné bit(y). V tejto špecifickej konfigurácii je výstup vždy $0x04000000$.



Obr. 6.3: Nameraný vzťah medzi frekvenciou, normálnym napätím (modrá) a potrebným podvolotovaním(oranžová) na dosiahnutie chybného výsledku násobenia vo vnútri enklávy SGX pre procesor i3-7100U-A [40]

Analýza efektu podvoltovania

Pomocou registra MSR 0x198 je možné zistiť aktuálne napätie v normálnom prevádzkovom režime a tiež ho zaznamenávať pri výpočte chybného výsledku. Merania v tomto registri nemusia byť absolútne presné, ale presne odrážajú relatívne podvoltovanie.

Po otestovaní rôznych hodnôt pre operandy násobenia, je možné chybné výsledky rozdeliť do dvoch kategórií:

- Prevrátenie jedného až päť (súvislých) bitov
- Prevrátenie všetkých najvýznamnejších bitov

6.1.3 Extrakcia kľúčov z enklávy pomocou injektovania chýb

Po preukázaní uskutočniteľnosti injektovania chýb do SGX enklávy v sekcii 6.1.2 sa aplikujú techniky podvoltovania na kryptografické knižnice používané v enklávach.

Extrakcia kľúča z dešifrovania/podpisu RSA-CRT v SGX pomocou IPP Crypto

API Intelu SGX-SDK *Tcrypto* odhaľuje iba obmedzený počet kryptografických primitív. Avšak, vývojár môže tiež priamo volať funkcie IPP Crypto keď sú potrebné ďalšie funkcie. Jednou funkciou, ktorá je dostupná prostredníctvom tohto API, je **dešifrovanie** alebo **generovanie podpisu** pomocou RSA s často používanou CRT (Chinese Remainder Theorem) optimalizáciou. V terminológii IPP Crypto, sa to označuje ako kľúče „typu 2“ inicializované pomocou funkcie `ippsRSA_InitPrivateKeyType2()`.

Operácie so súkromným kľúčom RSA-CRT (dešifrovanie a podpis) sú známe tým, že sú zraniteľné voči útokom **Bellcore** a **Lenstra** [15]. Tieto útoky vyžadujú chybu presne v jednom z dvoch umocnení jadra operácie RSA bez ďalších požiadaviek na povahu alebo lokalitu chyby.

Prvým krokom k praktickému uskutočneniu tohto útoku pre SGX bolo injektovanie chyby do funkcie *ippsRSA_Decrypt()* spustenej v rámci enklávy SGX počas celého trvania operácie RSA. To však malo za následok nevyužiteľné chyby, pravdepodobne preto, že obe čiastočné umocnenia boli chybné. Preto bolo zavedené druhé vlákno (v nedôveryhodnom kóde), ktoré po jednej tretine celkového trvania ECALL nastaví napätie na stabilnú hodnotu. S tým získané chyby by sa mohli použiť na získanie 2048bitového RSA modula pomocou útokov Lenstra a Bellcore. To znamená, že je možné obnoviť celý kľúč jediným chybným dešifrovaním alebo podpisom a zanedbateľným výpočetným úsilím. Presný postup ako obnoviť tento kľúč je popísaný v 5.1.

Diferenciálna chybová analýza v šifrovaní AES-NI v SGX

Sada inštrukcií pre procesory Intel poskytuje efektívnu hardvérovú implementáciu pre naplánovanie kľúča AES a výpočty kola. Napríklad na architektúre Skylake má AES inštrukcia na výpočet kola latenciu iba štyri hodinové cykly a priepustnosť jeden cyklus na inštrukciu[1]. AES-NI je veľmi často používaný v kryptografických knižniciach, vrátane SGX `Tcrypto` API, ktoré odhaľuje funkcie pre AES v režime počítania Galois (GCM), v normálnom režime počítadla a v konštrukcii CMAC. Tieto kryptografické primitíva sa potom používajú v rámci Intel SGX-SDK, vrátane kritických operácií, ako je zapečatenie a odpečatenie údajov o enkláve.

V experimentoch bolo dokázané, že šifrovacia inštrukcia *(v)aesenc* AES-NI na výpočet kola je **zraniteľná** voči útokom Plundervolt. Zaznamenané chyby boli vždy jedno prevrátenie bitov v druhom bajte zľava. Jedno prevrátenie bitov je ideálna chyba pre diferenciálnu chybovú analýzu(DFA). Príklad výstupu:

```
[Enclave] plaintext: 697DBA24B0885D4E120FFCAB82DDEC25
[Enclave] round key: F8BD0C43844E4B4F28A6D3539F3A73E5
[Enclave] ciphertext1: C9210B59333A07A922DE59788D7AA1A7
[Enclave] ciphertext2: C9230B59333A07A922DE59788D7AA1A7
[Enclave] plaintext: 4C96DD4E44B4278E6F49FCFC8FCFF5C9
[Enclave] round key: BE7ED6DB9171EBBF9EA51569425D6DDE
[Enclave] ciphertext1: 0D42753C23026D11884385F373EAC66C
[Enclave] ciphertext2: 0D40753C23026D11884385F373EAC66
```

Následne boli tieto jednokolové chyby použité na vytvorenie útoku na obnovenie kľúča proti celej šifre AES. Kanonická implementácia AES použitím inštrukcií AES-NI2 bola spustená v enkláve, samozrejme aj s podvoltovaním. Avšak pravdepodobnosť, že chyba zasiahne konkrétnu inštrukciu kola je približne 1/10, čo naznačuje rovnomerné rozloženie pravdepodobnosti na každom z desať kôl AES. Častým opakovaním operácie (v priemere 5-krát) sa podarilo dosiahnuť chybu v 8. kole. Príklad výstupu (pomocou kľúča $0x000102030405060708090a0b0c0d0e0f$) je nasledujúci:

```
[Enclave] plaintext: 5ABB97CCFE5081A4598A90E1CEF1BC39
[Enclave] CT1: DE49E9284A625F72DB87B4A559E814C4 <- chybný
[Enclave] CT2: BDFADCE3333976AD53BB1D718DFC4D5A <- správny
vstup do kola 10:
[Enclave] 1: CD58F457 A9F61565 2880132E 14C32401
```

[1] https://software.intel.com/sites/landingpage/IntrinsicsGuide/#expand=233&text=_mm_aesenc_si128

```
[Enclave] 2: AEEBC19C D0AD3CBA A0BCBAFA C0D77D9F
vstup do kola 9:
[Enclave] 1: 6F6356F9 26F8071F 9D90C6B2 E6884534
[Enclave] 2: 6F6356C7 26F8D01F 9DF7C6B2 A4884534
vstup do kola 8:
[Enclave] 1: 1C274B5B 2DFD8544 1D8AEAC0 643E70A1
[Enclave] 2: 1C274B5B 2DFD8544 1D8AEAC0 646670A
```

Pre pochopenie, aká chyba vznikla, sa zobrali aj správne aj chybné šifrové texty a následne boli dešifrované kolo za kolom pri porovnaní medzistavov. Výsledok je možné vidieť vo vyššie uvedenom výstupe. Za povšimnutie stojí bajt vyznačený na červeno v **8. kole**. Ten sa zmenil z $0x66$ na $0x3E$. Tento chybný bajt zavinila operácia XOR s $0x02$ (t.j. jednobitové preklopenie). Na základe chyby v 8. kole bolo možné použiť techniku **diferenciálnej analýzy chýb** od Tunstalla [53] a jej implementáciu od Jovanovica[2].

Pomocou tohto útoku je možné obnoviť pre daný pár správny a chybný šifrovaný text, celý 128-bitový kľúč AES s výpočtovou náročnosťou v priemere $2^32 + 256$ šifrovaní. V praxi extrahovanie celého kľúča AES z enklávy zabralo iba pár minút, vrátane injektovania chýb a výpočtu kľúča.

6.1.4 Reakcia Intelu

Intel po nahlásení útoku reagoval takmer okamžite a aktualizáciou BIOSu (CVE-2019-11157) **zakázal** zmenu napätia pomocou registra MSR. To znamená, že ak sa na počítači nachádza verzia BIOSu z konca roku 2019 a novšia, už nie je možné vykonať tento útok.

6.2 VoltPillager

Všetky informácie v tejto sekcii sú čerpané zo štúdie tohto útoku [18].

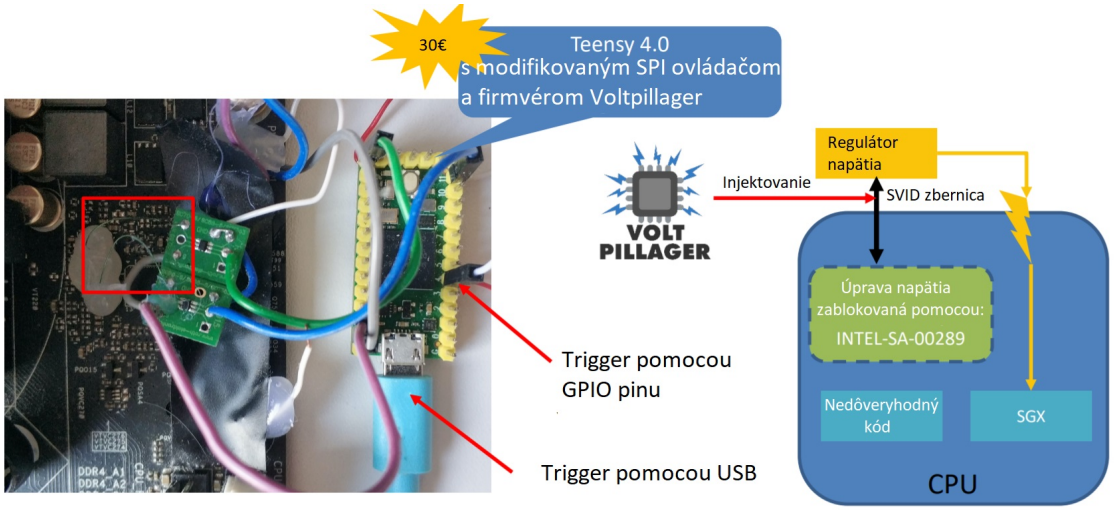
VoltPillager je **hardvérový útok** cielený na procesory s architektúrou Intel x86 s podporou SGX, ktorý využíva nízkonákladové vybavenie pre injektovanie správ na zbernicu **SVID** (Serial Voltage Identification) medzi CPU a regulátor napätia na základnej doske. To umožňuje presne kontrolovať napätie jadra CPU. S týmto vybavením sa úspešne podarilo vykonať útok na injektovanie chyby, ktorý narušuje **dôvernosť** a **integritu** enkláv Intel SGX. Voltpillager obsahuje aj `proof-of-concept` útoky na **obnovenie kľúča** proti kryptografickým algoritmom bežiacim vo vnútri SGX. Tento útok je dokonca nebezpečnejší ako softvérové útoky proti SGX (napríklad PlunderVolt), pretože funguje aj na opravených systémoch s najnovšími protiopatreniami, kde už softvérové útoky nie je možné vykonať. Ochrana pred útokom VoltPillager nie je jednoduchá a môže vyžadovať prehodnotenie modelu SGX, kde je poskytovateľ cloudu nedôveryhodný a útočník má fyzický prístup k hardvéru.

Avšak VoltPillager vyžaduje zrejmý **hardvérový prístup** k základnej doske systému a dôkladné pripojenie k regulátoru napätia, takže aj keď útok Plundervolt bol obmedzený potrebou prístupu root/admin v systéme, VoltPillager je oveľa obmedzenejší.

Cena všetkého potrebného vybavenia k útoku je približne 30€. Vybavenie sa skladá z:

- Dosky Teensy 4.0 Development Board[3], ktorá celý útok riadi

[2] https://github.com/Daeinar/dfa-aes
[3] https://www.pjrc.com/store/teensy40.html



Obr. 6.4: Injektovanie chyby do procesora Intel s podporou SGX pomocou útoku VoltPillager

- 2xSOT IC adaptér[4]
- 2xRiadič zbernice NL17SZ07XV5T2G[5]

6.2.1 Rozhrania na riadenie napätia CPU

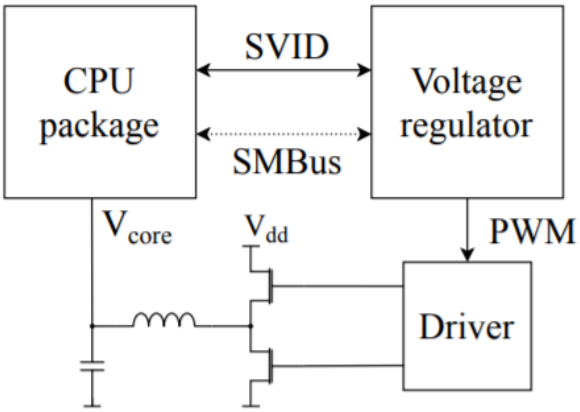
V moderných počítačoch je zvyčajne jeden alebo viac regulátorov napätia pripojených k CPU na základnej doske. Používajú sa na riadenie výkonu a spotrebu energie systému tým, že zmenia napätie jadra (a iné napätia) dodávané do CPU. Keď CPU beží na nižších frekvenciách alebo je v nečinnom režime, CPU posiela do regulátora napätia príkazy na zníženie napätia. Naopak, keď CPU pracuje pri veľkej záťaži a/alebo pri vysokej frekvencii, požaduje od regulátora napätia zvýšenie napätia. Na základnej doske boli nájdené 2 rozhrania k VR, ktoré možno použiť na zmenu napätia CPU, a teda pomocou nich uskutočniť podvoltovací útok:

- rozhranie **SVID**
- rozhranie **SMBus**

I keď injektovanie chýb pomocou zbernice SMBus je možné, zbernica SVID bola uprednostnená pre niekoľko dôvodov:

- vyššia frekvencia SVID, čo umožňuje presnejšie injektovanie chýb
- SVID príkazy sú rovnaké pre všetky testované systémy, na zbernici SMBus sa líšili podľa použitého regulátora napätia
- SVID používajú všetky moderné základné dosky a CPU, SMBus používajú iba konkrétne základné dosky a regulátory napätia

[4] https://www.farnell.com/datasheets/1934984.pdf
[5] https://www.farnell.com/datasheets/2007350.pdf



Obr. 6.5: Architektúra napájania na systémoch x86. CPU je prepojené s VR pomocou SVID a SMBus(voliteľné, bodkované) k VR. VR pomocou pulzne šírkovej modulácie (PWM) riadi výstup na generovanie potrebného napätia pre procesor

Zbernica SVID (Serial Voltage Identification)

SVID je rozhranie používané na odosielanie procesorom požadovaného napätia do externého regulátora napätia (VR). K tejto zbernici Intel neposkytuje žiadnu detailnú dokumentáciu, avšak z dokumentácie k CPU sa dá zistiť, že SVID používa 3 piny:

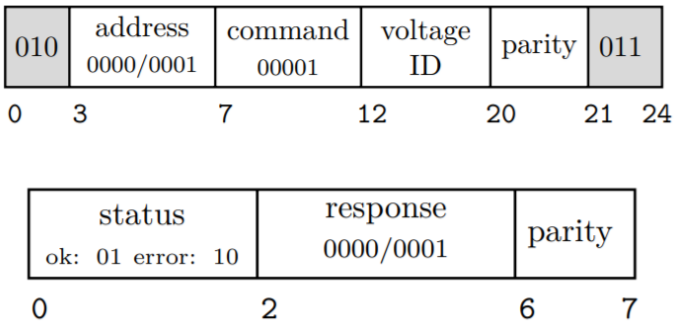
- **VCLK** - hodinový signál
- **VDIO** - dáta
- **ALERT#** - VR potvrdzuje, že zmena napätia bola dokončená

Piny VCLK a VDIO sa používajú na obojsmernú sériovú komunikáciu podobnú bežným sériovým protokolom ako I2C či SPI.SVID používa napäťové úrovne 0V (logická 0) a 1V (logická 1). Hodinový signál má frekvenciu **25MHz**. Hodinové (VCLK) aj dátové (VDIO) linky sú zapojené cez **pullup rezistory** na 1 V a sú aktívne riadené do logickej 0 pomocou CPU alebo VR pri výmene dát. Toto umožňuje pripojenie viacerých zariadení k SVID. To sa využije na pripojenie dosky Voltpillager na injektovanie príkazov.

Po úspešnom identifikovaní hodinovej a dátovej linky na základnej doske bol pripojený logický analyzátor DSlogic k zbernici SVID. Pozorovaním zbernice, nastavením známych hodnôt pre napätie a pomocou logického analyzátora nastaveného na protokol SVID sa reverzným inžinierstvo podarilo skonštruovať príslušné príkazy používané na konfiguráciu napäťového výstupu regulátorom napätia. Na základe toho bol zanalyzovaný príkaz, ktorý konfiguruje napätie a jeho odpoveď (Obrázok 6.6).

6.2.2 Vyvolanie chyby násobenia

Na vyvolanie chyby násobenia bol použitý veľmi podobný zdrojový kód, ako v prípade PlunderVolt útoku(viď výpis 6.1): použijú sa 2 operácie násobenia (skompilované do inštrukcie *imul*), ktoré sa vykonávajú s rovnakým vstupom v cykle a v tesnej blízkosti. Výsledok výpočtu sa porovnáva po každej operácii. Pred vstupom do cyklu je však generovaný signál *trigger* na spustenie hardvérového podvoltovania pomocou VoltPillager.



Obr. 6.6: Formát 24-bitového SVID príkazu od CPU do VR pre nastavenie napätia (hore) a 7-bitová odpoveď od VR k CPU (dole)

```
TRIGGER_SET // Nastavenie signalu trigger
do {
    i++;
    correct_a = operand1 * operand2 ;
    correct_b = operand1 * operand2 ;
    if ( correct_a != correct_b ) {
        faulty = 1;
    }
} while ( faulty == 0 && i < iterations ) ;
TRIGGER_RST // Resetovanie signalu trigger
```

Výpis 6.2: Kód v jazyku C použitý na demonštráciu chyby násobenia

Pri nastavení operandov násobenia na 0xAE0000 a 0x18, bol docielený rovnaký chybný výsledok na všetkých testovaných CPU(0xC500000 namiesto 0x10500000).

6.2.3 Extrakcia kľúča z dešifrovania/podpisu RSA-CRT v SGX

Tento útok opäť vychádza z útoku PlunderVolt *sgx_crt_rsa*[6], ktorý bol mierne upravený, aby mohol byť vykonaný hardvérový útok. Tento program vypočíta RSA podpis/dešifrovanie vo vnútri SGX enklávy pomocou štandardných funkcií knižnice *ipps*. Tento útok sa podarilo úspešne vykonať, získať chybné podpisy a zároveň sa podarilo dokázať, že tieto chybné hodnoty sa dajú použiť na získanie súkromného kľúča použitím útoku **Lenstra** [15].

6.2.4 Porovnanie so softvérovými útokmi

Oproti sotfvérovým útokom využívajúcim register MSR 0x150 má hardvérový útok VoltPillager niekoľko výhod. Prvou a najzásadnejšou je, že útok VoltPillager je uskutočniteľný aj po vydaní bezpečnostných opatrení CVE-2019-11157, ktoré obmedzili softvérové útoky. Čiže ani najnovšie CPU Intel s najnovším BIOSom sa **neubránia** útoku VoltPillager.

Druhou výhodou je **časová presnosť**. Autori útoku PlunderVolt uvádzajú, že na úspešné vykonanie útoku je potrebných aspoň 100 000 iterácií na vyvolanie chyby násobenia inštrukcie *imul*. Okrem toho, chyba nemôže byť cielená na **konkrétnu iteráciu cyklu**. VoltPillager dokáže prekonať obe obmedzenia. Z vhodnými nastaveniami sa vždy podarilo vyvolať

[6]Dostupný z: https://github.com/KitMurdock/plundervolt/tree/master/sgx_crt_rsa

chybu s menej ako 1680 iteráciami. Pre cielenie na konkrétnu iteráciu boli tiež nájdené vhodné nastavenia, ktorými sa podarilo dosiahnuť, že chyba v 75% prípadoch nastala v iterácii 14 334 až 14 934.

Pri softvérových útokoch vyvolávajúcich chyby bolo pozorované **veľké oneskorenie** medzi zápisom do registra MSR a skutočnou zmenu napätia. To obmedzuje schopnosť generovať krátke a potenciálne „hlbšie“ rušivé impulzy. Na rozdiel od toho, VoltPillager je obmedzený pri šírke rušivého impulzu iba rýchlosťou prebehu SR. Za predpokladu že napätie vyvolávajúce chybu je o $200mV$ nižšie ako napájacie napätie a typická $SR = 20mV/\mu s$, minimálna hodnota rušivého impulzu je teda $T_{min} = 20\mu s$.

6.3 TrustZone-M(eh)

Tento útok je zameraný na prelomenie ochrany prostredia dôveryhodného vykonávania (TEE) TrustZone-M, na mikrokontrolérech `ARMv8-M`. Útok sa zaoberá viacerými MCU a na každý MCU je útok dosť odlišný. V tejto práci je popísaný útok na Microchip SAM-L11, pretože na tento mikrokontrolér bol aj útok zrealizovaný. Všetky informácie v tejto sekcii sú čerpané z prezentácie[7] tohto útoku.

Cieľom je prečítať zabezpečené dáta (z zabezpečeného sveta) z nezabezpečeného sveta, s využitím modelu chyby na **preskočenie inštrukcie**. Tým sa docieli, že potenciálne nebezpečný firmvér je nahratý ako bezpečný.

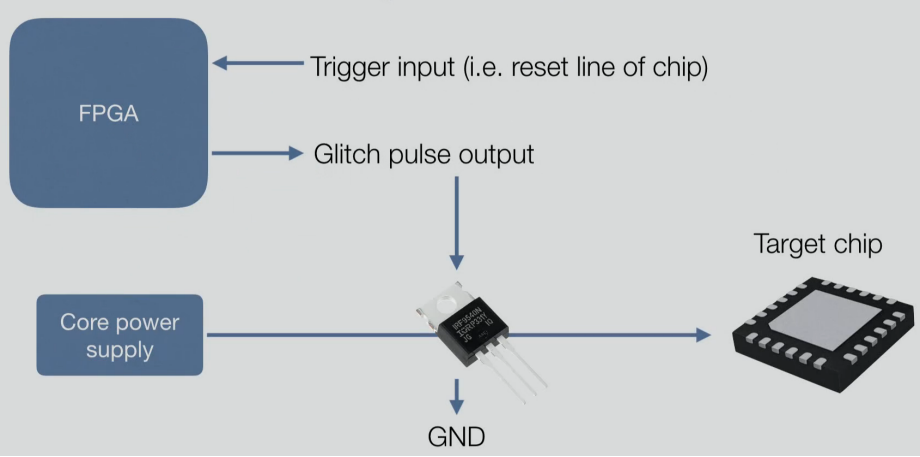
Generovanie pulzu na vyvolanie chyby v správnom momente je realizované pomocou **FPGA Lattice iCEstick**. Na to, aby mal pulz dostatočne veľký prúd a správne napätie na napájanie mikrokontroléra, je potrebné ešte použiť **N-kanalový MOSFET tranzistor** a **zdroj napätia**. Princíp injektovania chýb je znázornený na obrázku 6.7. Spočíva v tom, že za normálnych okolnoností sa na mikrokontrolér posiela cez tranzistor správne napájacie napätie zo zdroja. V momente, ako odošle FPGA impulz na tranzistor, tranzistor sa otvorí, čo spôsobí, že napájacie napätie je prepojené na zem. Šírka tohto pulzu určuje, ako dlho bude mikrokontrolér bez napájania. Čiže aj **moment** podvoltovania a ako **dlho** bude MCU podvoltovaný sú ovládané pomocou FPGA. Moment, kedy má prísť k podvoltovaniu sa zvyčajne volí tak, že na niektorý zo vstupov FPGA sa pošle informácia o tom, že príšlo k resetovaniu MCU a FPGA potom s presne definovaným **oneskorením** odošle impulz na tranzistor. Toto oneskorenie musí byť veľmi **presné** (zvyčajne μs až desiatky ns), aby sa podarilo preskočiť správnu inštrukciu.

Okrem spomínaného FPGA je možné použiť aj akýkoľvek MCU, ktorý má dostatočnú frekvenciu na injektovanie chýb v správnom čase. Napríklad v prezentácii k tomuto útoku sa spomína, že útok sa úspešne podarilo vykonať aj pomocou MCU Atmel ATTINY85, ktorý stojí približne 3€.

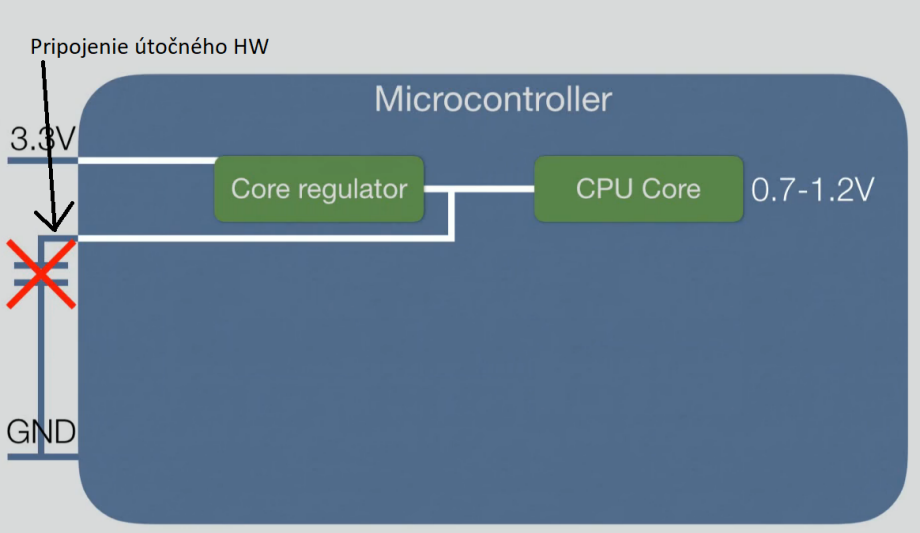
6.3.1 Pripojenie útočného HW k MCU

Vstup na napájanie mikrokontrolérov `ARMv8-M` požaduje hodnotu $3.3V$. Avšak periférie vo vnútri bežia na rôznom napätí. **Jadro CPU**, čo je pre tento útok najdôležitejšie beží typicky na $0.7V$ **až** $1.2V$. Na reguláciu napätia sa používa regulátor, ktorý sa nachádza priamo na čipe. Avšak na veľkom množstve čipov medzi regulátor napätia a jadro CPU býva pripojený kondenzátor, ktorý je mimo čipu. Tento kondenzátor sa používa na stabilizáciu napätia, pretože regulátor má zašumený výstup. Avšak ak sa odstráni tento kondenzátor,

[7] Dostupná z: https://media.ccc.de/v/36c3-10859-trustzone-m_eh_breaking_armv8-m_s_security



Obr. 6.7: Princíp podvoltovania MCU

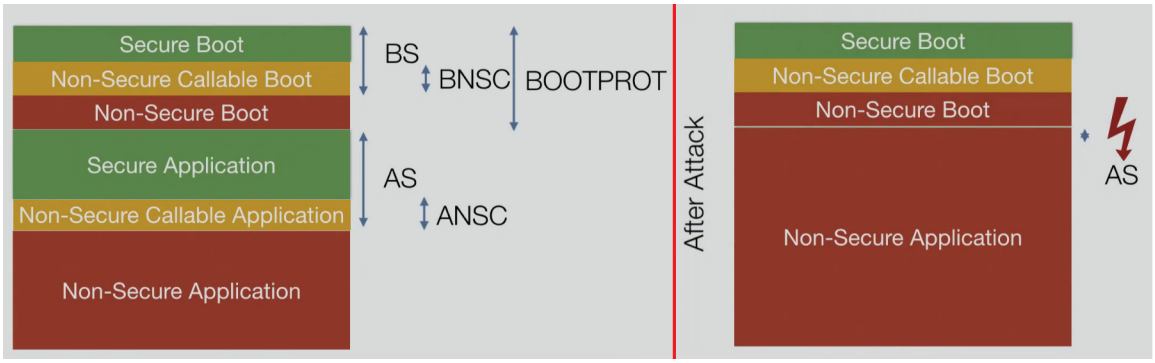


Obr. 6.8: Pripojenie útočného HW k MCU

máme priamy prístup ku zmene napätia jadra CPU. Ako je aj zobrazené na obrázku 6.8, po odstránení kondenzátora získame nový vývod, ku ktorému sa pripojí útočný hardvér, ktorý bude riadiť napätie jadra CPU.

6.3.2 Útok na MCU SAM-L11

Flash pamäť v `SAM-L11` je rozdelená na 2 časti a to na sekciu pre bootloader a pre aplikácie. TEE TrustZone-M, ako je už popísané v sekcii 3.2.2 rozdeľuje ešte každú časť na 3 ďalšie a to **zabezpečená**, **nezabezpečená zabezpečenej**, ktorú je možné zavolať z nezabezpečenej a **nezabezpečená**. Veľkosť jednotlivých sekcií sa konfiguruje iba pomocou jednotky `Implementation Defined Attribution Unit (IDAU)`, pretože tento MCU jednotku SAU nemá. IDAU sa nachádza v konfiguračnej flash pamäti a je nakonfigurovaná počas bootovania ROM. Na obrázku 6.9 je znázornené, ako jednotlivé registre jednotky IDAU môžu meniť jednotlivé oblasti flash pamäte. Čím je hodnota v registri väčšia, tým je aj daná oblasť väčšia. Cieľom je nastaviť hodnotu v **registri AS** na 0, pretože potom celá časť



Obr. 6.9: Stav jednotlivých sekcií flash pamäte pred útokom (vľavo) a po útoku (vpravo)

pamäte aplikácií je označená za nezabezpečenú, čo znamená, že z nezabezpečenej časti je možné prečítať pamäť, ktorá mala byť zabezpečená.

Keďže hodnoty registrov na začiatku majú hodnotu 0, postačí vyvolať chybu v momente čítania hodnoty tohto registru, čo spôsobí, že táto inštrukcia sa **preskočí** a hodnota zostane nulová.

Kapitola 7

Protokol replikácie útokov

Ako už je spomenuté v kapitole 6, útoky delíme na hardvérové a softvérové. Hardvérový útok je vždy náročnejšie zrealizovať a navyše vyžaduje aj pokročilé znalosti z oblasti elektrotechniky. Celkovo sa dosť líšia od softvérových. Preto je pre oba typy útokov vytvorený vlastný postup, ktorý podrobne popisuje replikáciu útokov v praxi.

7.1 Replikácia SW útoku

Na replikáciu softvérového útoku je potrebné:

1. Zaobstarať si potrebné vybavenie na útok
2. Zistiť, ako sa dá softvérovo zmeniť napätie zaobstaraného CPU/MCU
3. Overiť, či je daný MCU/CPU náchylný na útoky zmenou napätia
4. Aktivovať prostredie dôveryhodného vykonávania (TEE)
5. Vyvolať chybu počas vykonávania kódu v TEE a chybu analyzovať

7.1.1 Potrebné vybavenie na útok

V prípade softvérových útokov sa jedná iba o zariadenia zraniteľné voči útoku. To znamená, že pri útokoch na mikrokontroléry ARM postačí samotný **mikrokontrolér s podporou TrustZone** a zariadenie, ktorým je možné MCU **naprogramovať**. Vo väčšine prípadov je možné kúpiť celú vývojovú dosku s MCU, kde už je všetko potrebné. Pri útokoch na smartfóny s architektúrou ARM postačí samotný **smartfón**.

Pri počítačoch s architektúrou Intel x86 je potrebný **kompletný PC**, kde najdôležitejším prvkom je CPU, ktorý musí podporovať **TEE Software Guard Extensions - SGX**. Okrem toho, je ešte veľmi dôležitá **základná doska**, na ktorej nemôže byť verzia BIOSu s opatrením `CVE-2019-11157` a novšia, kedy bola zakázaná zmena napätia CPU pomocou softvéru. Čiže ani novšie CPU nie je možné použiť, pretože ak aj sedí do základnej dosky, CPU vyžaduje update BIOSu na staršej základnej doske. Na konfigurácii ostatných komponentov už nezáleži.

Okrem toho je vhodné mať pripravený voltmeter alebo osciloskop, v prípade problémov s nastavením napätia na CPU/MCU.

7.1.2 SW zmena napätia

Najjednoduchší spôsob na zistenie, ako sa mení napätie pomocou softvéru je vyhľadať existujúce štúdie softvérových útokov na daný CPU/MCU, kde je tento spôsob popísaný. Ako softvérovo zmeniť napätie na **smartfónoch** veľmi dobre popisuje útok **VoltJockey** [47], alebo útok na procesory **Intel** rôznej rady veľmi dobre popisuje útok **PlunderVolt**, ktorý je aj v sekcii 6.1 popísaný.

Ak však takýto útok ešte neexistuje, je možné skúsiť spôsob niektorého z útokov na tú istú architektúru. Ak však ani táto metóda nepomôže, zostáva zistiť ako sa mení napätie CPU/MCU pomocou softvéru z technickej dokumentácie k CPU/MCU ak je to vôbec možné.

7.1.3 Overenie náchylnosti na zmenu napätia

Na overenie, čí je možné vyvolať chybu zmenou napätia sa bežne používa **vyvolanie chyby počas násobenia**. Ako taký kód vyzerá bolo už popísané v sekcii 6.1.2 vo výpise 6.1. Tento kód vyvolá chybu bez ohľadu na to, aký model chyby bol použitý, alebo ak je aj v tomto bode neznámy, je možné ho zistiť podľa toho aká chyba bola vyvolaná. Pretože ak sa podarí viackrát vyvolať chybu pri operácii `var *= multiplier` je možné z výsledku zistiť, či došlo k **nastaveniu/resetovaniu** viacero bitov alebo k **prevráteniu** bitov, či je možné určiť miesto podľa veľkosti/času podvoltovania, alebo je miesto chyby **náhodné**. Alebo ak aj nastane **preskočenie** tejto inštrukcie alebo inštrukcie `var = 0xdeadbeef`, z výsledku je jasné, že prišlo k preskočeniu. Okrem overenia, či je možné vyvolávať chyby v CPU, môže kód slúžiť aj na nájdenie najlepších **parametrov**, kedy je vyvolanie chyby najpravdepodobnejšie. Hľadané sú nasledujúce parametre:

- Veľkosť podvoltovania
- Ako dlho má byť procesor podvoltovaný
- Frekvencia CPU

7.1.4 Aktivácia TEE

Na aktiváciu TEE na MCU s podporou TrustZone, je pred nahratím potrebné **zvoliť**, či je firmvér bezpečný alebo nie. V prípade použitia nezabezpečeného firmvéru sa pred nahratím zvolí aj **zabezpečený firmvér**, ktorý sa vyžaduje na správny chod a získava z neho potrebné potrebné informácie cez NSC (non-secure callable) svet. Aby užívateľ nezaviedol chybu do zabezpečeného sveta, niektoré MCU podporujú len nahratie nezabezpečeného firmvéru. V tom prípade je možné volať iba vopred definované funkcie zabezpečeného sveta cez NSC svet.

Na vytvorenie prostredia dôveryhodného vykonávania na smartfónoch s architektúrou ARM s operačným systémom Android, sa používa izolovaný OS **Trusty TEE**[1], ktorý je **vždy aktívny**, čiže nie je potrebné ho špeciálne aktivovať.

Procesory s architektúrou Intel x86 vyžadujú aktiváciu TEE Intel SGX v **BIOS nastaveniach**. Typicky je na výber z troch možností a to povolené, zakázané alebo softvérovo kontrolované, čo znamená, že softvér zvolí, či bude TEE aktívne alebo nie. Ak sa v BIOS setupe nenachádza táto možnosť, pravdepodobne doska/procesor nepodporuje SGX, alebo

[1] https://source.android.com/security/trusty

je softvérovo kontrolované. V prípade softvérovo kontrolovaného SGX je možné využiť aplikáciu `sgx-software-enable`[2], pomocou ktorej je možné povoliť SGX na Linuxových operačných systémoch.

7.1.5 Vyvolanie chyby v TEE a jej analýza

Cieľom softvérových útokoch je typicky vyvolať chybu v šifrovaní **RSA** alebo **AES**, ktoré TEE používa. Zdrojový kód, v ktorom je možné vyvolať požadovanú chybu má veľmi podobný koncept, ako pri vyvolaní násobenia a princíp môže byť popísaný nasledovne:

```
cipher_text_correct = encrypt(plaintex);
cipher_text = encrypt(plaintex);
while(cipher_text == cipher_text_correct){
    undervolt(delay);
    cipher_text = encrypt(plaintex);
}
```

Výpis 7.1: Kód v jazyku C vhodný na demonštráciu vyvolania chyby v šifrovaní

Funkcia `encrypt()` zabezpečuje šifrovanie otvoreného textu pomocou RSA alebo AES v rámci TEE. Pri použití vždy toho istého šifrovacieho kľúča by cyklus nemal nikdy skončiť, iba ak sa podarí vyvolať chybu. Funkcia `undervolt()` slúži ako spúšťač príkazu podvoltovania s oneskorením `delay`, ktorý musí bežať na inom vlákne. Frekvenciu CPU, hodnotu podvoltovania a aj čas podvoltovania je možné použiť z kroku, v ktorom bola vyvolaná chyba násobenia.

Na analýzu vyvolanej chyby sa pri RSA používa útok **Bellcore** popísaný v sekcii 5.1. Existujú aj voľne dostupné implementácie ako aj napríklad implementácia pomocou jazyka python na variantu CRT[3] v rámci voľne dostupných zdrojových kódov k útoku PlunderVolt.

Na analýzu chyby v šifrovaní AES sa používa **diferenciálna chybová analýza** popísaná v sekcii 5.2. Veľmi známa voľne dostupná implementácia DFA je od Jovanovica[4]. Pre použitie tejto implementácie je potrebné vyvolať chybu v **8. kole**. Okrem toho je vhodné vedieť v ktorom bajte bola chyba vyvolaná, pretože to výrazne zrýchli nájdenie šifrovacieho kľúča. Čiže pri implementácii vyvolania chyby je vhodné po úspešnom útoku vypísať pre každé kolo šifrovania správne zašifrovaný text a zašifrovaný text s chybou.

7.2 Replikácia HW útoku

Hardvérový útok sa so softvérovým zhoduje v bode aktivácia TEE. I keď aj niektoré ďalšie kroky majú rovnaký alebo podobný názov, sú v nich rozdiely, ktoré sú aj nižšie popísané. Na replikáciu hardvérového útoku je potrebné:

1. Zaobstarať si potrebné vybavenie na útok
2. Zostavenie útočného hardvéru
3. Zistiť, ako sa hardvérovo mení napätie zaobstaraného CPU/MCU

[2]Dostupná z: https://github.com/intel/sgx-software-enable/tree/master
[3]Dostupná z: https://github.com/KitMurdock/plundervolt/blob/master/sgx_crt_rsa/Evaluation/eval.py
[4]Dostupná z: https://github.com/Daeinar/dfa-aes

4. Vytvoriť/použiť existujúci firmvér a nahrať tento firmvér na zmenu napätia CPU-/MCU do útočného hardvéru
5. Pripojenie útočného hardvéru k základnej doske/pinu MCU
6. Overiť, či je daný MCU/CPU náchylný na útoky zmenou napätia
7. Aktivovať prostredie dôveryhodného vykonávania (TEE)
8. Vyvolať chybu počas vykonávania kódu v TEE a chybu analyzovať

7.2.1 Potrebné vybavenie na útok

Okrem potrebného vybavenia spomínaného v softvérových útokoch, je nutné si zaobstarať útočný hardvér. Typicky sa jedná o **mikrokontrolér** s vysokou frekvenciou, aby bolo možné odosielať príkazy na podvoltovanie s **vysokou presnosťou**, ktorá je pri útokoch zmenou napätia kľúčová. Typicky odosielané príkazy na podvoltovanie musia mať iné napätie ako je schopný mikrokontrolér odosielať (napríklad I/O MCU pracujú s napätím 3.3V a napätie jadra CPU/MCU 0.9V až 1.2V). Preto je potrebný **napájací zdroj** s požadovaným napätím a **tranzistor** typu N-kanálový MOSFET alebo vysokorýchlostný **multiplexor** alebo súčiastku špecializovanú na tieto účely. Okrem toho je vhodné mať prichystaný **osciloskop** alebo **logický analyzátor** s dostatočne vysokou vzorkovacou frekvenciou, ktorý pomôže pri riešení problémov.

7.2.2 Zostavenie útočného hardvéru

Ak sa nejedná o špecifický hardvér na daný útok, ktorý je možné zakúpiť ako 1 súčiastku, útočný hardvér je potrebné aj zostaviť. K tomu sa používa **schéma zapojenia**, ktorá väčšinou býva dostupná k útoku. Ak však schéma nie je dostupná, tak je nutné schému odvodiť z popisu k útoku. Väčšinou sa okrem útočného mikrokontroléra používajú iba bežne používané súčiastky ako je rezistor, kondenzátor alebo tranzistor, z ktorých nie je problém schému zhotoviť už pri základných znalostiach z elektrotechniky.

7.2.3 Princíp zmeny napätia

Na tento účel opäť najlepšie poslúžia rôzne štúdie ako napríklad **VoltPillager** je veľmi známy útok na CPU Intel popísaný v sekcii 6.2. Zmena napätia v tomto prípade pozostáva z odosielaniu príkazov regulátoru napätia cez zbernicu.

Útokmi na rôzne MCU architektúry ARM sa zaoberá napríklad výskumná skupina **chip.fail**[5], kde príkazy na podvoltovanie priamo riadia napätie jadra MCU. Vo všetkých preskúmaných štúdiach sa používal jeden z týchto 2 prístupov ktorými je možné podvoltovať CPU/MCU.

7.2.4 Vytvorenie a nahratie firmvéru na zmenu napätia

Keďže na zmenu napätia sa používajú 2 prístupy, takisto útočný firmvér má 2 podoby. Vytvoriť firmvér na odosielanie hodnôt na zbernicu je oveľa náročnejšie. pretože to vyžaduje najskôr **reverzné inžinierstvo** a **logický analyzátor** na zistenie ako príkaz na zmenu

[5]Dostupné z: https://chip.fail/

napätia vyzerá. Dokonca väčšinou je potrebné aj **detekovať zbernicu** na základnej doske pomocou **osciloskopu** alebo logického analyzátora, pretože technická dokumentácia k regulátoru napätia nie je dostupná. Avšak napríklad v útoku VoltPillager na CPU Intel už bolo všetko potrebné vykonané, takže postačí nahrať existujúci firmvér[6] do útočného mikrokontroléra Teensy 4.0.

V prípade útokov na mikrokontroléry architektúry ARM, podvoltovanie je možné realizovať pomocou jednoduchých príkazov na **nastavenie digitálneho výstupu** na logickú 0 a na logickú 1. Medzi tieto príkazy sa vloží oneskorenie, ktorého čas určuje, ako dlho bude trvať podvoltovanie. Príklad útočného firmvéru, je zobrazený vo výpise 7.2.

```
bool end = false;
while(end == false){
  for(unsigned int j = GLITCH_WIDTH_MIN; j < GLITCH_WIDTH_MAX; j++){
    set_digital_pin(PIN_GLITCH, LOW);
    delay(j);
    set_digital_pin(PIN_GLITCH, HIGH);
    if (read_digital_pin(PIN_READ)) {
      Serial.print("Glitch width: ");
      Serial.println(j);
      end = true;
      break;
    }
  }
}
```

Výpis 7.2: Kód v vytvorený v prostredí Arduino vhodný na nájdenie ideálnych parametrov na vyvolanie chyby

`GLITCH_WIDTH_MIN` a `GLITCH_WIDTH_MAX` určujú minimálny/maximálny čas podvoltovania, čiže šírku impulzu. Pri zraniteľných MCU s vysokou frekvenciou, tieto časové údaje majú typicky iba niekoľko na nanosekúnd. Útok bude úspešný, ak sa na vstupe pinu `PIN_READ` objaví logická 1. Vtedy sa na sériový výstup vypíšu informácie o šírke impulzu.

7.2.5 Pripojenie utočného HW k základnej doske/pinu MCU

Zbernica na ktorú sa pripája útočný hardvér typicky pracuje na **inom napätí** ako odosielané príkazy. Preto sa na tento účel používa **riadič zbernice**, ktorý slúži ako buffer, ale aj zníži napätie príkazov z MCU na hodnotu, na ktorej pracuje zbernica. V útoku VoltPillager sa na tento účel používa SN74LVC1G07DRLR[7]. Ak sa na riadenie napätia použije hotový zdrojový kód, detekcia zbernice pomocou **dokumentácie regulátora napätia** alebo pomocou osciloskopu/logického analyzátora musí prebehnúť v tomto kroku.

Pri útokoch na MCU sa pomocou technickej dokumentácie k danému MCU určí pin `VDDCORE`, ku ktorému sa pripája útočný hardvér. Pri použití vývojovej dosky je potrebné **odstrániť kondenzátory**, ktoré by dočasný výkyv napätia na jadre MCU tým pádom vyhladili. Útočný hardvér takisto nie je možné pripojiť priamo na pin VDDCORE, keďže pracuje na inom napätí. Využije sa k tomu **multiplexor** alebo **tranzistor**. K tranzistoru-

[6] Dostupný z: https://github.com/zt-chen/voltpillager/tree/master/voltpillager-firmware

[7] Technická dokumentácia dostupná z: https://www.ti.com/lit/ds/symlink/sn74lvc1g07.pdf?ts=1591037569523

/multiplexoru sa pripája aj **napájací zdroj**, ktorý nahradí odstránené kondenzátory. Ten musí byť nastavený na hodnotu napätia jadra, čo je typicky 0.9V až 1.2V.

7.2.6 Overenie náchylnosti na zmenu napätia

Na zraniteľnom CPU/MCU sa použije ten istý kód, ktorý slúži na vyvolanie chyby v násobení pri softvérovom útoku. Pomocou útočného MCU sa potom nájde hodnota podvoltovania, šírka impulzu a oneskorenie pred podvoltovaním. Nájsť správne oneskorenie môže výrazne urýchliť, ak zraniteľný CPU/MCU odošle informáciu, kedy začína inštrukcia násobenia. To je možné pri CPU s architektúrou Intel prostredníctvom **USB zbernice** alebo pomocou **RS232**, čo má lepšiu odozvu, čiže pomocou RS232 sa dosiahne aj lepšia presnosť. Pri útokoch na mikrokontroléry je možné použiť **digitálny I/O pin**.

7.2.7 Vyvolanie chyby v TEE a jej analýza

Pri útokoch na šifrovanie RSA alebo AES je možné použiť zdrojový kód v softvérovom vyvolaní chyby z výpisu 7.1. Jediný rozdiel je vo funkcii `undervolt()`, ktorá v tomto prípade odošle informáciu útočnému MCU pred začatím šifrovania. Analýza sa v tomto prípade takisto zhoduje so softvérovým útokom.

Ak je cieľom preskočenie inštrukcie, čo spôsobí, že sa načíta nezabezpečený firmvér ako bezpečný existujú 2 možnosti na vyvolanie chyby v TEE. Ako je popísané v sekcii 6.3.2, cieľom je preskočenie inštrukcie, ktorá nastavuje do príslušného registra veľkosť zabezpečeného sveta. Prvou možnosťou je **výkonová analýza**. Pri čítaní rozličných hodnôt registra sa mení aj odber. Preto do registra, ktorý určuje veľkosť zabezpečeného sveta sa zapíšu rozličného hodnoty a porovnáva sa, ako sa zmenil tento odber. Zápis rozličných hodnôt do registra AS, sa realizuje pomocou programovania poistiek. Keďže ostatný kód je rovnaký, tak moment, kedy sa odber najvýraznejšie mení určuje presný čas, kedy nastáva inštrukcia čítania hodnoty daného registra. Ak však výkonovú analýzu nie je možné vykonať, zostáva **náhodne voliť moment**, kedy má prísť k podvoltovaniu, čo však môže zabrať veľmi veľa času.

Kapitola 8

Implementácia

Keďže celkovo boli vykonané 3 útoky, implementácia je rozdelená na tri časti. Prvá časť popisuje replikáciu softvérového útoku PlunderVolt. V druhej časti je popísaná replikácia hardvérového útoku VoltPillager a v poslednej časti hardvérový útok TrustZone-M(eh) na mikrokontrolér SAM-L11. Vo všetkých častiach sa postupuje presne podľa protokolu replikácie hardvérových alebo softvérových útokov, ktorý je popísaný v kapitole 7.

8.1 Replikácia útoku PlunderVolt

Replikácia útoku PlunderVolt pozostáva z **5 krokov**, popísaných v protokole replikácii softvérového útoku. Všetky potrebné informácie sa nachádzali v samotnej štúdii a dokonca na vyvolanie chýb sú voľne dostupné zdrojové kódy. Tento útok bol uskutočnený na **2 rôzne** počítače s procesorom Intel. Jediný problém, ktorý pri replikácii vznikol, bol nevhodný operačný systém. Tento problém je bližšie popísaný v sekcii 8.1.3. Preloženie, inštalácia všetkých potrebných závislostí a spustenie voľne dostupných kódov so správnymi parametrami nebolo takisto triviálne, pretože to vyžadovalo naštudovanie samotných kódov, keďže k nim neexistuje návod na spustenie.

8.1.1 Potrebné vybavenie

Na replikáciu útoku PlunderVolt postačí samotné PC s CPU Intel s podporou TEE. Útok bol zreplikovaný na 2 počítačoch s procesorom Intel Core i5 6500 a i5 7400. Kompletná špecifikácia počítačov je v tabuľke 8.1. Oba PC mali na základnej doske verziu BIOSu ešte pred opatrením `CVE-2019-11157`, čo znamená, že napätie CPU by malo byť možné softvérovo zmeniť.

Na meranie napätia na CPU bol použitý multimeter ELMA BM257s.

Základná doska	ASRock Z170 EXTREME4	GIGABYTE B250M-Wind
Procesor	Intel Core i5 6500	Intel Core i5 7400
RAM	2xKingston 4 GB DDR4 2133 MHz	Tigo 8GB DDR4 2400 MHz
Disk	Seagate ST1000DX001	Colorful SL300 120GB SSD
Zdroj	SeaSonic SSP-350GT	Segotep Wasrship S7
OS	Microsoft Windows 7	Fedora 32

Tabuľka 8.1: Špecifikácia počítačov využitých k útokom

8.1.2 Zmena napätia

Napätie procesorov Intel sa mení pomocou registra MSR 0x150. Podrobnejšie vysvetlenie, ako sa vkladá do tohto registra požadovaná hodnota je v sekcii 6.1. Na otestovanie zmeny napätia bol použitý hotový kód z útoku PlunderVolt `undervolt.c`[1]. Všetky voľne dostupné kódy sú pripravené pre operačné systémy UNIXového typu, preto na PC, kde sa nachádzal OS `Windows 7`, bol pridaný OS `Ubuntu 20.04`, na ktorom boli vykonané aj všetky útoky.

Aby bolo možné čítať a zapisovať hodnotu do registra MSR, je potrebné použiť príkaz `sudo modprobe msr`. Potom kód postačilo preložiť príkazom:

```
gcc undervolt.c -Wall -lncurses -o test_voltage
```

a následne `test_voltage` spustiť s parametrom napätia. Či prišlo aj k zmene napätia CPU bolo overené pomocou hotového skriptu `0x198_read_msr.sh`[2], ktorý zobrazuje aktuálnu hodnotu napätia CPU. Pre dodaťočné overenie napätia CPU bol použitý aj multimeter. Napätie CPU je možné nameraťna niektorých kondenzátoroch v blízkosti CPU. Postupne bola zmeraná hodnota napätia na všetkých kondenzátoroch v blízkosti CPU a tam, kde bolo napätie najbližšie k hodnote, ktorú ukazoval skript bolo zvolené ako napätie CPU. Na oboch PC pri zmene napätia pomocou príkazu `test_voltage`, nastala aj zmena na zvolenom kondenzátore. Pri nastavení napätia na príliš nízku hodnotu, prišlo navyše k zamrznutiu PC, čo je očakávané chovanie.

8.1.3 Overenie náchylnosti na zmenu napätia

Na overenie náchylnosti na zmenu napätia pomocou vyvolania chyby násobenia, bol použitý hotový kód v jazyku C operation.c[3]. Ten postačilo preložiť príkazom `make`, čím sa vytvorí súbor `undervolt`. V útoku PlunderVolt sa uvadza, že pre úspešné vyvolanie chyby násobenia je potrebné nájsť pár napätie frekvencia. K nastaveniu frekvencii CPU napríklad na 1.1GHz a zobrazenie, či sa úspešne podarilo nastaviť frekvenciu je možné použiť príkazy:

```
sudo cpupower -c all frequency-set -u 1.1GHz
sudo cpupower -c all frequency-set -d 1.1GHz
grep "cpu MHz" /proc/cpuinfo
```

Žiaľ ani na jednom z PC nastavenie frekvencie CPU nebolo úspešné. Ukázalo sa, že na novších UNIXovských OS je nastavenie frekvencie na požadovanú hodnotu **zakázané**. I po dlhšom hľadaní, funkčné riešenie, ktoré by zabralo na použité OS sa nepodarilo nájsť. Riešením môže byť pretaktovanie CPU na požadovanú frekvenciu v BIOS nastaveniach, alebo inštalácia staršieho OS. Bola zvolená inštalácia staršieho OS, konkrétne **Ubuntu 18.04** bol nainštalovaný na oba PC, kde už nebol problém s nastavením frekvencie. Dokonca v útoku PlunderVolt výskumníci uvádzajú, že útoky sa podarilo úspešne vykonať na tomto OS, čiže by nemali vzniknúť už ďalšie problémy.

Po použití príkazu na prístup k čítaniu a zápisu registru MSR, preložení kódu a nastavení frekvencie bol spustený súbor `undervolt` s nasledujúcimi parametrami:

- Počet vlákien CPU: -t 4
- Počet iterácií: -i 1000000

[1] Dostupné z: https://github.com/KitMurdock/plundervolt/tree/master/utils

[2] Dostupné z: https://github.com/KitMurdock/plundervolt/blob/master/utils/0x198_read_msr.sh

[3] Dostupný: https://github.com/KitMurdock/plundervolt/blob/master/faulting_multiplications/operation.c

- Operandy násobenia: -1 0xFFFFFFFFFF -2 0xFFFFFFFFFF
- Nastavenie operandov ako meniace sa: -z max -x max
- Počiatočná veľkosť podvoltovania o 30mV: -s -30
- Koncová hodnota podvoltovania o 280mV: -e -280
- Počet krokov podvoltovania: -X 250
- Veľkosť kroku podvoltovania nastavená na 1mV: -v 1

Tieto parametre boli konzultované s tvorcom útoku a je pri nich najväčšia pravdepodobnosť vyvolania chyby. V tomto nastavení sa používajú **rôzne** hodnoty operandov násobenia, kde ich maximálna hodnota je 0xFFFFFFFFFF, čo výrazne urýchli nájdenie vhodných operandov. Aby parametre násobenia udávali fixnú hodnotu a nie maximum, stačí nastaviť parametre -z, -x na hodnotu `fixed`. Koncová hodnota podvoltovania je nastavená tak, že buď sa podarí na danej frekvencie vyvolať chybu, alebo príde až k zamrznutiu PC. Postupne boli testované všetky frekvencie procesorov od 1.0GHz po ich maximum. Na CPU i5 6500 sa podarilo vyvolať chyby na frekvenciách 1.6GHz, 1.7GHz a 2.0GHz a na CPU i5 7400 to boli frekvencie 2.4GHz, 2.6GHz a 2.7GHz. Výstup po vyvolaní chyby vyzerá nasledovne.

```
    ------   CALCULATION ERROR DETECTED   ------
 > Iterations    : 00000260
 > Operand 1     : 0000000057e19499
 > Operand 2     : 00000000ff1a223b
 > Correct       : 5792abadb5439143
 > Result        : 5792abada5439143
 > xor result    : 0000000010000000
 > undervoltage  : -240
 > Frequency     : 1700MHz
Done.
```

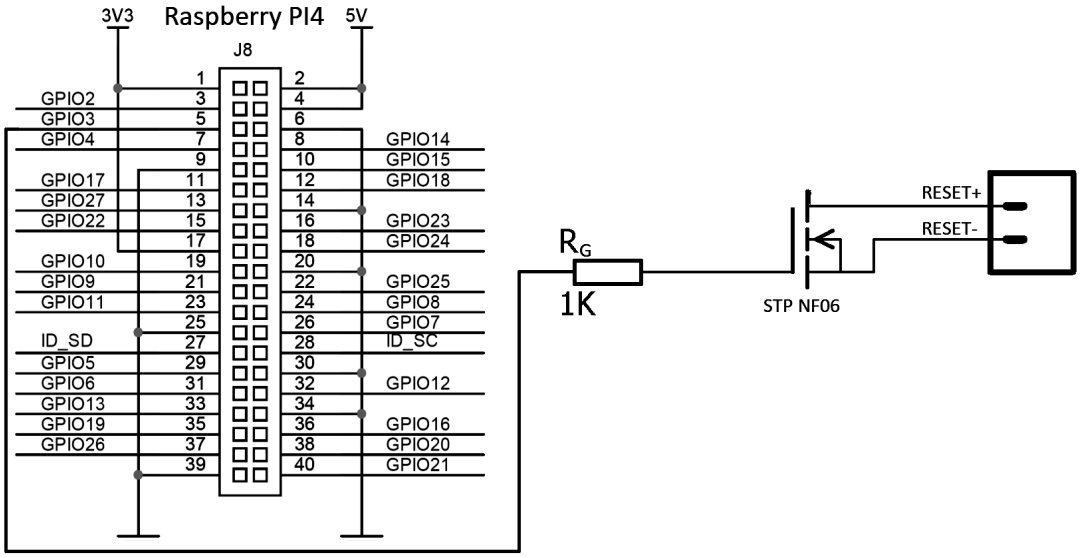
Na priloženom príklade vidno, že prišlo k preklopeniu 1 bitu, k čomu prichádzalo najčastejšie. To je presne druh chyby, ktorý je nutné vyvolať v šifrovaní RSA alebo AES.

Nájdenie optimálnych parametrov

Ideálne parametre pre PC s i5 7400 nebolo problém nájsť. Bola to frekvencia **2.6GHz** a podvoltovanie v rozmedzí **215mV až 220mV**. Tieto parametre dosahovali úspešnosť približne v **90% prípadoch** a osvedčili sa neskôr aj pri útokoch na šifry RSA a AES.

Nájdenie optimálnych parametrov pre CPU i5 6500 bolo oveľa náročnejšie, pretože najväčšia dosiahnutá úspešnosť **50%** bola pri frekvencii **2.0GHz** s podvoltovaním v rozmedzí **232mV až 238mV**. Navyše však veľmi často prichádzalo k zaseknutiu PC, po ktorom bolo jedinou možnosťou počítač manuálne resetnúť. Neskôr sa navyše ukázalo, že tieto parametre nie sú vhodné na vyvolanie chyby v šifrovaní AES. Preto bolo vykonané komplexné testovanie všetkých frekvencií. Avšak ručne reštartovať PC a vždy spúšťať ručne skript by zabralo aj niekoľko dní.

Na ukladanie všetkých potrebných informácií do súborov bol vytvorený vlastný skript **test_PV_VP.sh**, ktorý ukladá informácie úspešnosti na danej frekvencii, vyvolané chyby. Navyše príde k automatickému presunu na ďalšiu frekvenciu po 51 pokusoch. Za úspech



Obr. 8.1: Schéma zapojenia Raspberry PI4 k počítaču na ovládanie reštartu

je považované, ak príde k vyvolaniu chyby a za neúspech, ak príde k zamrznutiu PC. Na automatické spúšťanie skriptu pri štarte OS bol použitý príkaz `crontab -e`, ktorý otvorí súbor do ktorého postačí pridať cestu ku skriptu ktorý sa má spúšťať. Avšak stále by bolo nevyhnutné počítač manuálne reštartovať. Tento problém bol vyriešený pomocou **RaspberryPi 4.0** a **tranzistora** typu N-kanálový MOSFET, ktorý bol pripojený k PC namiesto reštartovacieho tlačidla. Schéma zapojenia k PC je zobrazená na obrázku 8.1. Veľmi dôležitá je aj hodnota rezistora R_G, ktorá musí byť minimálne 206.25Ω, pretože inak by mohlo prísť k poškodeniu RaspberryPi. Táto hodnota je odvodená z ohmovho zákona. Keďže napätie výstupu RaspberryPi je $3.3V$ a maximálny prúd je $16mA$[4], potom minimálna hodnota rezistora R_G bude:

$$R_G = \frac{U}{I} = \frac{3.3}{0.016} = 206.25\Omega$$

Pre RaspberryPi bol potom vytvorený program v jazyku python **reset_PC.py**, ktorý každých 10 sekúnd overil či PC nezamrzol pomocou príkazu `ping`. Čiže ak neprišla odpoveď, odošle sa logická 1 na výstupný pin GPIO3, čo otvorí tranzistor a príde k reštartu PC. Následne už postačilo ponechať PC zapnutý pár dní, kým sa nevykonali všetky pokusy o vyvolanie chyby. Dáta z testov sa nachádzajú v priečinku `data`. Súbor s názvom 10 označuje namerané dáta na frekvencii 1.0GHz, 11 na frekvencii 1.1GHz atď. Výsledky sú veľmi prekvapivé, pretože pri viacerých frekvenciách sa podarilo dosiahnuť až **100% účinnosť**. Pri ručnom testovaní sa podarilo dosiahnuť na tých istých frekvenciách maximálne 50% úspešnosť. To môže byť dôsledkom jedine automatickým spustením skriptu pri štarte PC.

Na druhom PC nemohlo byť vykonané automatické testovanie, pretože sa jednalo iba o zapožičaný PC bez možnosti zásahu do hardvéru. Na overenie, či príde k zvýšeniu úspešnosti bolo aspoň nastavené automatické spúšťanie skriptu pri štarte PC na frekvenciách, ktoré predtým úspešne vyvolali chybu. V tomto prípade však prišlo k miernemu **zhoršeniu**.

[4]Špecifikácie RaspberryPi získané z: https://www.tomshardware.com/reviews/raspberry-pi-gpio-pinout,6122.html

CPU	Frekvencia (GHz)	Podvoltovanie	Úspešnosť	Typ testu
i5 7400	2.6	od 215mV do 220mV	90%/8%/2%	Ručný
i5 6500	2.0	od 236mV do 240mV	50%/15%/35%	Ručný
i5 6500	1.8, 2.0, 2.1, 2.3 2.4, 2.5, 2.8	od 30mV do 280mV	100%/0%/0%	Automatický

Tabuľka 8.2: Parametre podvoltovania a úspešnosť vyvolania chyby v násobení

Podrobné informácie o úspešnosti najlepších parametrov sú zobrazené v tabuľke 8.2, kde v stĺpci úspešnosti sú uvedené 3 čísla ktoré vyjadrujú, úspešne vyvolanú chybu násobenia/žiadny efekt/zamrznutie PC.

8.1.4 Aktivácia TEE

Na oboch PC bolo možné aktivovať TEE Intel SGX v BIOS nastaveniach. Pred vykonaním útoku na šifrovanie RSA a AES pomocou voľne dostupných zdrojových kódov z útoku PlunderVolt, musí byť stiahnutá a nainštalovaná knižnica `sgx-step`[5]. Príkazy

```
make clean load
source /opt/intel/sgxsdk/environment
```

musia byť vykonané v priečinku `kernel`, po každom reštarte počítača.

8.1.5 Vyvolanie chyby v TEE

V útoku PlunderVolt je cieľom vyvolať chybu v šifrovaní RSA a AES, ktoré bežia v rámci enklávy prostredia dôveryhodného vykonávania.

Najskôr bol zreplikovaný útok na dešifrovanie RSA. Voľne dostupný kód[6] sa snaží vyvolať chybu v dešifrovaní RSA vo variante RSA, ktorá využíva optimalizáciu pomocou čínskych zvyškov. Dešifrovanie v rámci enklávy vychádza z príkladu od Intelu[7], z ktorého sa využili aj parametre šifrovania, čiže sú známe aj prvočísla `p` a `q`. To znamená, že postačí vykonať samotný útok Bellcore/Lenstra, ktorý ak je úspešný, získa jedno z týchto čísiel z čoho už je triviálne odvodiť pôvodný nešifrovaný text.

Na CPU i5 7400 s využitím najlepších parametrov získaných pri chybe násobenia, sa podarilo vyvolať chybu **40% prípadoch**. CPU i5 6500 bol o niečo menej úspešný a pri ručnom teste boli najlepšie výsledky dosiahnuté pri frekvencii **2.0GHz** a to **25% úspešnosť**. Avšak jedná sa iba o vyvolanie samotnej chyby, **nie** aj jej využiteľnosť.

Na odvodenie kľúča z vyvolanej chyby bol použitý skript `eval.py`[8], v ktorom sa nachádza implementácia útoku Bellcore aj Lenstra. Do skriptu postačilo zadať chybne a správne dešifrovaný text pre analýzu Bellcore alebo chybne dešifrovaný text a správne šifrovaný pre analýzu Lenstra. Na oboch CPU sa podarilo získať kľúč z chybného dešifrovania v menej ako **40% prípadoch**. To znamená, že celková úspešnosť je ešte **oveľa nižšia**. Na PC s CPU i5 6500 bol vykonaný aj automatický test všetkých frekvencií pomocou skriptu

[5] Dostupná z: https://github.com/jovanbulck/sgx-step

[6] Dostupný z: https://github.com/KitMurdock/plundervolt/tree/master/sgx_crt_rsa

[7] Dostupný z: https://www.intel.com/content/www/us/en/develop/documentation/ipp-crypto-reference/top/public-key-cryptography-functions/rsa-algorithm-functions/rsa-primitives/example-of-using-rsa-primitive-functions.html

[8] Dostupný z: https://github.com/KitMurdock/plundervolt/blob/master/sgx_crt_rsa/Evaluation/eval.py

CPU	F(GHz)	Podvoltovanie	Úspešnosť	Typ testu
i5 7400	2.6	od 215mV do 220mV	45%/45%/10%/15%	Ručný
i5 6500	2.0	od 236mV do 240mV	25%/25%/50%/10%	Ručný
i5 6500	1.4	od 30mV do 280mV	43%/0%/57%/39%	Automatický

Tabuľka 8.3: Parametre podvoltovania a úspešnosť vyvolania chyby v dešifrovaní algoritmu RSA

test_PV_VP.sh, ktorý bol rozšírený o možnosť komplexného testovania RSA. To prinieslo veľmi prekvapivé výsledky, pretože najúspešnejšia bola frekvencia **1.4GHz**, čo pri chybe násobenia bola jedna z menej úspešných.

Podrobnejšie štatistiky sú popísané v tabuľke 8.3, kde F udáva frekvenciu CPU a oproti tabuľke 8.2 je úspešnosť doplnená ešte o štvrtú hodnotu, ktorá udáva **celkovú úspešnosť**. To znamená, ako často sa podarilo vyvolať chybu, ktorú je možné zneužiť na získanie dešifrovacieho kľúča RSA. Čiže ak je úspešnosť vyvolania akejkoľvek chyby 40%, z čoho sa podarilo získať dešifrovací kľúč s 50% úspešnosťou, celková úspešnosť bude 20%. Na výpočet celkovej úspešnosti bol vytvorený python skript **success_rate.py**, ktorý vypíše celkovú úspešnosť pre každý súbor ktorý sa nachádza v zadanom priečinku. V skripte sa nachádza aj implementácia útoku Bellcore, ktorá je prevzatá zo spomínaného skriptu `eval.py`.

Útok na šifru AES[9] bol dokonca ešte menej úspešný. V prípade CPU i5 7400 iba v **15% prípadoch** dochádzalo k vyvolaniu chyby a dokonca prišlo aj k častému zaseknutiu PC, s čím doposiaľ na tomto CPU nebol problém. Počet iterácii a zmena hodnôt podvoltovania tento problém iba zhoršili. Celkovo bolo na tomto PC vyvolaných 20 chýb v šifrovaní. V programe sa po vyvolaní chyby vypíše všetkých 10 kôl šifrovania správne a chybne zašifrovaného textu, avšak ani v 1 prípade **nenastala** požadovaná chyba v 8.kole šifrovania. Dokonca väčšinou sa tieto 2 zašifrované texty úplne líšili v každom kole šifrovania. V ostatných prípadoch bolo zrejmé, že chyba nastala v 10. kole a vždy prešlo k preklopeniu v rovnakých bitov, avšak to nestačí na úplnú extrakciu kľúča.

S procesorom i5 6500 bol dokonca ešte väčší problém na frekvencii 2.0GHz, ktorá bola zatiaľ najúspešnejšia na vyvolanie chyby v násobení aj RSA pri ručných testoch. Chybu sa nepodarilo vyvolať a dokonca ani na žiadnej frekvencii, ktoré dosahovali vysoké úspešnosti aj pri automatických testoch. Takže jedinou možnosťou bolo do skriptu **test_PV_VP.sh** pridať možnosť automatického vyvolávania chýb v šifrovaní AES.

Nakoniec sa podarilo chybu vyvolať, ale s veľmi **nízkou úspešnosťou**. Avšak vyvolané chyby mali ten istý charakter, ako pri CPU i5 7400. Celkovo bolo vyvolaných iba **25 chýb** z 1071 pokusov z toho ani raz **nenastala chyba** v požadovanom 8. kole, iba raz nastala chyba v 6. kole a ostatné chyby mali neznámy charakter.

Podrobné štatistiky útokov na šifrovanie AES sú v tabuľke 8.4. Tabuľka obsahuje rovnaké parametre, ako tabuľka úspešnosti vyvolania chyby v dešifrovaní RSA.

Súhrn

Po zaobstaraní si všetkého potrebného vybavenia bolo otestované, či je možné meniť napätie CPU. Po úspešnej zmene sa prešlo vyvolaniu chyby v násobení. K úspešnému vyvolaniu chyby bolo nutné preinštalovať OS, na ktorom bolo možné nastaviť frekvenciu CPU. K nájdeniu optimálnych parametrov bol vytvorený vlastný skript. Keďže prichádzalo k častému

[9]Použité zdrojové kódy dostupné z : https://github.com/KitMurdock/plundervolt/tree/master/sgx_aes_ni

CPU	F(GHz)	Podvoltovanie	Úspešnosť	Typ testu
i5 7400	2.6	od 215mV do 220mV	15%/45%/40%/0%	Ručný
i5 6500	-	-	0%/0%/0%/0%	Ručný
i5 6500	1.0, 1.1 a 1.4	od 30mV do 280mV	8%/0%/92%/0%	Automatický

Tabuľka 8.4: Parametre podvoltovania a úspešnosť vyvolania chyby v šifrovaní algoritmu AES

zamrznutiu CPU, bol vytvorený vlastný systém na reštart PC pri zamrznutí pomocou Raspberry PI 4.0, čím sa podarilo dosiahnuť **100% úspešnosť**na vyvolanie chyby násobenia. Po aktivácii TEE sa prešlo k vyvolaniu chyby v dešifrovaní RSA a šifrovaní AES. Optimálne parametre získané pri vyvolaní chyby násobenia nedosahovali dostatočnú úspešnosť a preto bol skript rozšírený aj na nájdenie optimálnych parametrov na vyvolanie chyby v AES a RSA. Pri dešifrovaní RSA sa v najlepšom prípade podarilo v **39% prípadoch** vyvolať chybu, ktorú bolo možné zneužiť na získanie tajného kľúča. V šifrovaní AES sa síce takisto podarilo vyvolať chybu, ale ani jednej chyby **nedalo zneužiť** na odvodenie šifrovacieho kľúča.

8.2 Replikácia útoku VoltPillager

Keďže sa jedná o harvérový útok, replikácia sa v tomto prípade riadi podľa protokolu replikácie hardvérových útokov. Na útok sú opäť voľne dostupné všetky zdrojové kódy[10], ktoré vychádzajú z útoku PlunderVolt. Jediným a zásadným rozdielom je nastavenie napätia pomocou útočného hardvéru. Zreplikovať tento útok je **náročnejšie** ako útok PlunderVolt, pretože to vyžaduje znalosti z elektroniky pre prácu s osciloskopom, mikrospájkovačkou a porozumenie schém zapojenia. Krok aktivácie TEE je vynechaný, pretože sa zhoduje s postupom popísaným v replikácii softvérového útoku.

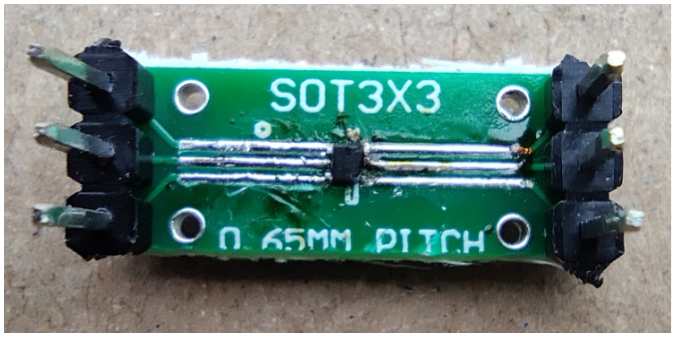
8.2.1 Potrebné vybavenie na útok a zostavenie útočného hardvéru

Útok bol uskutočnený na 1 PC s CPU i5 6500, podrobná špecifikácia tohto PC je v tabuľke 8.1. Útočný hardvér bol zložený z 3 komponentov:

- Teensy 4.0 Development Board
- 2x NL17SZ07XV5T2G - slúži na zmenu napätia hodinového a dátového signálu odosielaných z dosky Teensy na zbernicu SVID. (z 3.3V na 1.1V).
- 2x SOT IC adaptér

Adaptér SOT IC slúži na prevedenie pinov z púzdra SOT na DIP. Čiže sa k nemu prispájkuje riadič zbernice NL17SZ07XV5T2G (obrázok 8.2), aby bolo možné prípojiť Teensy dosku k NL17SZ07XV5T2G a z neho sa pripojiť na zbernicu SVID. K tomu bola použitá **mikrospájkovačka** s veľmi tenkým hrotom. Hrot musí byť tenší, ako sú vývody riadiča zbernice, čo je $0.2mm$, pretože inak môže ľahko prísť k poškodeniu vodivých ciest, ktoré vedú k vývodom. K prispájkovaniu je odporúčané použiť **spájkovaciu pastu**. Avšak tú sa nepodarilo zohnať, tak bol použitý bežný cín, s ktorým je to oveľa náročnejšie a zároveň je menšia šanca, že sa to podarí úspešne. Medzi vodivými cestami na ktoré sa

[10]Dostupné z: https://github.com/zt-chen/voltpillager



Obr. 8.2: Prispájkovaný NL17SZ07XV5T2G na SOT IC adaptér

NL17SZ07XV5T2G prispájkováva, je rozdiel iba 0.45mm, čo znamená, že veľmi ľahko môže vzniknúť skrat. Preto po spájkovaní prebehla kontrola, či nevznikli skraty a prípadne aj studené spoje pomocou multimetra. Nájdený bol 1 skrat, ktorý bol následne odstránený.

Okrem toho k vykonaniu útoku boli ešte použité nasledujúce zariadenia a súčiastky:

- Osciloskop RIGOL DS1074Z
- Multimeter ELMA BM257s
- Mikrospájkovačka ZD-931
- MAX232
- Kontaktné pole so sadou prepojok

Keďže najpresnejšou metódou na časovanie podvoltovania je pri tomto útoku posielať príkazy z počítača do dosky Teensy pomocou RS232, bolo potrebné znížiť napätie zo sériového portu na napätie, na ktorom pracuje doska Teensy (logická 1 12V na 3.3V, logická 0 z -12V na 0V). Na tento účel bola použitá doska s integrovaným obvodom MAX232. Základná doska ASRock Z170 nemá zo zadnej strany dosky výstup RS232, ako bolo typické na starších doskách, ale z technickej dokumentácie k základnej doske ASROCK[11], bolo zistené, že sériový port je priamo na doske označený ako COM1. Konektor z dosky MAX232 nie je možné pripojiť priamo do konektora COM1, pretože MAX232 upravuje iba signály RX a TX, avšak príkaz na podvoltovanie sa odosiela na pin DTR. To znamená, že pin RX dosky MAX232 bol pripojený na pin DTR zo sériovej linky.

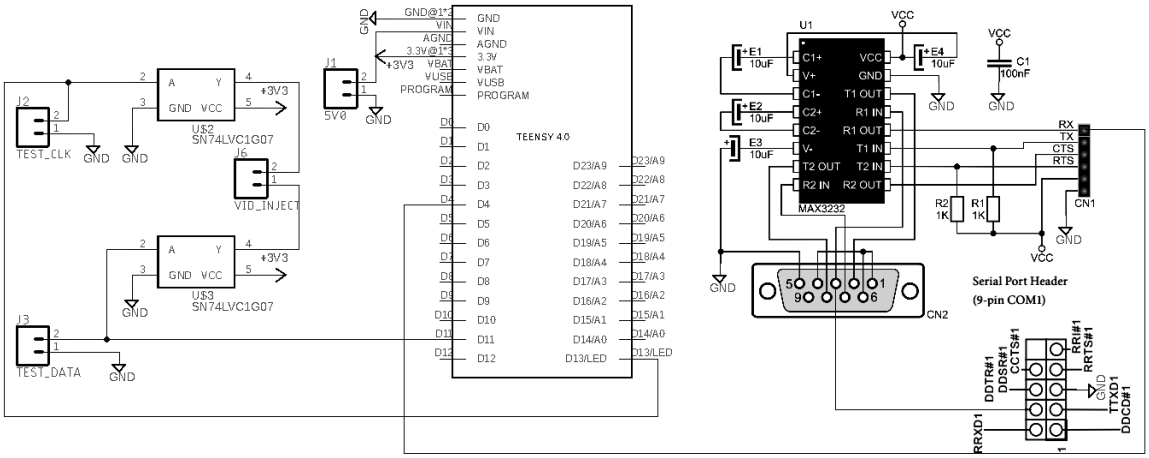
Schéma zapojenia, podľa ktorej bol útočný hardvér zostavený je zobrazený na obrázku 8.3. Oproti pôvodnej schéme zapojenia získanej zo štúdie útoku VoltPillager[12], je ešte rozšírená o už spomínaný integrovaný obvod MAX232 a jeho pripojenie k PC a doske Teensy.

8.2.2 Princíp zmeny napätia

Útok VoltPillager využíva na zmenu napätia SVID zbernicu na základnej doske, ku ktorej sa pripojí útočný hardvér, ktorý odosiela príkazy na zmenu napätia. Podrobnejší popis sa nachádza v sekcii 6.2.

[11] Dostupná z: https://download.asrock.com/Manual/Z170%20Extreme4.pdf

[12] Dostupná z: https://github.com/zt-chen/voltpillager/blob/master/VoltPillager-board/VoltPillager-board.png



Obr. 8.3: Schéma zapojenia útočného hardvéru k vykonaniu útoku VoltPillager

8.2.3 Nahratie firmvéru na zmenu napätia

Na zmenu napätia bol použitý už hotový firmvér voltpillager-firmware[13].

Aby bolo možné nahrať firmvér na dosku, je najskôr nevyhnutné stiahnuť a nainštalovať programy:

- Arduino (podporované verzie 1.8.5, 1.8.9, 1.8.11, 1.8.12, 1.8.13)
- Teensydrino
- TeensyTimerTool library v0.1.8 (Pomocou správcu knižníc v arduino)

Po inštalácii všetkých závislostí postačí preložiť a nahrať správny firmvér k útoku do dosky Teensy v prostredí Arduino.

8.2.4 Pripojenie útočného HW k základnej doske/pinu MCU

Pred pripojením útočného hardvéru musela byť **detekovaná** zbernica SVID na základnej doske. Prvým krokom je detekovanie **regulátora napätia (VR)** na základnej doske. Niektorí predajcovia poskytujú schematické diagramy svojich dosiek, ktoré výrazne zjednodušujú postup. Žiaľ, väčšina predajcov nezverejňuje takú podrobnú dokumentáciu o svojom hardvéri. VR sú zvyčajne umiestnené v tesnej blízkosti k CPU a k veľkým spínacím tranzistorom a induktorom, vďaka čomu je ľahké ich vizuálne identifikovať. Každá súčiastka má na sebe názov, čiže je možné si danú súčiastku podľa tohto názvu vyhľadať, či sa jedná skutočne o VR. Väčšinou nie je problém nájsť aspoň základný popis k danej súčiastke. Na základnej doske použitej k útoku sa VR nachádza priamo nad procesorom. Jedná sa o regulátor napätia **ISL95824**.

Po identifikovaní VR bolo možné identifikovať aj **zbernicu SVID**. V technickej dokumentácii k VR by mali byť všetky potrebné informácie, ktoré piny sú pripojené na VR. Avšak väčšinou dokumentácia nie je k dispozícii. Preto bola zbernica identifikovna pomocou **osciloskopu** a to tak, že postupne sa pripája osciloskop na vývody VR a následne bolo

[13]Dostupný z :https://github.com/zt-chen/voltpillager/tree/master/voltpillager-firmware

Obr. 8.4: Signál VCLK zobrazený na osciloskope

analyzované, či sa signály na danom výstupe chovajú podľa toho, ako by sa mali na zbernici SVID. K útoku sa využívajú iba signály VDIO a VCLK zbernice SVID, preto signál ALERT# nebol identifikovaný.

Detekcia zbernice SVID pomocou osciloskopu

SVID musí byť prepojená s **pullup rezistormi**, ktoré na zbernicu privádzajú logickú 1, v prípade, že sa na zbernicu nič nevysiela. Takže piny VR, ktoré boli pripojené k rezistorom, boli overené ako prvé. Hľadaný priebeh, ktorý by sa mal zobraziť na osciloskope pre signál VCLK je typický **hodinový signál** s frekvenciou 25MHz. U signálu VDIO, dáta hľadaného signálu vychádzajú z obrázku 6.6 so vzorkovacou frekvenciou 25MHz. Príkazy na zbernicu sú posielané i keď je v BIOSe nastavené pevné napätie pre CPU i keď v obmedzenom počte oproti iným režimom. Pre čo najrýchlejšiu detekciu bol v BIOSe nastavený režim, ktorý dynamicky mení napätie na procesore - 'auto'. Prístup k pinom VR pomocou osciloskopu bol v PC skrini veľmi obmedzený a pri tak malých pinoch bola veľmi veľká šanca, že vznikne skrat. Preto pre lepší prístup musela byť základná doska vytiahnutá zo skrine. To uľahčí aj prípadné pripojenie Voltpillager dosky na SVID zbernicu.

Ešte pred hľadaním zbernice bolo nutné nastaviť osciloskop, aby detekoval hľadané signály. Na osciloskope bolo preto nastavené zobrazenie 0.5V na 1 dielik a časový usek 20ns na 1 dielik, čo by malo byť postačujúce na zobrazenie signálov, ktoré pracujú na napätí 1V a frekvencii 25MHz.

Po správnom nastavení postačilo už nájsť iba zem (GND) na základnej doske, kde sa vždy pripája 1 vývod osciloskopu. K tomu bola opäť použitá technická dokumentáciu k základnej doske a využitá bola zem sériového portu COM1, zobrazená v schéme zapojenia na obrázku 8.3.

Následne už bolo možné hľadať samotné signály. Hodinový signál **VCLK** zobrazený na obrázku 8.4, pri správnom nastavení osciloskopu nebolo problém nájsť.

Nájdenie signálu VDIO už je o dosť náročnejšie, ako v prípade hodinového signálu, ktorý má typický periodický priebeh. K detekcii môže pomôcť logický analyzátor a k nemu softvér na analýzu SVID dát. Avšak ten nebol k dispozícii, ale tento signál je možné detekovať aj pomocou osciloskopu. Stačí zobrať väčšiu časť dát a zanalyzovať, či sa v dátach nachádza

Obr. 8.5: Analýza dátového signálu

24-bitový SVID príkaz od CPU, alebo 7 bitová odpoveď od VR. Preto bola pre tento prípad zobrazená väčšia časť signálu, ako pri hľadaní VCLK (200ns/dielik). Navyše dosť pomôže fakt, že typicky je tento signál veľmi blízko hodinového signálu. Po chvíli bol nájdený kandidát na hľadaný signál. Čiže boli zanalyzované dáta tohto kandidáta.

V kandidátovi na dátový signál zobrazenom na obrázku 8.5 bol pri analýze nájdený 24-bitový SVID príkaz od CPU k VR, ktorý presne zodpovedá očakávanému príkazu podľa obrázka 6.6:

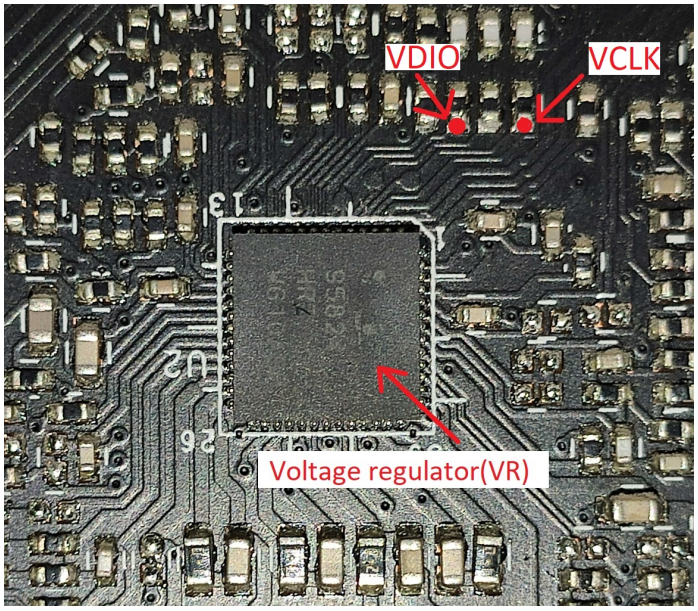
- Prvé 3 bity - 010
- Adresa - 0000
- Príkaz - 00111
- ID napätia - 00010101
- Parita - 0
- Posledné 3 bity - 011

Z toho vyplýva, že tohto kandidáta je možné označiť ako dátový signál SVID zbernice - **VDIO**.

Obrázok 8.6 ukazuje, kde na základnej doske bol detekovaný dátový a hodinový signál SVID zbernice. Medzi rezistormi na ktorých bol zmeraný signál VDIO a VCLK, sa nachádza ešte 1 rezistor, na ktorý bude pravdepodobne privedený signál ALERT#, pretože tieto 3 rezistory sú pripojené na 3 vývody procesora vedľa seba. Potvrdzuje to aj blokový diagram regulátora napätia[14], kde sú piny VCLK, ALERT# a VDIO vedľa seba presne v očakávanom poradí. Avšak keďže na tento pin sa nepripája doska VoltPillager, nebolo analyzované, či sa skutočne jedná o signál ALERT#.

Po detekovaní zbernice už postačilo pripojiť útočný hardvér k zbernici, to sú prípojky `TEST_CLK(J2)` a `TEST_DATA(J1)` označené na schéme zapojenia z obrázku 8.3. Pripojenie celého útočného hardvéru vrátane prípravy na automatické testovanie, kde je potrebný reštart pomocou RaspberryPI možno vidieť v prílohe C.1.

[14] https://www.renesas.com/sites/default/files/isl95824_0.png



Obr. 8.6: Detekcia pinov SVID zbernice na základnej doske

8.2.5 Overenie náchylnosti na zmenu napätia

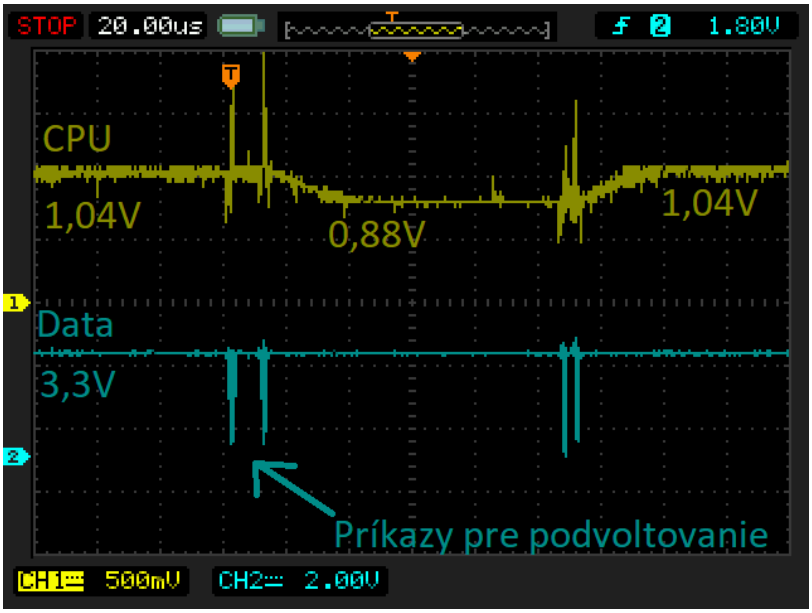
Ešte pred vyvolaním chyby v násobení bolo overené, či sa vôbec podarí úspešne zmeniť napätie CPU. K tomu či sa to úspešne podarilo, je potrebné sledovať napätie na procesore. To je možné napríklad pomocou **nástrojov na monitorovanie** (napríklad lm-sensors), alebo pomocou osciloskopu pripojeného k základnej doske. Keďže podvoltovanie trvá iba pár mikrosekúnd, najlepšou voľbou je použiť **osciloskop**, ktorý podvoltovanie určite zaznamená.

Po nahratí správneho firmvéru do dosky Teensy a pripojení útočného hardvéru k zbernici, postačí zapojiť dosku Teensy k PC cez USB, čo umožní komunikáciu s doskou. To znamená dostávať a odosielať do nej príkazy.Pre komunikáciu so sériovým portom dosky Teensy je možné použiť napríklad softvér Arduino (Nástroje->Monitor sériového portu). Pomocou sériovej linky je teraz možné príkazom odoslať parametre podvoltovania. Príkaz má tvar:

$$< N >< V_p >< T_p >< V_f >< T_f >< V_n >$$

kde:

- N - Počet opakovaní podvoltovania
- V_p - Napätie pred podvoltovaním. Typicky $\leq V_n$
- T_p - Čas prepnutia na napätie V_p a čas, koľko je napätie V_p udržiavané
- V_f - Napätie použité na vloženie chyby
- T_f - Čas prepnutia na napätie V_f a čas, koľko je napätie V_f udržiavané
- V_n -Stabilné prevádzkové napätie po V_f



Obr. 8.7: Podvoltovanie CPU zaznamenané pomocou osciloskopu

Nesprávne nastavenie týchto parametrov ovplyvní stabilitu systému. To znamená, že ak je napríklad nastavené napätie príliš nízke, CPU zamrzne alebo zhavaruje. Parametre boli nastavené na:

$$N = 10, V_p = V_n = V_{cc} = 1,050V, V_f = 0,88V, T_p = 10\mu s, T_f = 32\mu s$$

Následne na ich uplatnenie sa zavolal na sériovu linku príkaz `arm`, a odoslal sa signál `trigger`. Ako už bolo vyššie vysvetlené, využíva sa k tomu USB zbernica alebo sériová linka RS232. Obe možnosti boli otestované. V prípade RS232 sa postačilo poslať logickú 0 na pin DTR linky RS232 a v prípade zbernice USB, sa odošle príkaz `delay n` na sériovú linku dosky Teensy, kde `n` určuje čas (v μs), za koľko doska príjme signál `trigger`.

Obrázok 8.7 ukazuje, ako prebieha podvoltovanie CPU. Osciloskop meria napätie na CPU a dáta, ktoré odosiela Teensy na NL17SZ07XV5T2G a ten ich následne odosiela na zbernicu SVID. V SVID zbernici tento signál predstavuje žiadosť od CPU k VR o zníženie napätia na 0.88V. VR napätie zníži, pretože na obrázku 8.7 možno vidieť, že napätie pokleslo na požadovanú hodnotu. Po pár mikrosekundách (nastavené podľa parametrov) dátový pin dosky Teensy odošle žiadosť na navýšenie napätia na pôvodnú stabilnú hodnotu. To je nutné, pretože pri dlhšom podvoltovaní CPU zamrzne. Navyše pre vyvolanie chýb sa vyžaduje presne takýto prístup.

Na vyvolanie chyby v násobení bol použitý voľne dostupný kód z útoku voltpillager[15], v ktorom sa nachádzajú aj príkazy na podvoltovanie, aj samotné násobenie. To je možné, pretože útočný hardvér sa pripája k rovnakému PC, na ktorý sa útočí, pretože takto sa dosiahne, čo najpresnejšie časovanie, keďže PC posiela signál `Trigger` vždy pred násobením. V súbore makefile sa najskôr zvolil typ signálu `Trigger`, potom bol kód preložený príkazom `make`, potom nastavená frekvencia CPU na **2GHz** a následne bol kód spustený s nasledujúcimi parametrami, ktorými sa podarilo vyvolať prvú chybu:

- Počet vlákien CPU: –calc_thread_num 4

[15]Dostupný z: https://github.com/zt-chen/voltpillager/tree/master/poc/mul

- Počet iterácií: –iter 1000000
- Počet pokusov: –retries 10
- Baudrate: -b 115200
- Sériový port Teensy: -p /dev/ttyACM0
- Operandy násobenia: –calc_op1 0xae0000 –calc_op2 0x18 -g
- Hodnota napätia pred podvoltovaním: –pre_volt 0.859
- Hodnota podvoltovacieho napätia: –glitch_voltage 0.655
- Hodnota napätia po podvoltovaní: –rst_volt 0.859
- Oneskorenie: –pre_delay 35
- Šírka impulzu: –rst_delay -30

Na komplexné testovanie chyby multiplikácie bol vytvorený skript **fault_VP.sh**. Najlepšia frekvencia už **nebola hľadaná**, pretože tú je možné použiť z útoku PlunderVolt. Hľadané bolo vhodné oneskorenie od príkazu `Trigger`, kedy má doska Teensy odoslať príkaz na zbernicu SVID. Testované bolo oneskorenie od $5\mu s$ po $100\mu s$ s krokom $5\mu s$. Na väčšom oneskorení už bolo zaznamenané výrazné zhoršenie výsledkov. Pre každé oneskorenie je vytvorený vlastný súbor v priečinku `data`, ktorý obsahuje údaje o úspešnosti a vyvolané chyby. Súbory sú pomenované podľa oneskorenia, napríklad súbor `5` obsahuje dáta pre oneskorenie $5\mu s$.

Testy boli vykonané pri odosielaní príkazu `Trigger` cez zbernicu **USB**, ale aj cez sériovú linku **RS232**. Dosiahnuté výsledky sú v tabuľke 8.5. Úspešnosť udáva s akou pravdepodobnosťou sa podarilo vyvolať chybu. Vo zvyšných prípadoch prišlo k pádu PC, keďže automatický test je stavaný tak, že buď bude chyba na danom oneskorení vyvolaná, alebo príde k pádu. Oba testy dosiahli **100% úspešnosť**, čo potvrdzuje, že najlepšia frekvencia získaná z útoku PlunderVolt je vhodná aj na útok VoltPillager a preto iné ani neboli testované.

Oneskorenie (μs)	**Podvoltovanie**	**Úspešnosť**	**Trigger**
60, 85	od 190mV do 210mV	100%	USB
25	od 190mV do 210mV	100%	RS232

Tabuľka 8.5: Parametre podvoltovania a úspešnosť vyvolania chyby v násobení útoku VoltPillager pre CPU i5 6500 s frekvenciou nastavenou na 2GHz

8.2.6 Vyvolanie chyby v TEE a jej analýza

Na dešifrovanie RSA bol tiež rovno vykonaný automatický test úspešnosti, tentokrát na frekvencii **1.4GHz**, ktorá bola pri útoku PlunderVolt najúspešnejšia. Opäť bola dosiahnutá **100% úspešnosť** vyvolania chyby pre oba spôsoby odosielania signálu `Trigger`. Dokonca **100% úspešnosť** bola dosiahnutá na **získanie tajného kľúča** z vyvolanej chyby.

Dosiahnuté výsledky sú popísané v tabuľke 8.6

Skript na automatické testovanie bol rozšírený aj na šifrovanie AES. Úspešnosť vyvolania chyby bola nižšia oproti predchádzajúcim algoritmom, a dokonca vyvolanú chybu **nebolo**

Oneskorenie (μs)	Podvoltovanie	Celková úspešnosť	Trigger
10, 25, 30, 70	od 170mV do 190mV	100%	USB
10, 25, 35, 50, 55, 60, 85	od 170mV do 190mV	100%	RS232

Tabuľka 8.6: Parametre podvoltovania a úspešnosť vyvolania chyby v algoritme RSA pomocou útoku VoltPillager pre CPU i5 6500 s frekvenciou nastavenou na 1.4GHz

možné zneužiť. Na najlepších parametroch bolo dokonca vyvolaných ďalších 200 chýb, ale i tak nenastala chyba v 8.kole. Štatistiky o úspešnosti sú v tabuľke 8.7. V stĺpci úspešnosti sú uvedené 3 čísla ktoré vyjadrujú úspešnosť vyvolania chyby v AES/zamrznutie PC/celková úspešnosť, ktorá udáva úspešnosť vyvolania chyby, z ktorej bolo možné získať šifrovací kľúč.

Oneskorenie (μs)	Podvoltovanie	Celková úspešnosť	Trigger
20	od 130mV do 160mV	19%/80%/0%	USB
70	od 130mV do 160mV	33%/67%/0%	RS232

Tabuľka 8.7: Parametre podvoltovania a úspešnosť vyvolania chyby v algoritme AES pomocou útoku VoltPillager pre CPU i5 6500 s frekvenciou nastavenou na 1.1GHz

Súhrn

Po zaobstaraní všetkého potrebného vybavenia na útok, bol útočný hardvér zostavený podľa schémy zapojenia na obrázku 8.3. Na zmenu napätia CPU bolo najskôr nutné zistiť princíp zmeny napätia, nahrať správny firmvér do útočného hardvéru a pripojiť útočný hardvér k základnej doske. K pripojeniu k základnej doske bolo potrebné detekovať regulátor napätia a následne zbernicu SVID pomocou osciloskopu. Optimálna frekvencia na vyvolanie danej chyby už bola nájdená pomocou predchádzajúceho útoku, avšak tentokrát bolo hľadané aj ideálne oneskorenie na odoslanie príkazov podvoltovania, k čomu sa opäť využil vytvorený skript **test_PV_PV.sh**. V prípade vyvolania chyby v násobení a dešifrovaní RSA, sa podarilo nájsť parametre, ktoré mali **100% úspešnosť**, vrátanie extrakcii tajného kľúča RSA. Pri vyvolaní chyby v šifrovaní AES nastal ten istý problém, ako pri útoku PlunderVolt, čiže z vyvolanej chyby **nebolo** možné získať šifrovací kľúč.

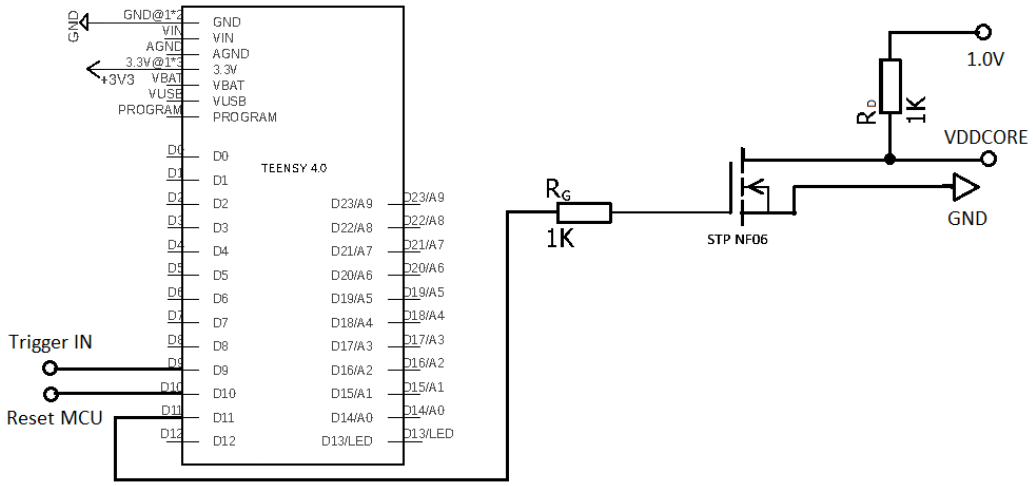
8.3 Replikácia útoku TrustZone-M(eh)

V implementácii útoku sa postupovalo podľa replikácie hardvérových útokov. K samotnému útoku je dostupná iba prezentácia, z ktorej všetky potrebné informácie sú popísané v sekcii 6.3. Čiže všetky zdrojové kódy k útoku v tomto prípade museli byť **vytvorené**. Krok analýzy vyvolanej chyby je v tomto prípade preskočený, pretože už pri samotnom vyvolaní chyby bude zrejmé, či sa útok podaril, alebo nie.

8.3.1 Potrebné vybavenie na útok

Hlavným prvkom vybavenia je mikrokontrolér **SAM L11**, na ktorý sa útočí. K tomu účelu bola zaobstaraná vývojová doska SAM L11 XPLAINED PRO[16], na ktorej sa priamo nachádza debugger pomocou ktorého je možné MCU programovať aj ladiť. V prezentácii sa

[16]Podrobnejšie informácie k doske: https://www.microchip.com/en-us/development-tool/dm320205



Obr. 8.8: Schéma zapojenia útočného hardvéru

spomína, že na riadenie útoku bolo použité FPGA s frekvenciou 100MHz, čo dokáže vytvárať dostatočne presné impulzy na podvoltovanie. V útoku VoltPillager bola však použitá doska **Teensy 4.0**, ktorá má MCU s frekvenciou **600MHz**. Takže namiesto spomínaného FPGA bola použitá táto doska, ktorá by mala dosiahnuť ešte väčšiu presnosť. Celý útočný hardvér sa skladá z nasledujúcich komponentov:

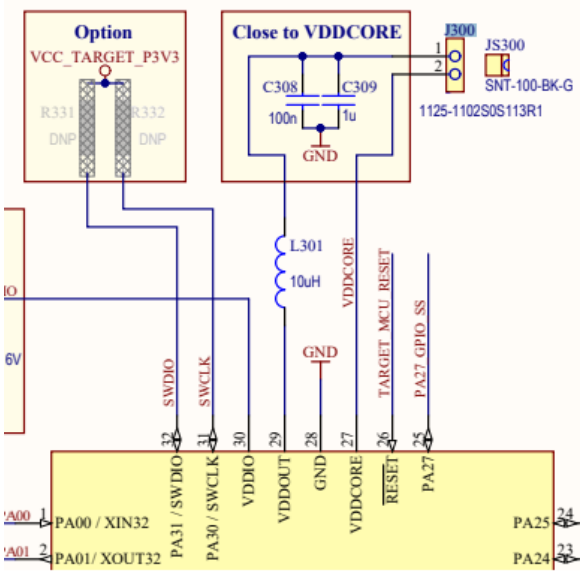
- Zdroj napätia
- Doska Teensy 4.0
- Tranzistor MOSFET N STP 16NF06
- Kontaktné pole s prepojkami
- Rezistory $1K\Omega$, 100Ω a 4.6Ω

8.3.2 Zostavenie útočného hardvéru

Základné prepojenie súčiastok vychádza z obrázku 6.7 získaného z prezentácie, avšak samotná schéma zapojenia k útoku nie je dostupná. Preto bola vytvorená **vlastná** (obrázok 8.8), ktorá pozostáva z MOSFET tranzistora, dosky Teensy a rezistorov. Rezistor R_G slúži pre obmedzenie maximálneho prúdu dosky Teensy. To je $4mA$, čiže hodnota rezistora musela byť vyššia ako: $R_G = \frac{U}{I} = \frac{3.3}{0.04} = 825\Omega$ rezistor R_D slúži ako "pullup" rezistor. `VDDCORE` je výstup, ktorý sa pripája k napätiu jadra MCU SAM L11. Na vstup `Trigger IN` odošle MCU SAM L11 logickú 1, ak sa podarí úspešne vykonať útok. `Reset MCU` riadi reštart mikrokontroléra SAM L11.

8.3.3 Princíp zmeny napätia

Príkazy na podvoltovanie priamo riadia napätie jadra MCU. Nenastavuje sa špeciálne napätie, iba sa odošle **zem** na určitý časový úsek. Na ochranu proti podvoltovaniu na pine



Obr. 8.9: Spôsob pripojenia kondenzátorov C308 a C309 k pinu VDDCORE, ktoré je potrebné odstrániť[19]

`VDDCORE`, obsahuje MCU SAM L11 detektor `BOD12`, ktorý pri zaznamenaní podvoltovania reštartuje mikrokontrolér. Avšak BOD12 na veľmi krátke impulzy (desiatky *ns*) **nedokáže reagovať**.

8.3.4 Vytvorenie a nahratie firmvéru na zmenu napätia

Útočný firmvér pre dosku Teensy na vyvolanie chyby násobenia **mul_err.ino**, vytvorený v prostredí Arduino vychádza z výpisu 7.2. Okrem vyvolania chyby je cieľom aj nájdenie **ideálnej šírky** impulzu. Na vyvolanie chyby, ktorá spôsobí, že nezabezpečený firmvér sa nahrá ako zabezpečený, bol vytvorený firmvér **boot_nonsec_as_sec.ino**, ktorý rozširuje pôvodný kód o resetovanie MCU SAM L11 po každom podvoltovaní a o jeden cyklus, ktorý hľadal ideálny čas vyvolania chyby po reštarte MCU.

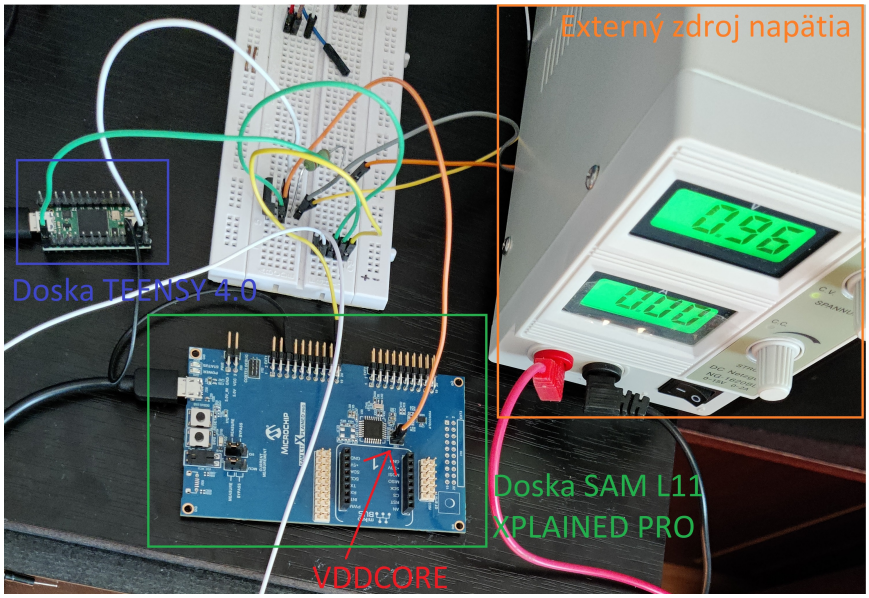
8.3.5 Pripojenie útočného HW k základnej doske/pinu MCU

Na pripojenie útočného hardvéru k MCU SAM L11 musel byť najskôr **detekovaný** pin `VDDCORE` pomocou technickej dokumentácie[17] k MCU. Následne museli byť detekované kondenzátory pripojené k pinu `VDDCORE`, ktoré musia byť odstránené. K tomu dopomohla schéma zapojenia k vývojovej doske SAM L11, ktorá je znázornená na obrázku 8.9. Kondenzátory nebolo potrebné fyzicky odstrániť, ale postačilo **odstrániť prepojku** `J300`, kde sa do pinu 2 pripojil útočný hardvér.

Pripojenie útočného harvéru k SAM L11 s využitím kontaktného poľa je znázornené na obrázku 8.10.

[17] Strana 1102: https://ww1.microchip.com/downloads/en/DeviceDoc/SAM-L10L11-Family-DataSheet-DS60001513F.pdf

[19] Schéma dostupná z (SAM L11 Xplained Pro Design Documentation): https://www.microchip.com/en-us/development-tool/dm320205



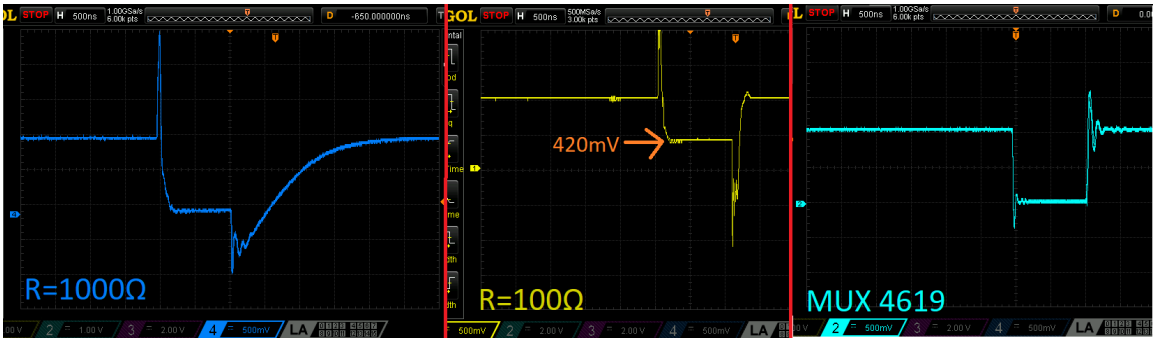
Obr. 8.10: Pripojenie útočného hardvéru k MCU SAM L11

8.3.6 Overenie náchylnosti na zmenu napätia

Na overenie náchylnosti na zmenu napätia sa použil typický príklad chyby násobenia, s odoslaním logickej 1 doske Teensy, ak sa podarí chybu vyvolať. Navyše bol kód rozšírený o blikanie LED každé 2 sekundy pri úspechu, pretože niekedy nastala situácia, že miesto vyvolania chyby prišlo k resetu mikrokotroléru a pravdepodobne sa vyvolala chyba v inicializácii MCU, čo spôsobilo, že bola odoslaná logická 1 doske Teensy, avšak blikanie LED s pravidelným intervalom nenastalo. K implementácii sa využilo **Atmel Studio**, kde už bol príklad[20], ako rozblikať LED na MCU SAM L11 s pravidelným intervalom. Čiže príklad bol iba doplnený o nekonečný cyklus s násobením a odosielanie logickej 1 za cyklom. Implementácia sa nachádza v súbore **led_flasher_main.c**, ktorým sa nahradí rovnako menovaný súbor v príklade `LED flasher`. Atmel studio navyše podporuje priamo dosku SAM L11 XPLAINED PRO, čiže je možné automaticky po preklade nahrať kód do mikrokontroléra.

S pripojením podľa schémy zapojenia zobrazenej na obrázku 8.8 a rozsahom šírky impulzu $10ns$ až $1000ns$ sa ani po niekoľkých hodinách **nepodarilo vyvolať chybu**. Preto bol odosielaný signál na pin `VDDCORE` analyzovaný pomocou osciloskopu. Už za rezistorom R_G odosielaný signál vyzeral úplne inak, ako mal. Keďže maximálny prúd bolo možné obmedziť aj na napájacom zdroji, rezistor bol odstránený. Stále však signál na pine `VDDCORE` nevyzeral ako bolo očakávané, preto bola šírka impulzu rozšírená až na $2\mu S$, čím sa odhalil problém. Ako možno vidieť vľavo na obrázku 8.11, už pri $2\mu S$, impulze trvá ďalšie $2\mu S$ kým sa vráti napätie `VDDCORE` na požadovanú hodnotu. Čiže pôvodne široký impulz napríklad $100ns$ je široký až $2100ns$, čo už bez problémov **zachytí detektor** `BOD12` a resetuje MCU. Keďže v prezentácii je uvedené zapojenie úplne bez rezistorov, bol odstránený aj druhý rezistor R_D, avšak podľa očakávaní v tomto prípade k podvoltovaniu ani neprišlo. Preto boli testované odpory R_D rôznej veľkosti, pomocou čoho sa zistilo, že čím je odpor `menší`, tým vrátenie na pôvodnú hodnotu trvá `kratšie`, avšak ak je odpor až moc malý

[20]Príklad možno zostrojiť podľa návodu z: http://ww1.microchip.com/downloads/en/Appnotes/Getting-Started-with-SAM%20L10L11-Xplained-Pro-DS00002722A.pdf



Obr. 8.11: Nameraný priebeh na výstupe útočného hardvéru pinu VDDCORE pri použití tranzistora a rezistora 1000Ω (vľavo, modrá farba), tranzistora a rezistora 100Ω (stred, žltá farba), multiplexora (vpravo tyrkysová farba)

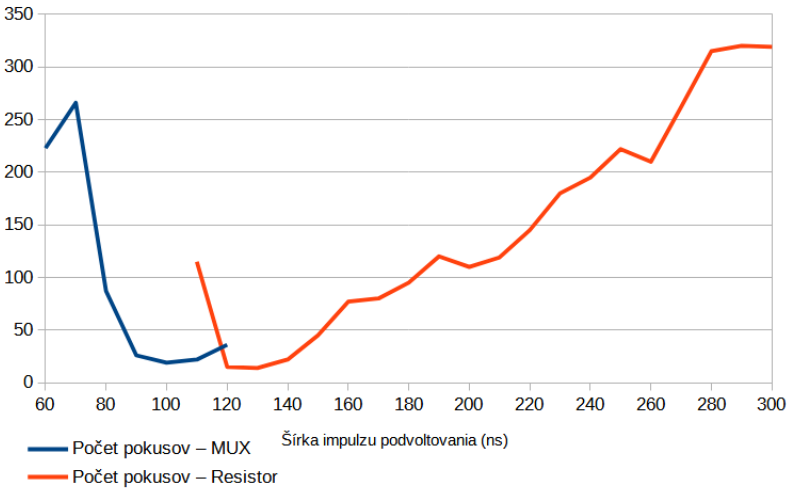
prichádza k podvoltovaniu iba o pár mV. Najlepšie vlastnosti vykazoval odpor 100Ω, kde prišlo k dostatočnému podvoltovaniu a zároveň k rýchlemu obnoveniu napätia na pôvodnú hodnotu. Avšak ani to nepomohlo, takže zostávala jediná možnosť a to použiť `multiplexor` MAX4619, ktorým sa nahradil tranzistor a pripojil sa k útočnému hardvéru podľa schémy zapojenia uvedenej v prílohe na obrázku D.2. Dosiahnutý priebeh pomocou multiplexora je zobrazený vpravo na obrázku 8.11 tyrkysovou farbou. Avšak ani multiplexor nepomohol, z čoho bolo usúdené, že problémom **nie je** nechcené predĺženie impulzu.

Pri ladení po jednotlivých krokoch kódu chyby multiplikácie bolo zistené, že do cyklu s násobením sa ani nedalo prejsť. Hodnota výsledku násobenia už bola dokonca vypočítaná a uložená do premennej pred samotným výpočtom. Program sa choval, ako keby vykonáva nekonečný prázdny cyklus. Preto sa aj chybu nepodarilo vyvolať, keďže nebolo akú. Riešením bolo `vypnutie optimalizácii`, ktoré boli pôvodne nastavené na úroveň `-O1`. Po opätovnom ladení po krokoch už všetko prebiehalo podľa očakávaní. Pomocou multiplexora MAX4619 sa podarilo vyvolať chybu už po pár sekundách. Po vyvolaní chyby, bola postupne optimalizovaná šírka impulzu. Pri šírke nad $120nS$ už prichádzalo k reštartu mikrokotroléra, čo značí, že podvoltovanie zaznamenal detektor `BOD12`. Preto bola skúmaná šírka impulzov od $2ns$, čo je minimum, ktoré dokáže doska Teensy generovať, po $120ns$. Následne boli testované aj verzie útočného hardvéru s tranzistorom a odpormi 100Ω a 1000Ω. S odporom 100Ω sa chyby **podarilo** takisto vyvolať s podobným počtom pokusov, ako pri multiplexore. Prekvapivé pri tomto nastavení bolo, že chyby boli vyvolané aj pri šírke impulzu $300ns$, kde predtým už pravidelne nastával reštart MCU. Dokonca ešte aj pri desiatkách mikrosekúnd sa podarilo vyvolať chybu pri väčšom počte pokusov. Je to spôsobené tým, že príde k podvoltovaniu iba na hodnotu približne $420mV$ (pri multiplexore na 0V), čo nedokáže detektor podvoltovania vôbec zachytiť. S odporom 1000Ω sa chybu ani dokonca **nepodarilo** vyvolať. To potvrdzuje, že čas, kým sa napätie jadra MCU dostane do normálu po podvoltovaní má na vyvolanie chyby zásadný vplyv.

Graf reprezentujúci počet pokusov potrebných na vyvolanie chyby pre multiplexor a 100Ω odpor je zobrazený na obrázku 8.12.

8.3.7 Aktivácia TEE

Na aktiváciu TEE sa použijú 2 samostatné programy, kde 1 je považovaný za bezpečný a druhý za nezabezpečený, ktoré budú následne prepojené. V Atmel Studio sa už nachádza hotový príklad zabezpečeného firmvéru `TrustZone-GetStart-S` a aj nezabezpečeného



Obr. 8.12: Graf reprezentujúci počet pokusov na úspešné vyvolanie chyby pomocou multiplexora alebo pomocou tranzistora s 100Ω rezistorom

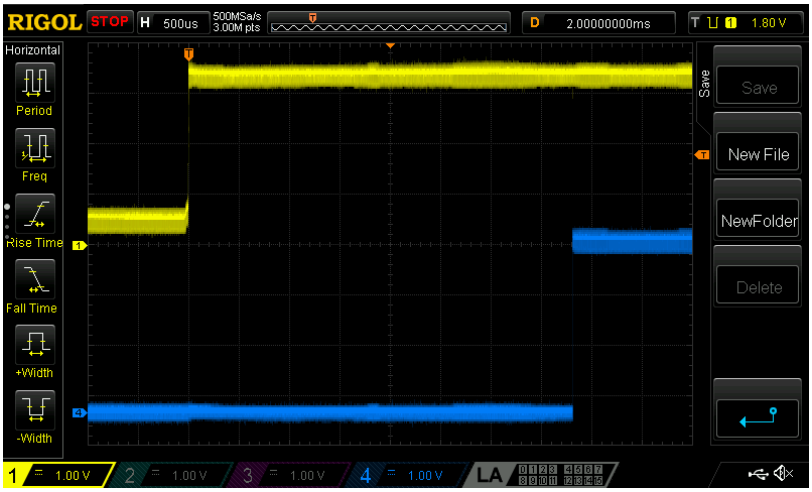
`TrustZone-GetStart-NS`. Na prepojenie týchto príkladov, ich správny preklad a nahratie bol použitý návod od microchipu k doske SAM L11 XPLAINED PRO[21]. Následne bolo otestované, či TEE naozaj funguje a to tým, že z nezabezpečenej časti bola použitá funkcia na rozsvietenie LED diódy. K rozsvieteniu naozaj neprišlo. Keď však bola vytvorená a potom zavolaná funkcia z časti NSC(non-secure callable), v ktorej bolo implementované rozsvietenie LED diódy, dióda sa podľa očakávaní rozsvietila.

8.3.8 Vyvolanie chyby v TEE

Cieľom bolo v príklade použitom v predchádzajúcej sekcii **rozblikať LED diódu** s pravidelným intervalom a **odoslať logickú 1** doske Teensy z pinu PA10, priamo z nezabezpečenej časti, čo je za normálnych okolností **nemožné**. Vytvorený zdrojový kód sa nachádza v priečinku `boot_nonsec_as_sec`, v ktorom sa nachádza súbor **main.c**, ktorým sa nahradí rovnako menovaný súbor vo vytvorenom príklade. Ako už bolo popísané v sekcii 6.3, docieli sa to tak že sa preskočí inštrukcia, ktorá číta veľkosť registra AS, ktorý určuje veľkosť zabezpečenej časti. To spôsobí, že jeho veľkosť zostane nulová a všetok kód bude v jednej sekcii. Čiže by malo byť možné pristupovať k akejkoľvek funkcii vrátane práce s I/O výstupmi, ktoré sú cieľom útoku.

Ako prvé bol určený časový úsek **od reštartu** MCU po **nastavenie logickej 1** na niektorom z výstupov mikrokontroléra. K tomuto účelu bol použitý osciloskop, ktorého meranie je zobrazené na obrázku 8.13 z čoho sa dá ľahko vyčítať, že tento čas je približne $3.8ms$. To znamená, že podvoltovanie musí nastať maximálne $3.8ms$ po reštarte. Stále to je však príliš veľký časový úsek, keďže inštrukcia beží pár desiatok nanosekúnd a navyše vyvolanie chyby sa nemusí podariť vyvolať na prvýkrát. Na určenie presnejšieho bodu injektovania chyby, bola použitá **výkonová analýza**.

[21]Dostupné z (strany 15-34): http://ww1.microchip.com/downloads/en/Appnotes/Getting-Started-with-SAM%20L10L11-Xplained-Pro-DS00002722A.pdf



Obr. 8.13: Zmeranie času pomocou osciloskopu od reštartu mikrokontroléra (žltý signál) po vykonanie prvej inštrukcie programu (nastavenie modrého signálu na logickú 1)

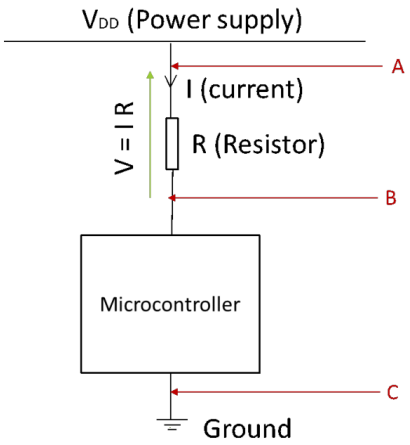
Výkonová analýza

Výkonová analýza sa použije na zmeranie spotreby pri čítaní čo **najväčšej** hodnoty registra AS a čo **najmenšej** hodnoty, pretože pri čítaní rozdielnych hodnôt registra by sa spotreba mala **líšiť**. Keďže všetko ostatné je v zdrojovom kóde nezmenené, najvýraznejší rozdiel v spotrebe určí presný moment, kedy prichádza k čítaniu hodnoty registra AS. Zápis hodnoty do registrov (zmena poistiek), ktoré určujú veľkosti zabezpečených a nezabezpečených častí sú uložené v súbore `trustzone_config`, kde bola zmenená hodnota makra `CONF_UROW_IDAU_AS`. Najväčšia a najmenšia hodnota registra AS, aby sa použitý kód zmestil do danej časti bola `0xF7` a `0x20`. Pomocou referenčnej príručky zabezpečenia SAM L11[22] bolo zistené, že okrem toho je potrebné nastaviť aj nové veľkosti zmenených pamätí v súboroch `saml11_nonsecure.ld` a `saml11_secure.ld`. V súbore `saml11_nonsecure.ld` sa iba nastavia parametre rom pamäte `ORIGIN` a `LENGTH` na hodnotu $AS * 0x100$. V súbore `saml11_secure.ld` sa nastaví parameter `LENGTH` pamäte rom a prameter `ORIGIN` pamäte rom_nsc na hodnotu $AS * 100 - ASNC * 0x20$. To je napríklad pri hodnote registra AS `0xF7` a hodnote registra ASNC `0x10`, hodnota `0xF500`.

Na výkonovú analýzu bola použitá **metóda bočníka**, to znamená, že sa zapojí nízkoohmový rezistor sériovo do obvodu so záťažou. Pomocou osciloskopu sa potom zmeria **úbytok napätia** na rezistore. Na tento účel sa bežne používa špeciálna diferenciálna sonda osciloskopu, ktorá sa pripojí k bodom A a B zobrazených na obrázku 8.14. Tá však nebola k dispozícii preto boli využité 2 bežné sondy, kde 1 sa pripája k bodom A a C a druhá k bodom B a C. Rozdiel nameraného napätia na sondách udáva úbytok napätia na bočníku. Na dosiahnutie čo najväčšej presnosti musí byť hodnota rezistora R čo **najmenšia**. Bolo otestovaných niekoľko hodnôt rezistora od 0.6Ω po 100Ω a bolo aj potvrdené, že s najmenším odporom, to je v tomto prípade 0.6Ω boli dosiahnuté najlepšie výsledky.

Doska SAM L11 XPLAINED PRO má na doske vývody, kde je možné pripojiť hardvér, ktorým sa vykonáva výkonová analýza, alebo v tomto prípade bočníkový odpor. Pre meranie spotreby I/O je to prepojka J102 a pre meranie spotreby MCU, čo je v tomto prípade

[22] Dostupná z: http://ww1.microchip.com/downloads/en/AppNotes/SAM-L11-Security-ReferenceGuide-AN-DS70005365A.pdf



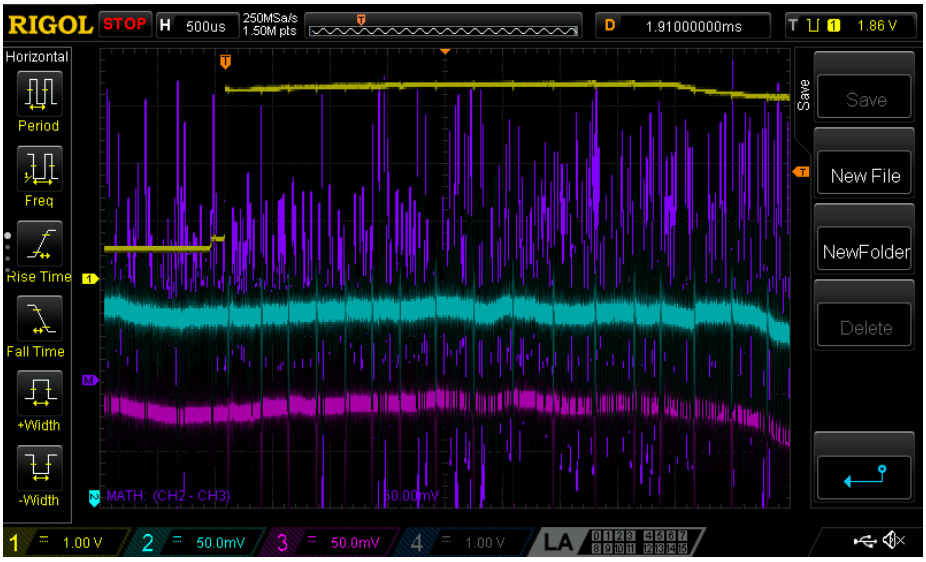
Obr. 8.14: Možnosti zapojenia osciloskopu na vykonanie výkonovej analýzy MCU pomocou bočníku [22]

podstatné, je to prepojka J103, ktorá sa odstráni a pripojí sa na ňu bočník. Okrem toho bolo ešte potrebné pripojiť k osciloskopu 1 sondu, ktorá sleduje, kedy príde k reštartu mikrokotroléra, čo slúží ako **spúšťač** merania spotreby. Výsledok z merania možno vidieť na obrázku 8.15.

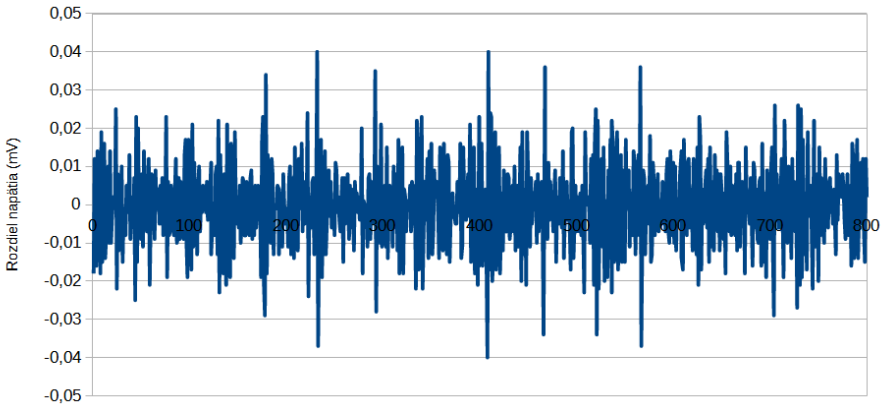
Celkovo bolo vykonaných **10 meraní** spotreby pri nastavení registra AS na hodnotu 0x20 a **10 meraní** spotreby pri nastavení registra AS na hodnotu 0xF7. Výsledky z meraní sa nachádzajú v priečinku `power_analysis`. Následne bola spriemerovaná spotreba pre obe hodnoty a bol vypočítaný rozdiel týchto priemerov, ktorý bol vykreslený do grafu, ktorý je na obrázku 8.16. Z grafu sa dá určiť **6 kandidátov**, kedy môže prichádzať k čítaniu hodnoty registra AS. Najlepší kandidát, ktorý bol testovaný ako prvý, je okolo vzorky čísla 400. Prichádza v tomto momente k najvyššiemu rozdielu spotreby a navyše aj susedné vzorky dosahujú väčší rozdiel v spotrebe, ako susedné vzorky iných kandidátov. Osciloskop vytvára vzorku každých $5\mu s$, to znamená, že okolie prvého kandidáta, kedy prichádza k výraznému rozdielu spotreby je približne $1900\mu s$ až $2300\mu s$ po reštarte MCU. Tieto parametre boli zadané do zdrojového kódu dosky Teensy, s krokom $1\mu s$. Najskôr bol vykonaný útok s použitím tranzistora s rezistorom 100Ω, pomocou čoho sa však **nepodarilo** vyvolať požadovanú chybu. Pri použití multiplexora sa po približne **90 minútach** podarilo poslať logickú 1 doske Teensy a rozblikať LED diódu. Čas po reštarte, kedy prišlo k správnemu vyvolaniu chyby bol $2180\mu s$. Toto okolie bolo ešte bližšie preskúmané s presnosťou na $2ns$, avšak chyba nastávala stále **náhodne** v intervale $2175\mu s$ až $2189\mu s$ po reštarte. Najlepšie parametre na vyvolanie chyby bol interval $2180\mu s$ až $2189\mu s$ s krokom $50ns$, čo vyvolalo požadovanú chybu v priemere každých **5 minút**.

Súhrn

Prvým krokom bolo zaobstaranie potrebného vybavenia a zostavenie útočného hardvéru podľa schémy zapojenia na obrázku 8.8. K zmene napätia bolo nutné zistiť, ako sa napätie mení, vytvoriť a nahrať firmvér do útočného hardvéru a pripojiť útočný hardvér k MCU SAM L11. Na overenie náchylnosti na zmenu napätia bol vytvorený a nahraný špeciálny firmvér pre MCU SAM L11 na vyvolanie chyby v násobení. Pôvodná schéma zapojenia



Obr. 8.15: Výkonová analýza pomocou osciloskopu. Kanál 1 (žltý signál) meria hodnotu na pine reštartu MCU, kanál 2 (tyrkysový signál) meria hodnotu napätia pred bočníkom, kanál 3 (ružový signál) meria napätia za bočníkom a fialový signál určuje úbytok napätia na bočníku.



Obr. 8.16: Výsledný graf z výkonovej analýzy na určenie zápisu hodnoty do registra AS

z obrázku 8.8 sa ukázala ako nevhodná a preto boli vytvorené 2 ďalšie možnosti (príloha D), ktorými sa už chybu podarilo vyvolať. Po aktivácii TEE bol vytvorený firmvér pre útočný hardvér na vyvolanie chyby v TEE. Na nájdenie oneskorenia, kedy má byť chyba vyvolaná bola využitá výkonová analýza, ktorou sa podarilo odhaliť čas vykonávania inštrukcie, ktorú je nutné podvoltovaním preskočiť. Tým sa pôvodný dôveryhodný kód stane nedôveryhodný a je možné jeho inštrukcie vykonávať aj z pôvodnej nedôveryhodnej časti. Na overenie, či sa túto chybu podarilo vyvolať bol vytvorený firmvér pre MCU SAM L11, kde sa nedôveryhodný kód pokúsi rozblikať LED, čo je za normálnych okolností zakázané. S najlepšími parametrami (to je oneskorenie 2180μs až 2189μs) prišlo ku blikaniu LED v priemere raz za 5 minút.

Kapitola 9

Diskusia

Diskusia je rozdelená na 3 časti, rovnako ako implementácia. V každej časti je prebrané, aké výsledky sa podarilo dosiahnuť v porovnaní s výsledkami, ktoré dosiahli tvorcovia útoku a možná typy obrán proti danému útoku.

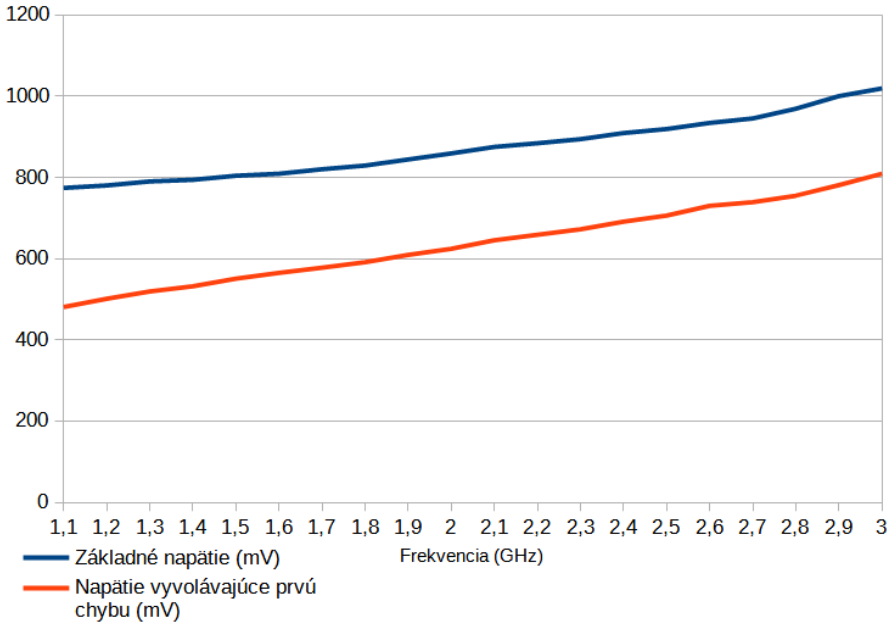
9.1 PlunderVolt

Softvérový útok PlunderVolt bol úspešne zreplikovaný na 2 rôznych CPU, dokonca každý inej generácie. Na CPU i5 6500 bol vykonaný aj automatický komplexný test a na CPU i5 7400 bolo overené, či je možné chybu vyvolať a na najúspešnejšej frekvencii bola vždy vyvolaná **20krát**. Výsledky sú však dosť **odlišné** oproti tomu, čo dosiahli tvorcovia útoku. Bod, kedy príde k úspešnému vyvolaniu chyby spôsoboval časté **pády**. Tento problém mal vyriešiť väčší počet **iterácií**, čím by sa znížila hodnota podvoltovania a nebola by už hraničná. Avšak už pri vyvolaní chyby násobenia zvýšenie počtu iterácií nemal **žiadny vplyv** na hodnotu podvoltovania. Dokonca pri rozsiahlom teste všetkých frekvencií 1000 iterácií dosiahlo väčšiu úspešnosť, ako 1000000 iterácií.

Pri chybe násobenia sa podarilo vyvolať požadovanú poruchu, to je preklopenie 1 bitu v násobení, čo spôsobilo, že nekonečný cyklus bol ukončený. Keďže na frekvenciách od 1.1GHz do 3.0GHz sa vždy podarilo vyvolať aspoň 1 chybu, bol vytvorený aj graf kedy prišlo k prvej chybe na danej frekvencii. Ako možno vidieť na obrázku 9.1 **zhoduje** sa s tým, čo dosiahli tvorcovia (obrázok 6.3), to znamená, že čím je vyššia frekvencia, tým je napätie podvoltovania takisto vyššie.

Útok na dešifrovanie algoritmom RSA bol takisto **úspešný** a podarilo sa získať tajný kľúč, avšak samotné vyvolanie chyby neznamenalo automaticky získanie dešifrovacieho kľúča. To bolo dosť závislé, od frekvencii na ktorej bola porucha vyvolaná. V niektorých prípadoch iba **10% chýb** sa dalo zneužiť, avšak tá najúspešnejšia (1.4GHz), ktorá vyvolala najviac porúch, vyvolávala až vyše **90% chýb**, ktorých sa dalo zneužiť. Celková úspešnosť bola preto stále dosť vysoká a predstavuje reálne nebezpečenstvo pre procesor, na ktorom bola chyba vyvolaná.

V útoku na šifrovanie AES boli dosiahnuté **neuspokojivé výsledky**. Nielenže vyvolanie chyby malo veľmi nízku úspešnosť aj po komplexnom testovaní, chybu, ktorou by sa dal získať šifrovací kľúč sa **nepodarilo vyvolať**. Preto bolo na najúspešnejších frekvenciách vykonaných ďalších vyše tisíc pokusov o vyvolanie poruchy, avšak stále ani 1 nevyvolal požadovanú chybu. Čiže z vyše 100 vyvolaných chýb bola stále **0% úspešnosť**, čo sa k úspešnosti 10% udávanej autormi ani nepribližuje. Boli však vyvolané chyby v **štvrtom**

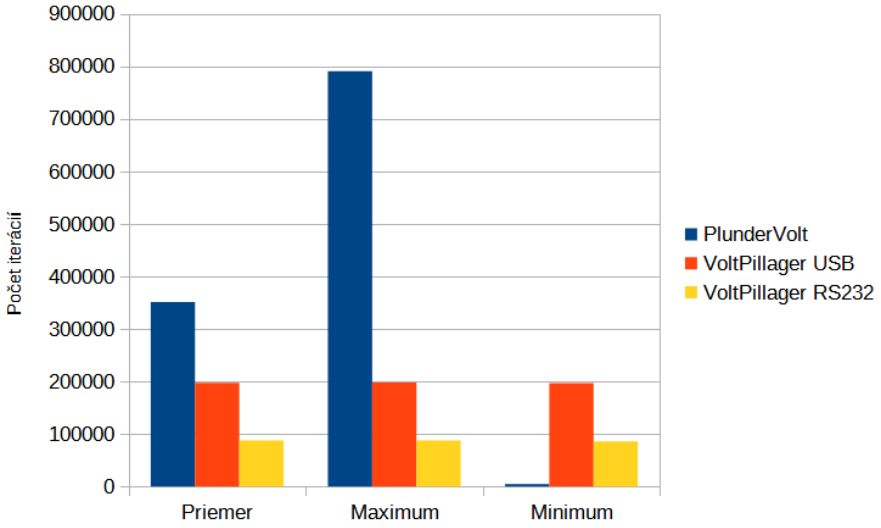


Obr. 9.1: Nameraný vzťah medzi frekvenciou, normálnym napätím (modrá) a potrebným podvolotovaním(oranžová) na dosiahnutie chybného výsledku násobenia pre procesor i5-6500

a **šiestom** kole, čiže teoreticky by bolo možné vyvolať chybu v požadovanom 8. kole. Celý test, ktorý bol dokonca neúspešný zabral vyše týždňa nepretržitého chodu PC so stovkami reštartov. Čiže je dosť nepravdepodobné, že by si obeť nevšimla zvláštne správanie PC a preto bolo usúdené, že tento útok na konkrétny procesor (i5 6500) nepredstavuje hrozbu. Toto chovanie však potvrdil aj druhý CPU, kde z 20 vyvolaných porúch takisto ani jednu sa **nedalo zneužiť**.

Obranné techniky

Útoku PlunderVolt je však možné veľmi jednoducho zabrániť. Stačí **aktualizovať BIOS** na verziu z roku 2019 a novšiu. V prípade CPU vyrobených od roku 2019 už ani to nie je potrebné, pretože tie podporujú iba verziu BIOSu, kde už je zavedená ochrana. Obranou je, že bola **zakázaná** zmena napätia CPU zápisom hodnoty do registra MSR. Okrem zákazu zápisu hodnoty do registra MSR, mohol Intel problém vyriešiť napríklad tým, že každá zmena napätia by bola po vstupe do enklávy automaticky **spätne škálovaná**. Pripadne by riešením mohlo byť, že pre každú frekvenciu by bolo povolené zmeniť napätie iba na overené a **bezpečné hodnoty**. Na úrovni softvéru je možné zvýšiť bezpečnosť vykonávaním kritických operácií **viackrát**. To znamená, že napríklad dešifrovanie RSA sa vykoná trikrát a porovná sa, či bol dosiahnutý vždy rovnaký výsledok. Stále je ale možné vyvolať chybu vo všetkých dešifrovaniach. Avšak je veľmi malá šanca vyvolať vo všetkých 3 dešifrovaniach tú istú chybu.



Obr. 9.2: Priemerný, maximálny a minimálny počet iterácií potrebných na vyvolanie poruchy v násobení CPU i5 6500 pri frekvencii 2.0GHz

9.2 VoltPillager

Hardvérový útok VoltPillager bol **úspešne** zreplikovaný na CPU i5 6500, kde bol opäť vykonaný aj komplexný test, ale tentokrát na nájdenie ideálnej hodnoty na oneskorenie, kedy odoslať príkaz na podvoltovanie. Tentokrát sú však dosiahnuté výsledky veľmi podobné tomu, čo dosiahli tvorcovia. I keď hodnota podvoltovania bola veľmi blízko k hodnote, kedy prichádzalo k zamrznutiu PC, pri ideálnych parametroch sa chybu podarilo vyvolať so **100% úspešnosťou** bez zamrznutia PC. A to aj pre **násobenie** a **dešifrovanie RSA**. Dokonca pri dešifrovaní RSA sa pri optimálnych parametroch podarilo vyvolať chybu, ktorú bolo možné **vždy zneužiť**. Čiže celková úspešnosť bola takisto 100%. Vyvolanie chyby v AES šifrovaní takisto dosahuje uspokojivú úspešnosť, ale **nepodarilo** sa vyvolať chybu, z ktorej bolo možné odvodiť šifrovací kľúč pomocou diferenciálnej chybovej analýzy. Čiže v prípade, že útok možno zaradiť do modelu hrozby, útok predstavuje reálne nebezpečenstvo iba pre algoritmus RSA.

Tvorcovia navyše tvrdia, že oproti útoku PlunderVolt, postačí **menšie** množstvo iterácií. To bolo **potvrdené**, ako ukazuje graf na obrázku 9.2, ktorý bol vytvorený na základe získaných dát. Navyše aj iterácie, v ktorých nastávala chyba pri útoku VoltPillager sa líšila maximálne o 2000. Pri útoku PlunderVolt, iterácia vyvolanej chyby bola čisto **náhodná** a rozdiely boli až v stovkách tisíc iterácii. Graf na obrázku 9.2 navyše aj potvrdzuje, že pri posielaní signálu `Trigger` cez seriovú linku RS232 postačí **výrazne menšie** množstvo iterácií.

Obranné techniky

Ochrana aplikovaná Intelom proti útoku PlunderVolt, kde bola zakázaná zmena zápisu hodnoty do regitra MSR je v tomto prípade neúčinná. Ani procesory najnovšej generácie **nie sú schopné** sa proti útoku brániť. Jedno z možných riešení, ktoré však nie je definitívne, je **šifrovanie komunikácie** na zbernici SVID. To by zabránilo reverznému inžinierstvu, ktorým sa podarilo zistiť príkazy na podvoltovanie na konkrétnu hodnotu. Avšak VR v podstate

konvertuje príkazy SVID na PWM signál, ktorý riadi tranzistory, ktorými je generované skutočné napätie jadra. Namiesto posielania príkazov na SVID by teda útočník mohol odpojiť SVID zbernicu a dodávať svoje **vlastné** (škodlivé) PWM signály, čo **obchádza** akúkoľvek autentifikáciu SVID. Dobre vybavený útočník by mohol dokonca úplne nahradiť VR vlastným. Riešením by však mohlo byť, ak by CPU **monitorovalo svoje napätie** a v prípade, že by kleslo pod bezpečnú hodnotu, **prerušilo** by vykonávanie v rámci enklávy. Procesory Intel síce podporujú monitorovanie napätia CPU (napríklad čítaním hodnoty registra MSR 0x198), ale všetky známe metódy majú **nízku vzorkovaciu frekvenciu**, ktorou by nebolo možné podvoltovanie vždy zachytiť. Čiže nové generácie CPU by mohli obsahovať špecializované obvody na sledovanie napätia z **vysokou obnovovacou frekvenciou**. Z pohľadu zákazníka definitívne riešenie nie je známe. Zvýšenie bezpečnosti je možné **redundanciou kritických operácií**, ako už bolo spomínané v obrane proti útoku PlunderVolt.

9.3 TrustZone-M(eh)

V implementácii sa podarilo dokázať, že útok je **zreplikovateľný** s podobným úspechom, ako v prezentácii k útoku. Pri vyvolaní chyby multiplikácie chyba nastávala takmer **okamžite**, avšak pri vyvolaní chyby v TEE bola úspešnosť **mierne horšia**. V prezentácii pri demonštrovaní vyvolania tejto chyby, nastane chyba po pár sekundách. V niektorých prípadoch sa takisto podarilo preskočiť požadovanú inštrukciu do pár sekúnd, ale v priemere to bolo aj pri najlepších parametroch až **5 minút**. To však autor neuvádza, či je to najlepší výsledok, alebo bežný. Pôvodne bolo zamýšľané, že to môže byť spôsobené tým, že autorovi sa podarilo nájsť čas vyvolania chyby od reštartu s presnosťou na $10ns$. To sa však nepotvrdilo, pretože doska Teensy by dokázala nájsť čas injektovania chyby s presnosťou $2ns$, keďže interval bol už dostatočne zúžený výkonovou analýzou. Stále sa však chyba vyskytovala náhodne v intervale od $2175\mu s$ po $2189\mu s$ a dokonca presnosť na $2ns$ nedosahovala najlepšie výsledky.

Výsledok z výkonovej analýzy **nie je** úplne presvedčivý. To môže byť spôsobené **metódou** výkonovej analýzy, **nedostatočným počtom meraní** alebo **nízkou frekvenciou vzorkovania**. Metóda by nemala spôsobovať problém, pretože tá bola úspešne použitá napríklad aj na dešifrovanie AES pri rôznych MCU [22], kde sa požaduje určite väčšia presnosť. Málo meraní bolo uskutočnených z dôvodu, že najrýchlejšia zistená možnosť, ako ukladať výsledky z osciloskopu bolo každé jedno meranie vždy **ručne uložiť na USB** vo formáte csv. Testovaná bola aj vyššia vzorkovacia frekvencia, s meraním napätia každých $200ns$, avšak výsledky boli ešte o niečo horšie. V každom prípade by bolo určite lepšie použiť priamo **výkonový analyzátor**, ktorým by bolo možné behom chvíle spraviť stovky meraní a tie spriemerovať. Napríklad Keil ULINKplus je podporovaný aj s použitou doskou SAM L11 XPLAINED PRO, kde sa nachádza priamo konektor na jeho pripojenie. Dokonca aj vývojová doska má vstavaný výkonový analyzátor, ktorého dáta sa dajú zobraziť v Atmel Studio, avšak pri meraní spotreby mikrokontroléra dokázal analyzátor zmerať maximálne 15000 vzorkov za sekundu. To je iba 1 vzorka každých približne $67\mu s$, čo je **nedostatočné**.

Obranné techniky

Softvérová obrana proti tomuto útoku na úrovni užívateľa **nie je možná**, pretože čítanie hodnoty registra AS sa deje počas **bootovania ROM** pamäte. Čiže čas a spôsob zápisu hodnoty do registra AS sa nedá ovplyvniť a spôsob ani nie je známy. Čiže zvýšiť bezpečnosť Mikrokontroléra môže jedine **výrobca**. Na zvýšenie bezpečnosti by mohol výrobca realizo-

vať čítanie hodnoty registra AS viackrát a v **náhodnom čase**. Následne by sa zistilo, či sa prečítané hodnoty v registri zhodujú. Stále však pri spravení kvalitnej výkonovej analýzy a veľkom množstve pokusov by bolo možné útok zrealizovať. Niektoré MCU s TrustZone-M, ako napríklad NXP LPC55s69 používa na prechod zo zabezpečeného sveta do nezabezpečeného špeciálnu inštrukciu **BLXNS**, ktorá zabraňuje neúmyselným skokom do nezabezpečeného sveta. Útok na tento MCU je možné stále vykonať, ale je to oveľa **náročnejšie**, keďže je nutné preskočiť aj túto inštrukciu. Na úrovni návrhu čipu by bolo riešením nevyviesť pin `VDDCORE`, na ktorý sa posielajú príkazy na podvoltovanie. Avšak to by spôsobilo, že by bolo napätie jadra MCU nestabilné a bolo by nutné pridať vyhladzovacie kondenzátory priamo do návrhu. Stále však by bolo možné použiť na preskočenie požadovaných inštrukcií injektovanie chyby napríklad pomocou **elektromagnetického impulzu**.

Kapitola 10

Záver

Cieľom práce bolo zreplikovať útoky, ktoré zmenou napätia vyvolávajú chyby vo výpočtoch CPU a MCU. Úspešne sa podarilo zreplikovať softvérový útok PlunderVolt a hardvérový útok VoltPillager na CPU Intel i5 6500. Okrem toho bol zreplikovaný aj hardvérový útok TrustZone-M(eh) na MCU SAM L11. K úspešnému zreplikovaniu bolo nutné najskôr naštudovať princíp útokov vyvolávajúcich chyby a prostredie dôveryhodného vykonávania, čo je ochrana CPU a MCU, ktorá sa používa na zabránenie primárne hardvérovým útokom. Avšak útoky, ktoré boli zreplikované, úspešne obišli toto prostredie.

Cieľom útokov PlunderVolt a VoltPillager je vyvolanie chyby preklopením bitov v šifrovacích algoritmoch RSA a AES, ktoré bežali v rámci prostredia dôveryhodného vykonávania Intel SGX. Zneužitie tejto chyby najskôr vyžadovalo naštudovanie samotných šifrovacích algoritmov a analyzačných techník na získanie tajného kľúča. Oba útoky boli úspešné iba v prípade dešifrovania RSA.

Hardvérovým útokom TrustZone-M(eh) sa dokonca podarilo obísť celé prostredie dôveryhodného vykonávania TrustZone-M. Zámerom bolo injektovať chybu, ktorou sa podarí preskočiť inštrukciu, ktorá určuje veľkosť zabezpečenej časti prostredia TrustZone-M. To spôsobilo, že jej veľkosť bola nulová a všetok kód sa preto nachádzal v nezabezpečenej časti. To znamenalo, že nezabezpečený kód mohol pristupovať ku všetkým funkciám pôvodne zabezpečeného kódu.

Posledná časť práce tvorí diskusia týchto útokov, kde sú podrobne prebrané dosiahnuté výsledky a návrhy na možnú ochranu proti týmto útokom. Tie nebolo možné aj implementovať, pretože tie účinné sú na úrovni hardvéru alebo mikrokódu, čiže ostáva na výrobcovi niektoré z nich implementovať.

Prácu by bolo možné rozšíriť testom ďalších procesorov Intel a pokúsiť sa vyvolať chybu v šifrovaní AES, ktorou by bolo možné získať šifrovací kľúč, pretože nebolo potvrdené, že je to možné. Replikácia útoku TrustZone-M(eh) na MCU SAM L11 tvorí dobrý základ toho, ako vyvolávať chyby v MCU s architektúrou ARM. V budúcnosti by bolo možné vykonať útoky na MCU Nuvoton NuMicro M2351 alebo NXP LPC55S69, ktoré využívajú podobný princíp, ako útok na MCU SAM L11. Tieto mikrokontroléry už majú implementovanú jednoduchú obranu proti týmto útokom, avšak túto ochranu je možné obísť. Podobný základ majú aj útoky na mikrokontroléry hardvérových peňaženiek Trezor One alebo Ledger Nano S, čím je možné sa postupne dopracovať až k získaniu PIN kódu k peňaženke.

Literatúra

[1] Abdullah, A. Advanced Encryption Standard (AES) Algorithm to Encrypt and Decrypt Data. Jún 2017, [cit. 2021-12-10].

[2] Ali, S. S. a Mukhopadhyay, D. Differential Fault Analysis of AES-128 Key Schedule Using a Single Multi-byte Fault. In: Prouff, E., ed. *10th Smart Card Research and Advanced Applications (CARDIS)*. Leuven, Belgium: Springer, September 2011, LNCS-7079, s. 50–64 [cit. 2021-12-15]. Smart Card Research and Advanced Applications. DOI: 10.1007/978-3-642-27257-8_4. Part 2: Invasive Attacks. Dostupné z: https://hal.inria.fr/hal-01596300.

[3] Ambedkar, B. a Bedi, S. A New Factorization Method to Factorize RSA Public Key Encryption. *International Journal of Computer Science Issues*. November 2011, zv. 8, [cit. 2021-12-03].

[4] Anati, I., Gueron, S., Johnson, S. a Scarlata, V. Innovative Technology for CPU Based Attestation and Sealing. In:. 2013 [cit. 2022-05-11]. Dostupné z: https://www.intel.com/content/www/us/en/developer/articles/technical/innovative-technology-for-cpu-based-attestation-and-sealing.html.

[5] ARamakrishnan, V., Venugopal, P. a Mukherjee, T. Proceedings of the International Conference on Information Engineering. In:. Association of Scientists, Developers and Faculties (ASDF), 2015, s. 9 [cit. 2021-11-15]. ISBN 9788192974279. Dostupné z: https://books.google.cz/books?id=Gw9pCwAAQBAJ&pg=PA9&redir_esc=y#v=onepage&q&f=false.

[6] Arfaoui, G., Gharout, S. a Traoré, J. Trusted Execution Environments: A Look under the Hood. In: *2014 2nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*. 2014, s. 259–266 [cit. 2022-03-17]. DOI: 10.1109/MobileCloud.2014.47.

[7] Asjad, S. The RSA Algorithm. December 2019, [cit. 2021-12-01].

[8] Awang, N. F. B. Trusted computing - opportunities amp; risks. In: *2009 5th International Conference on Collaborative Computing: Networking, Applications and Worksharing*. 2009, s. 1–5 [cit. 2022-03-15]. DOI: 10.4108/ICST.COLLABORATECOM2009.8402.

[9] Bagheri, N., Ebrahimpour, R. a Ghaedi Bardeh, N. New differential fault analysis on PRESENT. *EURASIP Journal on Applied Signal Processing*. December 2013, zv. 2013, s. 145–, [cit. 2021-12-01]. DOI: 10.1186/1687-6180-2013-145.

[10] Baksi, A. et al. *Fault Attacks In Symmetric Key Cryptosystems* [online]. 2020 [cit. 2021-11-17]. Dostupné z: https://eprint.iacr.org/2020/1267.pdf.

[11] Barenghi, A., Bertoni, G., Parrinello, E. a Pelosi, G. Low Voltage Fault Attacks on the RSA Cryptosystem. In:. September 2009, s. 23–31 [cit. 2021-11-16]. DOI: 10.1109/FDTC.2009.30.

[12] Benot, O. *Fault Attack*. Encyclopedia of Cryptography and Security, 2011 [cit. 2021-11-12]. ISBN 978-1-4419-5905-8.

[13] Biham, E. a Shamir, A. Differential fault analysis of secret key cryptosystems. In: Kaliski, B. S., ed. *Advances in Cryptology — CRYPTO '97*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, s. 513–525 [cit. 2021-12-15]. ISBN 978-3-540-69528-8.

[14] Blömer, J. a Seifert, J.-P. *Fault based cryptanalysis of the Advanced Encryption Standard* [Cryptology ePrint Archive, Report 2002/075]. 2002 [cit. 2021-11-28]. Dostupné z: https://ia.cr/2002/075.

[15] Boneh, D., Demillo, R. a Lipton, R. On the Importance of Checking Computations. Júl 1997, [cit. 2021-12-18].

[16] Breier, J., Hou, X. a Liu, Y. Fault Attacks Made Easy: Differential Fault Analysis Automation on Assembly Code. *IACR Transactions on Cryptographic Hardware and Embedded Systems*. May 2018, zv. 2018, č. 2, s. 96–122, [cit. 2021-12-17]. DOI: 10.13154/tches.v2018.i2.96-122. Dostupné z: https://tches.iacr.org/index.php/TCHES/article/view/876.

[17] Cai, Y., Ghose, S., Luo, Y., Mai, K., Mutlu, O. et al. Vulnerabilities in MLC NAND Flash Memory Programming: Experimental Analysis, Exploits, and Mitigation Techniques. In: *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 2017, s. 49–60 [cit. 2021-11-10]. DOI: 10.1109/HPCA.2017.61.

[18] Chen, Z., Vasilakis, G., Murdock, K., Dean, E., Oswald, D. et al. VoltPillager: Hardware-based fault injection attacks against Intel SGX Enclaves using the SVID voltage scaling interface. In: *30th USENIX Security Symposium (USENIX Security 21)*. Vancouver, B.C.: USENIX Association, August 2021 [cit. 2021-10-20]. Dostupné z: https://www.usenix.org/conference/usenixsecurity21/presentation/chen-zitai.

[19] Condé, R. C. R., Maziero, C. A. a Will, N. C. Using Intel SGX to Protect Authentication Credentials in an Untrusted Operating System. In: *2018 IEEE Symposium on Computers and Communications (ISCC)*. 2018, s. 00158–00163 [cit. 2022-03-18]. DOI: 10.1109/ISCC.2018.8538470.

[20] Daemen, J. a Rijmen, V. *The Design of Rijndael:AES — The Advanced Encryption Standard*. Springer, Berlin, Heidelberg, 2002 [cit. 2021-12-03]. ISBN 978-3-642-07646-6.

[21] Elbeltagy, M. *StegoCrypt3D: 3D Object and Blowfish*. Dizertačná práca.

[22] Gamaarachchi, H. a Ganegoda, H. *Power Analysis Based Side Channel Attack.* arXiv, 2018 [cit. 2022-04-27]. DOI: 10.48550/ARXIV.1801.00932. Dostupné z: https://arxiv.org/abs/1801.00932.

[23] Gueron, S. A Memory Encryption Engine Suitable for General Purpose Processors. *IACR Cryptol. ePrint Arch.* 2016, zv. 2016, s. 204, [cit. 2021-12-17].

[24] Hua, Z., Gu, J., Xia, Y., Chen, H., Zang, B. et al. vTZ: Virtualizing ARM TrustZone. In: *26th USENIX Security Symposium (USENIX Security 17).* Vancouver, BC: USENIX Association, August 2017, s. 541–556 [cit. 2022-03-20]. ISBN 978-1-931971-40-9. Dostupné z: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/hua.

[25] Jakub, B., Jap, D. a Chen, C.-N. Laser Profiling for the Back-Side Fault Attacks: With a Practical Laser Skip Instruction Attack on AES. *CPSS 2015 - Proceedings of the 1st ACM Workshop on Cyber-Physical System Security, Part of ASIACCS 2015.* Apríl 2015, s. 99–103, [cit. 2021-11-30]. DOI: 10.1145/2732198.2732206.

[26] Jovanovic, P., Kreuzer, M. a Polian, I. A Fault Attack on the LED Block Cipher. In: Schindler, W. a Huss, S. A., ed. *Constructive Side-Channel Analysis and Secure Design.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, s. 120–134 [cit. 2021-11-20]. ISBN 978-3-642-29912-4.

[27] Kenjar, Z., Frassetto, T., Gens, D., Franz, M. a Sadeghi, A. V0LTpwn: Attacking x86 Processor Integrity from Software. *CoRR.* 2019, abs/1912.04870, [cit. 2021-10-22]. Dostupné z: http://arxiv.org/abs/1912.04870.

[28] Kim, Y., Daly, R., Kim, J., Fallin, C., Lee, J. H. et al. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. In: *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA).* 2014, s. 361–372 [cit. 2021-10-25]. DOI: 10.1109/ISCA.2014.6853210.

[29] Korkikian, R. *Side-channel and fault analysis in the presence of countermeasures : tools, theory, and practice.* 2016. [cit. 2021-11-17]. Dizertačná práca. Université Paris sciences et lettres. Dostupné z: https://tel.archives-ouvertes.fr/tel-01762404/document.

[30] Korkikian, R., Pelissier, S. a Naccache, D. Blind Fault Attack against SPN Ciphers. In: *2014 Workshop on Fault Diagnosis and Tolerance in Cryptography.* 2014, s. 94–103 [cit. 2021-11-28]. DOI: 10.1109/FDTC.2014.19.

[31] Küçük, K. A., Paverd, A., Martin, A., Asokan, N., Simpson, A. et al. Exploring the use of Intel SGX for Secure Many-Party Applications. In:. December 2016, s. 1–6 [cit. 2022-03-17]. DOI: 10.1145/3007788.3007793.

[32] Lee, H., Lee, K. a Shin, Y. AES Implementation and Performance Evaluation on 8-bit Microcontrollers. *CoRR.* 2009, abs/0911.0482, [cit. 2021-12-10]. Dostupné z: http://arxiv.org/abs/0911.0482.

[33] Lenstra, A. K. Memo on RSA signature generation in the presence of faults. 1996, [cit. 2021-11-30]. manuscript. Dostupné z: http://infoscience.epfl.ch/record/164524.

[34] Martinásek, Z. *Kryptoanalýza postrannímí kanály.* 2013. [cit. 2021-11-14]. 129 s. Dizertačná práca. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií. Dostupné z: https://www.vut.cz/www_base/zav_prace_soubor_verejne.php?file_id=62844.

[35] Metula, E. Chapter 9 - Defending against MCRs. In: Metula, E., ed. *Managed Code Rootkits.* Boston: Syngress, 2011, s. 261–290 [cit. 2022-03-15]. DOI: https://doi.org/10.1016/B978-1-59749-574-5.00009-X. ISBN 978-1-59749-574-5. Dostupné z: https://www.sciencedirect.com/science/article/pii/B978159749574500009X.

[36] Minni, R., Sultania, K., Mishra, S. a Vincent, D. R. An algorithm to enhance security in RSA. In: *2013 Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT).* 2013, s. 1–4 [cit. 2021-12-02]. DOI: 10.1109/ICCCNT.2013.6726517.

[37] Mohamed, A. A. a Madian, A. H. A modified Rijndael algorithm and it's implementation using FPGA. In: *2010 17th IEEE International Conference on Electronics, Circuits and Systems.* 2010, s. 335–338 [cit. 2021-12-03]. DOI: 10.1109/ICECS.2010.5724521.

[38] Moro, N., Dehbaoui, A., Heydemann, K., Robisson, B. a Encrenaz, E. Electromagnetic Fault Injection: Towards a Fault Model on a 32-bit Microcontroller. In: *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography.* 2013, s. 77–88 [cit. 2021-11-28]. DOI: 10.1109/FDTC.2013.9.

[39] Moro, N., Heydemann, K., Dehbaoui, A., Robisson, B. a Encrenaz, E. Experimental evaluation of two software countermeasures against fault attacks. In: *2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST).* 2014, s. 112–117 [cit. 2021-11-29]. DOI: 10.1109/HST.2014.6855580.

[40] Murdock, K., Oswald, D., Garcia, F. D., Van Bulck, J., Gruss, D. et al. Plundervolt: Software-based Fault Injection Attacks against Intel SGX. In: *41st IEEE Symposium on Security and Privacy (S&P'20).* 2020 [cit. 2021-10-25].

[41] Ngabonziza, B., Martin, D., Bailey, A., Cho, H. a Martin, S. TrustZone Explained: Architectural Features and Use Cases. In:. November 2016, s. 445–451 [cit. 2022-03-23]. DOI: 10.1109/CIC.2016.065.

[42] Otto, M. *Fault Attacks and Countermeasures.* 2004. [cit. 2021-11-13]. Dizertačná práca. Universität Paderborn, Fakultät für Elektrotechnik, Informatik und Mathematik. Dostupné z: https://digital.ub.uni-paderborn.de/ubpb/urn/urn:nbn:de:hbz:466-20040101308.

[43] Padate, R., Patel, A. a Rodrigues, C. Encryption and Decryption of Text using AES Algorithm. In:. 2014 [cit. 2021-12-11].

[44] Pinto, S. a Santos, N. Demystifying Arm TrustZone: A Comprehensive Survey. *ACM Comput. Surv.* New York, NY, USA: Association for Computing Machinery. jan 2019, zv. 51, č. 6, [cit. 2022-03-23]. DOI: 10.1145/3291047. ISSN 0360-0300. Dostupné z: https://doi.org/10.1145/3291047.

[45] Pinto, S., Araujo, H., Oliveira, D., Martins, J. a Tavares, A. Virtualization on TrustZone-Enabled Microcontrollers? Voilà! In: *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 2019, s. 293–304 [cit. 2022-03-20]. DOI: 10.1109/RTAS.2019.00032.

[46] Piret, G. a Quisquater, J.-J. A Differential Fault Attack Technique against SPN Structures, with Application to the AES and Khazad. In: Walter, C. D., Koç, Ç. K. a Paar, C., ed. *Cryptographic Hardware and Embedded Systems - CHES 2003*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, s. 77–88 [cit. 2021-11-15]. ISBN 978-3-540-45238-6.

[47] Qiu, P., Wang, D., Lyu, Y. a Qu, G. VoltJockey: Breaching TrustZone by Software-Controlled Voltage Manipulation over Multi-Core Frequencies. In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. New York, NY, USA: Association for Computing Machinery, 2019, s. 195–209 [cit. 2022-04-20]. CCS '19. DOI: 10.1145/3319535.3354201. ISBN 9781450367479. Dostupné z: https://doi.org/10.1145/3319535.3354201.

[48] Rivest, R. L., Shamir, A. a Adleman, L. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Commun. ACM*. New York, NY, USA: Association for Computing Machinery. feb 1978, zv. 21, č. 2, s. 120–126, [cit. 2021-12-03]. DOI: 10.1145/359340.359342. ISSN 0001-0782. Dostupné z: https://doi.org/10.1145/359340.359342.

[49] Rivière, L., Najm, Z., Rauzy, P., Danger, J.-L., Bringer, J. et al. High precision fault injections on the instruction cache of ARMv7-M architectures. In: *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. 2015, s. 62–67 [cit. 2021-11-29]. DOI: 10.1109/HST.2015.7140238.

[50] Roscian, C., Sarafianos, A., Dutertre, J.-M. a Tria, A. Fault Model Analysis of Laser-Induced Faults in SRAM Memory Cells. In: *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography*. 2013, s. 89–98 [cit. 2021-11-28]. DOI: 10.1109/FDTC.2013.17.

[51] Shepherd, C., Akram, R. N. a Markantonakis, K. Establishing Mutually Trusted Channels for Remote Sensing Devices with Trusted Execution Environments. In:. August 2017 [cit. 2022-03-16]. DOI: 10.1145/3098954.3098971.

[52] Tang, A., Sethumadhavan, S. a Stolfo, S. CLKSCREW: Exposing the Perils of Security-Oblivious Energy Management. In: *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, 2017, s. 1057–1074 [cit. 2021-11-09]. ISBN 978-1-931971-40-9. Dostupné z: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/tang.

[53] Tunstall, M., Mukhopadhyay, D. a Subidh Ali, S. Differential Fault Analysis of the Advanced Encryption Standard Using a Single Fault. In:. Január 2011, s. 224–233 [cit. 2021-12-18].

[54] Wang, J., Hong, Z., Zhang, Y. a Jin, Y. Enabling Security-Enhanced Attestation With Intel SGX for Remote Terminal and IoT. *IEEE Transactions on*

Computer-Aided Design of Integrated Circuits and Systems. 2018, zv. 37, č. 1, s. 88–96, [cit. 2022-05-11]. DOI: 10.1109/TCAD.2017.2750067.

[55] Wu, C.-H., Hong, J.-H. a Wu, C.-W. RSA cryptosystem design based on the Chinese remainder theorem. In: *Proceedings of the ASP-DAC 2001. Asia and South Pacific Design Automation Conference 2001 (Cat. No.01EX455)*. 2001, s. 391–395 [cit. 2021-12-02]. DOI: 10.1109/ASPDAC.2001.913338.

[56] Yuce, B., Schaumont, P. a Witteman, M. Fault Attacks on Secure Embedded Software: Threats, Design and Evaluation. *CoRR*. 2020, abs/2003.10513, [cit. 2021-11-09]. Dostupné z: https://arxiv.org/abs/2003.10513.

[57] Zussa, L., Dutertre, J.-M., Clediere, J. a Robisson, B. Analysis of the fault injection mechanism related to negative and positive power supply glitches using an on-chip voltmeter. In: *2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*. 2014, s. 130–135 [cit. 2021-11-17]. DOI: 10.1109/HST.2014.6855583.

Príloha A

Obsah priloženého pamäťového média

```
/
├── auto_reset_PC_RPI
│   ├── Readme.md
│   ├── reset_PC.py
│   └── scheme_reset.png
├── PlunderVolt
│   ├── data
│   ├── Readme.md
│   └── RSA_success.py
├── test_scipt
│   ├── Readme.md
│   └── test_PV_VP.sh
├── thesis
│   ├── thesis.pdf
│   └── thesis.zip
├── TrustZone_meh
│   ├── Arduino
│   ├── Atmel_studio
│   ├── data_power_analysis
│   ├── data_mul_error.ods
│   ├── Readme.md
│   ├── tz_meh_hw_scheme.png
│   └── tz_meh_hw_scheme_2.png
└── VoltPillager
    ├── data
    └── Readme.md
```

- Adresár **auto_reset_PC_RPI** obsahuje skript, na automatický reštart PC pomocou Raspberry PI 4.0 pri zamrnutí. Okrem toho, adresár obsahuje návod na pripojenie Raspberry PI k PC vrátane schémy zapojenia.
- Adresár **PlunderVolt** obsahuje namerané dáta z útoku PlunderVolt a skript na získanie úspešnosti extrakcie kľúča z dát chýb vyvolaných v dešifrovaní RSA.

- Adresár **test_scipt** obsahuje skript na vykonanie útokov VoltPillager a PlunderVolt s využitím zdrojových kódov k útoku. Skript navyše zbiera dáta o úspešnosti a ukladá vyvolané chyby do súborov.
- Adresár **thesis** obsahuje text diplomovej práce a zdrojové súbory potrebné na jej vytvorenie.
- Adresár **TrustZone_meh** obsahuje:
  - Podadresár **Arduino** obsahujúci útočné firmvéry na vyvolanie chyby násobenia a vyvolanie chyby, ktorá spôsobí že zabezpečený kód je považovaný za nezabezpečený.
  - Podadresár **Atmel_studio** obsahuje firmvéry pre SAM L11 na overenie, či sa požadované chyby podarilo úspešne vyvolať útočným hardvérom.
  - Podadresár **data_power_analysis** obsahuje dáta z výkonovej analýzy
  - Súbor **data_mul_error.ods** obsahuje dáta získané z vyvolania chyby násobenia v útoku TrustZone-M(eh).
  - Návod na preloženie zdrojových kódov a zostavenie útočného hardvéru pomocou priložených schém zapojenia.
- Adresár **VoltPillager** obsahuje namerané dáta z útoku VoltPillager.

Príloha B

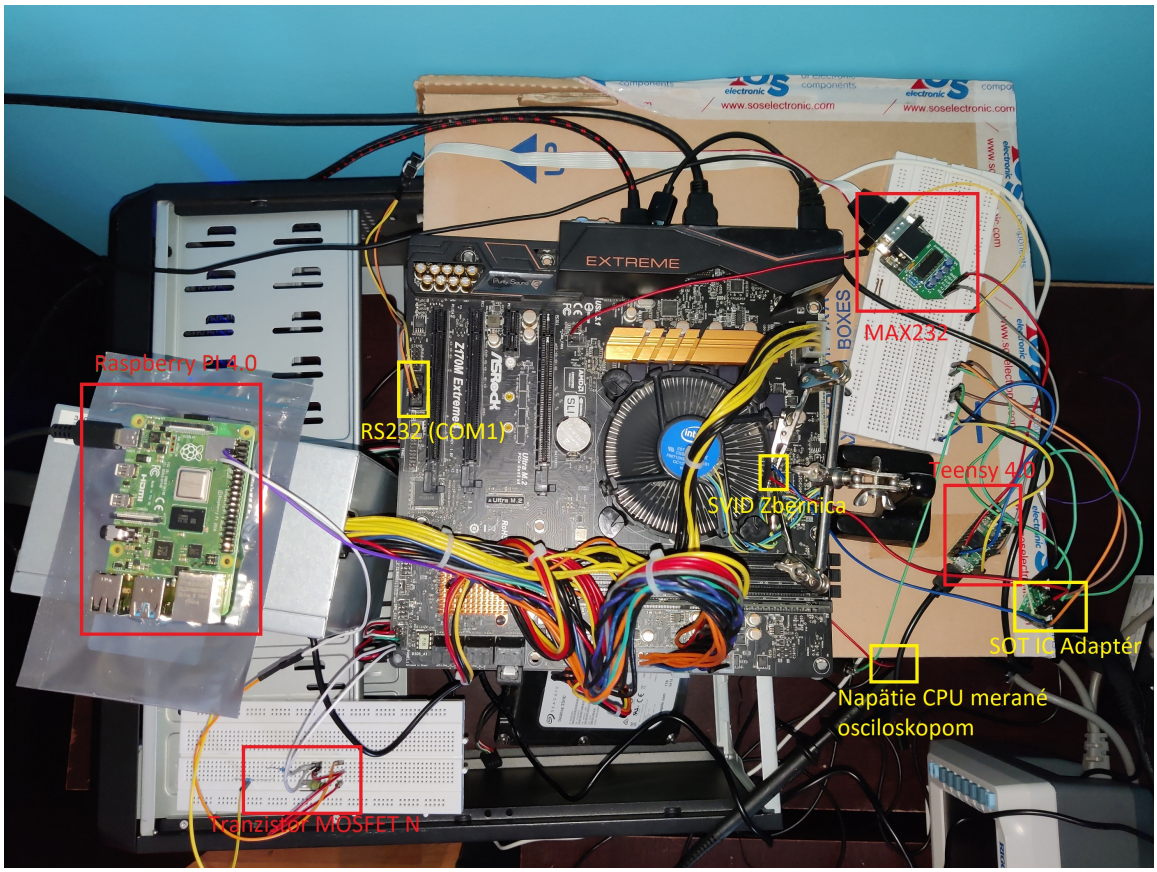
Substitučná tabuľka S-box

	00	**01**	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**	**0a**	**0b**	**0c**	**0d**	**0e**	**0f**
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Tabuľka B.1: S-box tabuľka[34]

Príloha C

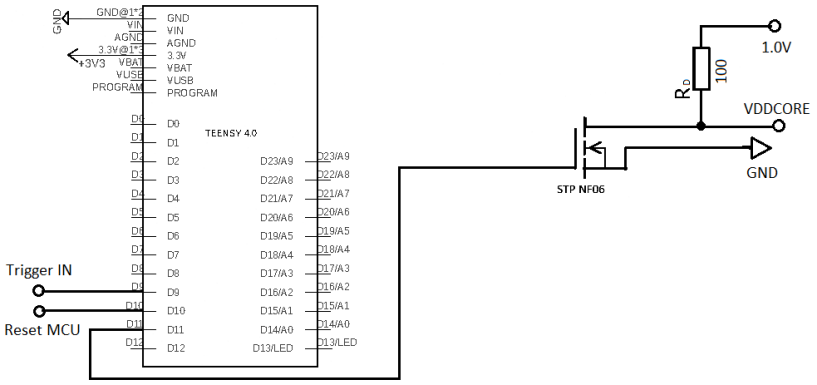
Doska VoltPillager vrátane rozšírení



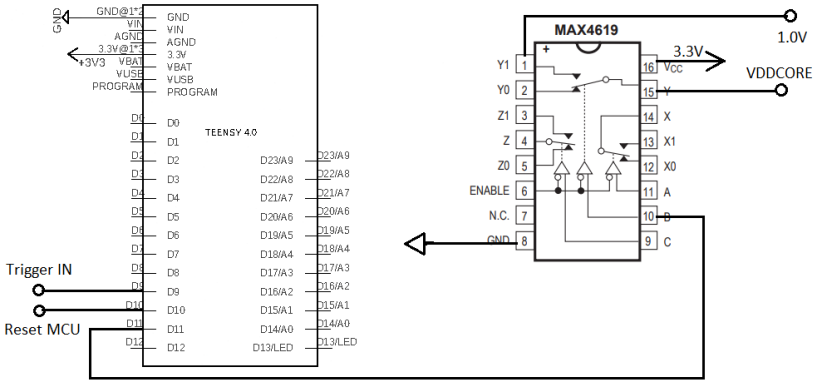
Obr. C.1: Pripojenie dosky VoltPillager a Raspberry PI k PC

Príloha D

Možnosti zapojenia schémy na vykonanie útoku TrustZone-M(eh)



Obr. D.1: Možnosť 1 - Schéma zapojenia na vykonanie útoku TrustZone-M(eh) s využitím tranzistora



Obr. D.2: Možnosť 2 - Schéma zapojenia na vykonanie útoku TrustZone-M(eh) s využitím multiplexora