FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	Analýza postranních kanálů postkvantového podpisu Rainbow
Student:	Bc. David Pokorný
Vedoucí:	Ing. Petr Socha
Studijní program:	Informatika
Obor / specializace:	Počítačová bezpečnost
Katedra:	Katedra informační bezpečnosti
Platnost zadání:	do konce letního semestru 2021/2022

Pokyny pro vypracování

Cílem diplomové práce je analýza postkvantového podpisového schéma Rainbow, založeného na problému řešení soustavy kvadratických rovnic, z hlediska zranitelnosti vůči útokům postranními kanály; a dále návrh protiopatření proti takovým útokům.

- Prostudujte algoritmus pro digitální podpis Rainbow a jeho varianty navržené pro standardizaci institutem NIST
- Zvolte variantu vhodnou pro vestavná zařízení a prostudujte její referenční implementaci
- Zprovozněte tuto implementaci na 32-bitovém mikrokontroléru ARM a navrhněte útok postranním kanálem, který bude umožňovat získání celého (nebo větší části) soukromého klíče
- Realizujte navržený útok a řádně experimentálně vyhodnoťte jeho složitost s ohledem na danou platformu a variantu algoritmu
- Navrhněte protiopatření proti takovým útokům založené na principech maskování
- Implementujte navržené protiopatření a řádně experimentálně (případně i formálně) vyhodnoťte jeho účinnost a efektivnost

---

Elektronicky schválil/a prof. Ing. Róbert Lórencz, CSc. dne 29. prosince 2020 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Analýza postranních kanálů postkvantového podpisu Rainbow

Bc. David Pokorný

Katedra informační bezpečnosti
Vedoucí práce: Ing. Petr Socha

6. května 2021

Poděkování

Rád bych poděkoval vedoucímu této diplomové práce Ing. Petru Sochovi za návrh tématu, pomoci při realizaci náročného měření, za poskytnutí věcných poznámek a konstruktivní kritiky. Bez jeho pomoci bychom nepublikovali částečné výsledky této práce na konferenci DATE 2021.

Rovněž bych rád poděkoval Dr.-Ing. Martinu Novotnému a laboratoři vestavné bezpečnosti za zapůjčení přípravku ChipWhisperer. S tímto přípravkem jsem byl schopen realizovat měření v domácích podmínkách během pandemie Covid-19.

Moje vděčnost také patří celé katedře číslicového návrhu, jejichž zaměstnanci silně projevují zájem o své studenty, aktivně nabízejí příležitosti pro vědecký výzkum a motivují své studenty formou finanční podpory.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principu při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisu, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

v Praze dne 6. května 2021 . . . . . .

Odkaz na tuto práci

Pokorný, David Pokorný. *Analýza postranních kanálů postkvantového podpisu Rainbow*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Abstrakt

Rainbow, postkvantové podpisové schéma postavené na řešení soustavy kvadratických rovnic, je kandidát na nový standard připravovaný Národním institutem standardů a technologie (NIST). Základem práce je navrhnout cílený útok pomocí postranních kanálů na referenční implementaci, která byla dodána pro testovací účely během procesu standardizace. Útok je testován na 32bitovém STM32F3 ARM mikrokontroléru. Spolu s útokem jsou navrhnuta vhodná protiopatření, která zvyšují implementační bezpečnost. Implementovaná protiopatření jsou vyhodnocena z hlediska úniku informace.

Klíčová slova postkvantová kryptografie, Rainbow, digitální podpis, multivarietní kvadratické rovnice, útok postranními kanály, zabezpečení implementace

Abstract

Rainbow, a layered multivariate quadratic digital signature, is a candidate for standardization by National institute of standards and technology (NIST). In this paper, we present a CPA side-channel attack on the submitted 32-bit

reference implementation. We evaluate the attack on an STM32F3 ARM microcontroller. After a successful attack, we propose countermeasures against side-channel attacks. Countermeasures are implemented and evaluated using leakage assessment.

Keywords post-quantum cryptography, Rainbow, digital signature, multivariate quadratic, side-channel analysis, leakage assessment

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Neoddělitelnou součástí dnešní bezpečnosti je asymetrická kryptografie, která nám umožňuje použitím privátního a veřejného klíče zajistit důvěrnost, autenticitu či nepopiratelnost zprávy. Široce používané standardy staví svoji bezpečnost na problému faktorizace a problému diskrétního logaritmu. Od roku 1995, kdy Petr W. Shor zveřejnil práci *Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer* [2], která řeší faktorizaci v polynomiálním čase na kvantovém počítači, jsou považovány mnohé asymetrické šifry/podpisy (např. RSA) z dlouhodobého hlediska jako nebezpečné. Pro větší instituce je nepřijatelné, aby jejich přenášená data byla za dekádu dešifrována a kompromitována.

Z toho důvodu *Národní institut standardů a technologie* (NIST) organizuje výběrové řízení na nové postkvantové, či kvantově rezistentní, standardy pro asymetrické šifrování a digitální podpis. Toto řízení je aktuálně v posledním výběrovém kole, kde jedním z kandidátů na podpisové schéma je právě Rainbow [3].

Rainbow je podpisové schéma založené na problému řešení soustavy multivarietních kvadratických rovnic (MQ problem). Tuto soustavu nazýváme centrální zobrazení. Centrální zobrazení je strukturováno tak, aby se dalo řešit stejně jako soustava lineárních rovnic. Neznámé jsou kategorizovány do *oil variables* a *vinegar variables*. Speciální struktura centrálního zobrazení je pro potřeby veřejného klíče skryta pomocí přidaných afinních zobrazení.

Prvotní schéma využívající oil/vinegar proměnné je schéma *Oil and Vinegar*, které obsahuje stejný počet oil a vinegar proměnných. Toto schéma bylo záhy prolomeno [4]. Dalším vývojovým krokem bylo schéma *Unbalanced Oil and Vinegar* (UOV), které mělo více vinegar variables než oil variables [5]. UOV má mnohá rozšíření (např. vyšší řády a HFEV) a jedním rozšířením je právě Rainbow, který dělí centrální zobrazení do vrstev.

Rainbow jako kryptosystém je možná teoreticky bezpečný, ale implementace rozhodně být bezpečná nemusí. Implementace kryptografických algoritmů

jsou známy pro svoji zranitelnost vůči útoku postranními kanály [6, 7]. Jedná se o útoky, které využívají informací získaných z jiných zdrojů způsobených samotným prováděním algoritmu. Tyto zdroje mohou být různého původu, např. časový kanál, akustický kanál, elekromagnetický kanál, nebo odběrový kanál, kde využíváme informace o spotřebě samotného zařízení.

V této práci navrhujeme útok postranními kanály s využitím korelační analýzy. Útok cílíme na 32bitovou referenční implementaci Rainbow, která byla dodána spolu s návrhem Rainbow na standardizaci. V rámci této práce rovněž naměříme záznamy o spotřebě a navržený útok provedeme na 32bitovém STM32F3 ARM mikrokontroléru. Výsledkem by mělo být plné získání soukromého klíče ze zařízení, pouze pomocí postranního odběrového kanálu. Následně útok analyzujeme pomocí metriky míry úspěchu [8].

Po úspěšném útoku navrhneme a implementujeme protiopatření, která zabrání předchozímu útoku. První protiopatření se bude zakládat na velice známém aditivním maskování [9], kdežto druhá verze protiopatření bude vycházet ze znalosti algoritmu a jeho soukromého klíče. Bude se jednat o generování ekvivalentního klíče, který může být použit místo původního soukromého klíče. Obě protiopatření budou následně řádně otestována pomocí měření úniku informace [10]. Rovněž ukazujeme, že je nezbytné vzít v úvahu všechny vektory útoku a adekvátně chránit implementaci. Pokud by tomu tak nebylo, dostali bychom u postkvantové kryptografie falešný pocit bezpečí místo skutečné výhody.

Kapitola 1

Úvod do problematiky

1.1 Podpisové schéma Rainbow

Nechť $\mathbb{F}$ je konečné těleso a $v_1, v_2, \ldots, v_u, v_{u+1} \in \mathbb{N}$ je sada $(u+1)$ parametrů, pro které platí $0 < v_1 < v_2 < \cdots < v_u < v_{u+1} = n$. Vrstva Rainbow představuje samostatnou množinu kvadratických rovnic, kde pro každou vrstvu definujeme dvě množiny indexů (vrstvy jsou indexovány přes l):

$$\forall l \in \{1, 2, \ldots, u\} : V_l = \{1, 2, \ldots, v_l\}, |V_l| = v_l, \tag{1.1}$$

$$O_l = \{v_l + 1, \ldots, v_{l+1}\}, |O_l| = o_l. \tag{1.2}$$

Zde dostáváme Rainbow s u vrstvami. Vrstva l obsahuje v_l vinegar proměnných s indexy V_l, to jsou proměnné $\{x_k | k \in V_l\}$, a o_l oil proměnných s indexy O_l, to jsou proměnné $\{x_k | k \in O_l\}$. l-tá vrstva obsahuje o_l rovnic, které jsou indexovány stejně jako oil proměnné, tedy indexy O_l. Celkový počet rovnic Rainbow značíme $m \coloneqq n - v_1$.

Centrální zobrazení $F = (f^{(v_1+1)}, f^{(v_1+2)}, \ldots, f^{(n)})$ je množina multivarietních kvadratických polynomů, kde

$$f^{(k)}(x_1, \ldots, x_n) \coloneqq \sum_{\substack{i,j \in V_l \\ i \leq j}} \alpha_{i,j}^{(k)} x_i x_j + \sum_{\substack{i \in V_l \\ j \in O_l}} \beta_{i,j}^{(k)} x_i x_j, \tag{1.3}$$

kde $\alpha_{i,j}^{(k)}, \beta_{i,j}^{(k)} \in \mathbb{F}$ jsou kvadratické koeficienty, $k \in \{v_1 + 1, \ldots, n\}$ a $l \in \{1, \ldots, u\}$ je takové, aby $k \in O_l$. V této definici vynecháváme pro jednoduchost lineární a absolutní koeficienty, které nejsou využity v referenční implementaci, na kterou budeme útočit. Pokud by se tyto koeficienty zahrnuly, náš útok by se nijak nelišil. Kvadratické polynomy, které nemají lineární a absolutní členy, nazýváme kvadratické formy.

Struktura v rovnici 1.3 je zvolena speciálně tak, abychom nemuseli řešit MQ-problem (viz kapitola 1.1.2). Tyto kvadratické polynomy musíme ovšem

zamaskovat, pokud je chceme zveřejnit ve veřejném klíči. Toho docílíme mixováním oil a vinegar proměnných pomocí dvou affiních zobrazení. Opět pro jednoduchost vynecháme posuv (translaci), čímž budeme předpokládat pouze lineární zobrazení.

Nechť $S : \mathbb{F}^m \to \mathbb{F}^m$ a $T : \mathbb{F}^n \to \mathbb{F}^n$ jsou lineární zobrazení definované regulárními maticemi $\mathbf{S} \in \mathbb{F}^{m\times m}$ a $\mathbf{T} \in \mathbb{F}^{n\times n}$. Následně můžeme definovat soustavu kvadratických rovnic $P = S \circ F \circ T : \mathbb{F}^n \to \mathbb{F}^m$. Vyřešit P^{-1} je MQ-problem, který je NP-těžký [11].

Privátní klíč (secret key) je definován jako $SK \coloneqq (\mathbf{S}^{-1}, F, \mathbf{T}^{-1})$ a veřejný klíč (public key) jako $PK \coloneqq (P)$.

1.1.1 Maticové vyjádření centrálního zobrazení

Centrální zobrazení je reprezentováno soustavou kvadratických forem. Nejdřív si zadefinujeme značení vektorů proměnných:

$$\mathbf{x}_l^v \coloneqq \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{v_l} \end{pmatrix}, \mathbf{x}_l^o \coloneqq \begin{pmatrix} x_{v_l+1} \\ x_{v_l+2} \\ \vdots \\ x_{v_{l+1}} \end{pmatrix}, \tag{1.4}$$

kde $\mathbf{x}_l^v$ je sloupcový vektor vinegar proměnných l-té vrstvy (tzn. vektor délky v_l) a $\mathbf{x}_l^o$ je sloupcový vektor oil proměnných l-té vrstvy (tzn. vektor délky o_l). Písmena v a o hrají pouze rozlišovací roli a nejsou to proměnné. Polynom centrálního zobrazení lze napsat jako:

$$f^{(k)} = (\mathbf{x}_l^v)^T \mathbf{A}^{(k)} \mathbf{x}_l^v + (\mathbf{x}_l^v)^T \mathbf{B}^{(k)} \mathbf{x}_l^o, \tag{1.5}$$

kde $\mathbf{A}^{(k)} \in \mathbb{F}^{v_l \times v_l}, \mathbf{B}^{(k)} \in \mathbb{F}^{v_l \times o_l}$, $k \in \{v_1 + 1, \ldots, n\}$ a $l \in \{1, \ldots, u\}$ je takové, že $k \in O_l$. Matice následně vypadají takto:

$$\mathbf{A}^{(k)} = \begin{pmatrix} \alpha_{1,1}^{(k)} & \alpha_{1,2}^{(k)} & \ldots & \alpha_{1,v_l}^{(k)} \\ 0 & \alpha_{2,2}^{(k)} & \ldots & \alpha_{2,v_l}^{(k)} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \alpha_{v_l,v_l}^{(k)} \end{pmatrix}, \mathbf{B}^{(k)} = \begin{pmatrix} \beta_{1,1}^{(k)} & \ldots & \beta_{1,o_l}^{(k)} \\ \vdots & \ddots & \vdots \\ \beta_{v_l,1}^{(k)} & \ldots & \beta_{v_l,o_l}^{(k)} \end{pmatrix}. \tag{1.6}$$

Celé centrální zobrazení může být, díky maticovému vyjádření, reprezentováno jako množina matic.

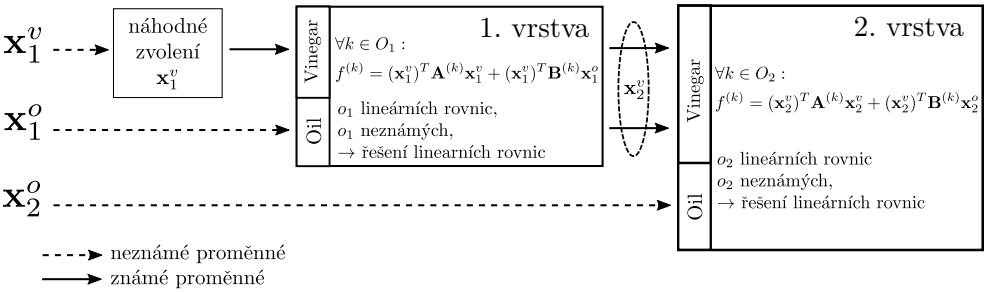
Veřejný klíč tvoří obecné kvadratické formy. Během generování se využije komutativity multiplikativní operace tělesa, čímž se sečtou stejné koeficienty u stejných kvadratických proměnných ($x_i x_j = x_j x_i$). Tím dostáváme horní trojúhelníkovou matici (stejně tomu je i u matice $\mathbf{A}^{(k)}$). Kvadratické formy veřejného klíče tedy lze reprezentovat jako m horních trojúhelníkových matic o rozměrech $n \times n$.

1.1.2 Výpočet inverze centrálního zobrazení F^{-1}

Členy polynomu definovaného v rovnici 1.3 jsme rozdělili na členy α a β. Řešení inverze centrálního zobrazení spočívá ve znalosti vinegar proměnných. Předpokládejme, že známe vinegar proměnné $\{x_i | i \in V_l\}$, pak suma s α koeficienty je plně známá a může být vypočítána. Členy s koeficienty β obsahují vždy jednu vinegar proměnnou a jednu oil proměnnou. Při znalosti vinegar proměnných lze za ně dosadit a polynom v rovnici 1.3 se následně redukuje na polynom prvního stupně.

V l-té vrstvě dostáváme o_l polynomů prvního stupně. Během podepisování položíme tyto polynomy do rovnosti s vektorem, pro který řešíme centrální zobrazení. Tím dostáváme soustavu o_l lineárních rovnic o o_l neznámých. Ty již umíme efektivně řešit.

Nyní ke znalosti vinegar proměnných. Pro první vrstvu máme $o_1 + v_1$ proměnných pro o_1 rovnic. Vygenerováním v_1 proměnných nám zůstane právě o_1 neznámých, které můžeme dopočítat. Vygenerovat v_1 proměnných si můžeme dovolit, protože centrální zobrazení má n proměnných a m rovnic. Celkový počet volných proměnných je tedy $n - m = v_1$. Pokud by řešení pro dané vygenerované vinegar proměnné neexistovalo, vygenerujeme nové a počítáme znova[1]. Pro zbylé vrstvy $l \in \{2, \ldots, u\}$ platí, že vinegar proměnné jsou výsledek počítání předchozí vrstvy, protože $V_l = V_{l-1} \cup O_{l-1}$. Jinými slovy řečeno, vstupem do další vrstvy je výstup z předchozí vrstvy a l-tá vrstva dopočítá o_l proměnných. Tuto skutečnost pro dvě vrstvy zachycuje obrázek 1.1.



Obrázek 1.1: Postup počítání inverze centrálního zobrazení

1.1.3 Generování a ověření podpisu

Mějme definované parametry Rainbow, zadaný privátní klíč $SK = (\mathbf{S}^{-1}, F, \mathbf{T}^{-1})$, dokument $\mathbf{d}$ k podepsání a náhodný vektor $\mathbf{s}$ (salt). Pak podpisem nazveme dvojici $(\mathbf{z}, \mathbf{s})$, kde

[1] V implementaci se ve skutečnosti negenerují nové vinegar proměnné, ale znova se randomizuje vstup pomocí náhodného vektoru v haši – viz sekce 1.1.3.

$$\begin{aligned} \mathbf{h} &\coloneqq \text{hash}\left(\text{hash}\left(\mathbf{d}\right) || \ \mathbf{s}\right), \\ \mathbf{y} &\coloneqq \mathbf{S}^{-1}\mathbf{h}, \\ \mathbf{x} &\coloneqq F^{-1}(\mathbf{y}), \\ \mathbf{z} &\coloneqq \mathbf{T}^{-1}\mathbf{x}, \end{aligned} \tag{1.7}$$

kde operace || je zřetězení a operace hash je libovolná bezpečná hašovací funkce. $\mathbf{S}^{-1}$, resp. $\mathbf{T}^{-1}$ je předpočítaná inverzní matice k $\mathbf{S}$, resp. $\mathbf{T}$, a F^{-1} se počítá dle kapitoly 1.1.2.

Nyní máme podpis $(\mathbf{z}, \mathbf{s})$ pro dokument $\mathbf{d}$. K ověření pravosti podpisu potřebuje veřejný klíč $PK = (P)$, kde $P = S \circ F \circ T$. Podpis je validní právě tehdy, když $\mathbf{h} = \mathbf{h}'$, kde

$$\begin{aligned} \mathbf{h} &\coloneqq \text{hash}\left(\text{hash}\left(\mathbf{d}\right) || \ \mathbf{s}\right), \\ \mathbf{h}' &\coloneqq P(\mathbf{z}). \end{aligned} \tag{1.8}$$

Výpočet $P(\mathbf{z})$ představuje dosazení vektoru $\mathbf{z}$ do množiny polynomů, což vede na snadný výpočet.

Bezpečnost podpisu podle EUF-CMA

Podpis musí prokazatelně splňovat bezpečnost podle definice EUF-CMA (Existential UnForgeability under Chosen Message Attack), tedy proti útoku, kdy útočník může volit zprávu k podpesání. Bezpečnost proti tomuto typu útoku často zaručuje Full Domain Hash scheme (FDH). FDH zde nemůže být aplikován, protože jednosměrná funkce f (trapdoor function) musí splňovat dvě podmínky. Za prvé, $f(x)$ musí být z rovnoměrného rozdělení při výběru x z libovolného rozdělení D. Za druhé, inverzní algoritmus nenalezne jen vzor k obrazu, ale daná množina vzorů bude opět spadat do dané distribuce D. V [12] docílili EUF-CMA zajištěním rovnoměrného rozdělení vstupu pro podpisová schémata UOV a HFE, se kterými je Rainbow příbuzný. Základní modifikace spočívá ve vygenerování náhodných vinegar proměnných, pro které bude centrální zobrazení F invertibilní. Nadále se vinegar nemění. Při počítání inverze se může stát, že pro daný vstup podpisu nelze nalézt výstup. V tomto případě se vstup znova randomizuje (vektor $\mathbf{s}$ v rovnici 1.7).

1.2 Korelační odběrová analýza

Korelační odběrová analýza (correlation power analysis – CPA) je metoda analýzy postranními kanály, kde se využívá záznamů o spotřebě. Tyto záznamy se korelují s ohadovanou spotřebou, která je závislá na citlivé informaci v zařízení. Tato metoda byla popsána v roce 2004 s použitím modelu úniku

informace (leakage model) [7]. Základním stavebním kamenem je model, který na základě hypotézy o klíči určuje míru spotřeby energie. Dnešní integrované logické obvody jsou postavené na technologii CMOS, u které lze predikovat relativní spotřebu. Spotřeba CMOS má dvě složky – statická a dynamická. Statická spotřeba tvoří minoritní část, a proto ji při analýze spotřeby neuvažujeme. Dynamická spotřeba se projevuje při změně stavu, v našem případě při změně hodnoty v registru nebo na sběrnici. Model spotřeby tedy predikuje (v našem případě) počet měnících se bitů v daném čase. V naší práci používáme odhad pomocí Hammingovy váhy.

1.2.1 Míra úspěšnosti (success rate)

Pro vyjádření úspěšnosti útoku postranním kanálem je potřeba určit metodologii, podle které výjadříme míru úspěchu. Tento krok je důležitý, abychom měli možnost naše výsledky porovnávat s jinými útoky postranním kanálem. Pro tyto účely použijeme metriku míry úspěchu (success rate) [8].

Mějme rodinu kryptosystému $E_K = \{E_k\}_{k \in \mathcal{K}}$, využívající klíče k z množiny všechny klíčů $\mathcal{K}$. Dvojici (E_K, L) nazveme fyzickým zařízením, které provádí kryptografickou operaci E_K nad klíči z $\mathcal{K}$, a ze kterého uniká informace postranním kanálem, který reprezentuje funkce L. Útoky postranními kanály se zakládají na metodě rozděl a panuj, proto si zavedeme množinu podklíčů S a zobrazení $\gamma : \mathcal{K} \to S$, které zobrazuje klíče do podmnožiny klíčů. V našem případě $k \in \mathcal{K}$ je soukromý klíč a S je množina prvků, na které útočíme.

Pak obnovení klíče postranním kanálem označíme $A_{E_K,L}$ s časovou složitostí τ, paměťovou složitostí m a q požadavky na zařízení. Výsledkem $A_{E_K,L}$ je vektor $\mathbf{g} = (g_1, g_2, \ldots, g_{|S|})$, který popisuje pravděpodobnost správného klíče od nejpravděpodobnějšího (g_1) po nejméně pravděpodobný $(g_{|S|})$.

Algoritmus 1 popisuje obnovení klíče postranním kanálem L kryptosystému E_K řádu o. Zde se klíč nastavuje náhodně, proto nebudeme tento proces nazývat útokem, ale experimentem Exp. Řád o zde popisuje, na kolikáté maximální pozici může být správný klíč, aby byl experiment úspěšný.

Algoritmus 1 Experiment $\mathbf{Exp}^{o}_{A_{E_K,L}}$

výstup: 1, pokud byl experiment úspěšný, jinak 0

1: $k \xleftarrow{R} \mathcal{K}$ (náhodně zvolen klíč)
2: $s = \gamma(k)$
3: $\mathbf{g} \leftarrow A_{E_k,L}$
4: **pokud** $s \in (g_1, \ldots, g_o)$ **pak**
5: **vrať** 1
6: **jinak**
7: **vrať** 0
8: **konec podmínky pokud**

Míra úspěchu řádu o následně představuje pravděpodobnost úspěšného experimentu popisující obnovení klíče postranním kanálem:

$$\mathbf{Succ}^{o}_{A_{E_K,L}}(\tau, m, q) = \mathrm{P}(\ \mathbf{Exp}^{o}_{A_{E_K,L}} = 1\ ) \tag{1.9}$$

1.3 Aditivní maskování

Aditivní maskování je způsob maskování citlivých informací, kde základním principem je vhodně rozdělit citlivé hodnoty (hodnoty závislé na tajném klíči) mezi $(d+1)$ sdílených hodnot x_i, kde d se značí řád masky. Pokud vezmeme libovolných d sdílených hodnot, pak ty nenesou žádné informace o původní citlivé hodnotě. Cílení útoku na takovou množinu hodnot nepřinese tedy žádný účinek [9].

Nyní je potřeba říct, co znamená rozdělit „vhodným“ způsobem. Označme citlivou informaci jako x, pak říkáme, že x rozdělíme na $d+1$ sdílených hodnot, pokud:

$$x = \bigoplus_{i=1}^{d+1} x_i, \tag{1.10}$$

kde $\forall i \in \{1, 2, \ldots, d\} : x_i$ jsou náhodné hodnoty z rovnoměrného rozdělení a x_{d+1} dopočítáme tak, aby součet sdílených hodnot dal původní informaci.

Pokud uvažujeme ochranu lineární transformace $y = L(x)$, pak nejjednodušším způsobem je zpracovat jednotlivé sdílené hodnoty nezávisle:

$$\forall i \in \{1, 2, \ldots, d+1\} : y_i = L(x_i). \tag{1.11}$$

Pak ze samotné definice lineární transformace dostáváme:

$$y = \bigoplus_{i=1}^{d+1} y_i = \bigoplus_{i=1}^{d+1} L(x_i) = L(\bigoplus_{i=1}^{d+1} x_i) = L(x). \tag{1.12}$$

Pokud jsou lineární transformace zpracovávány nezávisle a v jiném čase, pak bychom při útoku museli cílit na všechny sdílené hodnoty. To vede na útok řádu $d+1$. Řád maskování d, používající $(d+1)$ sdílených hodnot, je odolný vůči útoku řádu d.

1.4 Multiplikativní maskování

Multiplikativní maskování je způsob maskování citlivé hodnoty pomocí nenulové náhodné masky a multiplikativní operace. Pokud citlivou hodnotu označíme x a masku jako $m \in \mathbb{F} \setminus \{0\}$, pak maskovanou hodnotu u vypočítáme jako:

$$u = x \otimes m, \tag{1.13}$$

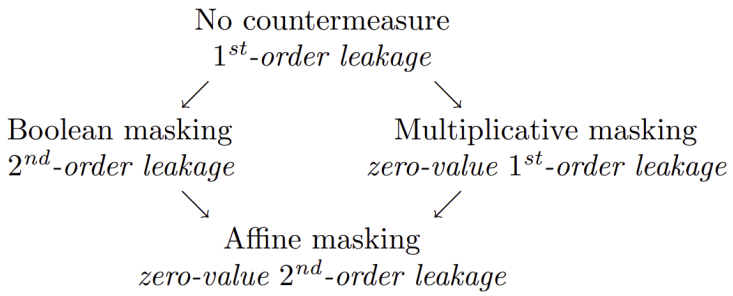
kde operace $\otimes$ je multiplikativní operace tělesa $\mathbb{F}$. Tuto hodnotu následně používáme v lineárních zobrazení podobně jako v kapitole 1.3.

Zásadní rozdíl z pohledu bezpečnosti je kritický případ, kdy $x = 0$. Následně samotné multiplikativní maskování nijak nezmění danou hodnotu, protože:

$$u = x \otimes m = 0 \otimes m = 0 \implies u = x. \tag{1.14}$$

Tímto dostáváme zranitelnost vůči nulovým prvkům při útoku prvního řádu.

V článku [1] se autoři věnují afinnímu maskování, které je kombinací aditivního a multiplikativního maskování. Úrovně bezpečnosti při použití různých maskování shrnují na obrázku 1.2.



Obrázek 1.2: Shrnutí úrovně zabezpečení podle typu maskování [1]

Na horní úrovni je nezabezpečená implementace, na kterou lze zaútočit útokem prvního řádu. První zabezpečení může být Boolean masking, které my popisujeme jako aditivní maskování prvního řádu. To lze napadnout útokem druhého řádu. Druhé možné zabezpečení je multiplikativní maskou, kterou lze napadnout ještě prvním řádem, ale můžeme cílit pouze na nulové prvky. Konečným stupínkem maskování (prvního řádu) máme afinní maskování, které lze napadnou až útokem druhého řádu, a to pouze nulové prvky.

1.5 Testování úniku informací (leakage assessment)

Předpokládejme, že již máme navržená a implementovaná protiopatření, pak bychom měli ověřit, zda jsou účinná. Aplikace původního útoku, nebo vytvoření nového útoku by v případě neúspěchu nedokazovalo, že jsou protiopatření dobrá. Každý dokáže systém zabezpečit před sebou samým. Na místo toho musíme volit obecnější přístup, kde nebudeme hledat konkrétní slabinu, ale únik libovolné informace ze zařízení.

Proces ověřování, že ze zařízení neuniká informace, se nazývá *leakage assesment* (LA). Popišme si základní myšlenku často používané kategorie LA, a to *partitioning-based LA*. Nejdříve se naměří mnoho záznamů o spotřebě (řádově více, než se měří pro samotný útok). Následně se záznamy rozdělí do množin podle zvoleného kritéria, které je závislé na zvolené LA metodě. Posledním krokem je pokus o nalezení rozdílu mezi jednotlivými množinami záznamů. Pokud jsme byli schopni nalézt rozdíl, tedy rozeznat záznamy od sebe, pak

ze zařízení zcela jistě uniká informace. Pokud jsme rozdíl nenalezli, pak buď volíme špatnou metodologii, nebo ze zařízení neuniká informace. Nikdy nemůžeme prohlásit, že je zařízení zcela jistě zabezpečeno proti útoku postranním kanálem.

1.5.1 Welchův t-test

Welchův t-test je základní testovací statistika, která odhaluje, zda dva náhodné výběry mají stejnou střední hodnotu. Zde nepředpokládáme stejný rozptyl náhodných proměnných a jedná se o nepárový t-test. Nulovou hypotézou nazveme fakt, že výběrové hodnoty mají stejnou střední hodnotu. Výsledkem Welchova t-testu je zamítnutí/nezamítnutí nulové hypotézy na určité hladině významnosti.

Mějme dvě množiny náhodných výběrů $\mathcal{Q}_0$ a $\mathcal{Q}_1$, které reprezentují naměřené hodnoty spotřeby. Nechť μ_0, μ_1 jsou odpovídající výběrové střední hodnoty, s_0^2, s_1^2 jsou jejich výběrové rozptyly a n_0, n_1 značí kardinality. Testovací statistika t a stupeň volnosti v je:

$$t = \frac{\mu_0 - \mu_1}{\sqrt{\frac{s_0^2}{n_0} + \frac{s_1^2}{n_1}}}, \qquad v = \frac{\left(\frac{s_0^2}{n_0} + \frac{s_1^2}{n_1}\right)^2}{\frac{\left(\frac{s_0^2}{n_0}\right)^2}{n_0 - 1} + \frac{\left(\frac{s_1^2}{n_1}\right)^2}{n_1 - 1}}. \tag{1.15}$$

Následně pro vysoké t hodnoty dostáváme důkaz k zamítnutí nulové hypotézy. Důsledkem je, že původní dvě výběrové množiny jsou měřeny z jiného rozdělení. Pro jednoduchost se jako hranice zamítnutí volí hodnota $|t| > 4{,}5$, bez toho, aby se stupeň volnosti bral v potaz. Tato hodnota t míří na konfidenční hladinu $> 0{,}99999$ [10].

Při použití testovací statistiky pro účely LA je potřeba zdůraznit, jakou informaci nám testovací statistika vlastně poskytne. Krom limitací vycházejících z omezení samotné testovací statistiky (falešně pozitivní/negativní výsledky) zde máme chybné použití za účelem důkazu bezpečnosti.

LA pomocí Welchova t-testu je varianta vyhodnocení z kategorie *Test vector leakage assessment* (TVLA). TVLA jsou metody, které využívají selektování vstupu z určitého rozdělení. Pokud použijeme TVLA za účelem důkazu bezpečnosti, pak je potřeba samotný šum brát jako bezpečnostní parametr. Tento fakt lze akceptovat pro výzkumné účely (kde míříme na útok určitého řádu a můžeme šum přijmout jako bezpečnostní parametr), ale potenciální riziko nastává u reálných produktů. Při útoku je šum fixní a jediná relevantní metrika je počet naměřených záznamů o spotřebě pro úspěšný útok [13].

Další důležitá součást TVLA je řád maskování a řád útoku. Pokud se bavíme o implementaci ochráněné maskováním řádu d, pak toto maskování je odolné vůči útoku řádu d. Následně má smysl provádět TVLA řádu $\leq d$, kde se snažíme dokázat bezpečnost dané implementace vůči tomuto řádu (opět

při daném šumu). Útok na vyšší řád může být úspěšný. Je tedy vždy nutné uvést vůči jakému řádu TVLA implementaci chráníme.

Kapitola 2

Analýza

Nyní máme definované naše prerekvizity a můžeme se vyjádřit ke směru, jakým se bude práce ubírat.

2.1 Analýza útoků postranními kanály

První část této práce je věnována právě útoku postranním kanálem (SCA) na podpisové schéma Rainbow. Do kategorie SCA útoků spadá celá škála technik.

Nejjednodušší verzí SCA je *simple power analysis* (SPA). Pro tyto útoky se využívá právě jeden (či jednotky) záznamů o spotřebě a samotná informace je obsažena napříč časem. Pokud si vykreslíme záznamy do grafu, pak čas je na horizontální ose a samotná množina záznamů na ose vertikální. Odtud se SPA říká rovněž horizontální útok. Tyto útoky se často zakládají na časovém postranním kanále, nebo na rozpoznání větvení samotného programu. Typickým příkladem SPA je útok na šifru RSA, která je impementována nezabezpečenou formou algoritmu *square and multiply*.

Druhou kategorií útoků jsou útoky vertikální, tedy s využitím mnoha záznamů o spotřebě. Tyto útoky se dělí na profilované (template) útoky a neprofilované útoky. Rozdíl spočívá v tom, že u profilovaného útoku předchází profilovací část, při které se nastavují parametry útoku pro přesnější zacílení informace, která uniká ze zařízení. U neprofilovaných útoku tento krok odpadá.

Nyní je otázkou, jakým způsobem zaútočíme na referenční implementaci my. Pokud bychom volili SPA (bez použití hlubokého učení), pak bychom museli objevit zranitelnost v samotném algoritmu, resp. v provádění programu. Pokud neznáme možný cíl útoku, pak lze využít měření úniku informace pro odhalení zranitelnosti. Toto je ovšem složitý problém, protože leakage assessment u SPA není natolik používaný jako u vertikálních útoků.

Předpokládejme, že chceme použít vertikální útok. Momentálně jediný nám známý útok na Rainbow je [14]. V tomto článku provedli korelační odběrovou analýzu na vlastní implementaci Rainbow. Oproti referenční implementaci, na kterou cílíme my, jsou zde podstatné rozdíly: ve struktuře klíče, v algoritmu samotného násobení nebo v architektuře na kterou cílí (používají 8-bitovou architekturu). Jelikož tyto rozdíly jsou podstatné, nejedná se pouze o reprodukování jejich výsledků, ale i vylepšení útoku. Útok zároveň provádíme na implementaci, která by se reálně mohla používat, čímž tato práce získává význam na poli výzkumu.

Z těchto důvodu jsme se rozhodli provést korelační odběrovou analýzu, která vychází z článku [14]. Jedná se tedy o vertikální neprofilovaný útok, využívající korelační analýzy. Útok cílí na lineární zobrazení, které skrývají centrální zobrazení. Po odhalení zobrazení S a T, lze extrahovat centrální zobrazení např. z veřejného klíče, tím získáme celý soukromý klíč.

2.2 Analýza referenční implementace

Součástí podání návrhu kryptosystému do výběrového řízení NIST je přiložení referenční implementace, která je následně používána k testování různých aspektů šifry (efektivita, velikosti klíčů, paměťová a časová náročnost podpisu apod.) Tato implementace je vhodná k provedení zamýšleného útoku z několika hledisek. Předně je veřejná a slouží právě pro účely testování, kde zranitelnosti mohou být před vydáním standardu upraveny. Rovněž, jelikož nevytváříme vlastní implementaci, nejsme v pokušení zavést do algoritmu umělou slabinu, která by nám napomohla k úspěšnému útoku. Po celou dobu počítáme s tím, že útok lze replikovat pro danou verzi Rainbow, kterou jsme použili.

Referenční implementaci zhotovili na Univerzitě v Cincinnati, Ohio, USA. Hlavním zadavatel je Jintai Ding. Implementace obsahuje dvě varianty klíče. Classic variant ukládá klíč v plné podobě jako pole elementů daného tělesa. CZ-Rainbow[2] je varianta, která dovoluje vygenerovat částečný veřejný klíč, dále vygenerovat mapy **S** a **T** a následně dopočítat zbytek. Tímto přístupem může být část veřejného klíče uskladněná jako seed pro PRNG, čímž se snižuje velikost veřejného klíče až o 70%. Obě tyto verze mají možnost Compressed key, kde se navíc používá další seed pro generování matic **S** a **T**. Jelikož tyto varianty stejně vyžadují dopočítat soukromný klíč do plné podoby, rozhodli jsme se pro variantu Classic, bez komprese klíče. Varianty přichází s předem danými parametry, které odpovídají daným bezpečnostním kategoriím. My jsme zvolili parametry Ia[3], které odpovídají NIST bezpečnostní kategorii I a II.To je podobná náročnost jako nalezení klíče u AES128 nebo nalezení kolize u SHA256 [15].

[2] Ve druhém kole se tato verze jmenovala Cyclic variant, kterou ale přejmenovali, aby zabránili představě, že používá cyklickou strukturu.

[3] Toto jsou parametry platné pro druhé kolo, ve třetím kole se mírně změnily.

Nyní diskutujme pouze variantu Classic, bez komprese klíče a s parametry Ia. Nechť máme dvou vrstvou ($u = 2$) variantu Rainbow používající $m = 64$ kvadratických forem s $n = 96$ neznámými nad tělesem $\mathbb{F} = GF(2^{2^2}) = GF(16)$. Parametry zde jsou $(v_1, v_2, v_3) = (32, 64, 96)$, tedy velikosti vrstev jsou $(o_1, o_2) = (32, 32)$. Tato varianta má matice $\mathbf{S}$ a $\mathbf{T}$ ve tvaru:

$$\mathbf{S} = \mathbf{S}^{-1} = \begin{pmatrix} \mathbf{I} & \mathbf{S}' \\ \mathbf{O} & \mathbf{I} \end{pmatrix}, \tag{2.1}$$

$$\mathbf{T} = \begin{pmatrix} \mathbf{I} & \mathbf{T}^{(1)} & \mathbf{T}^{(2)} \\ \mathbf{O} & \mathbf{I} & \mathbf{T}^{(3)} \\ \mathbf{O} & \mathbf{O} & \mathbf{I} \end{pmatrix}, \ \mathbf{T}^{-1} = \begin{pmatrix} \mathbf{I} & \mathbf{T}^{(1)} & \mathbf{T}^{(4)} \\ \mathbf{O} & \mathbf{I} & \mathbf{T}^{(3)} \\ \mathbf{O} & \mathbf{O} & \mathbf{I} \end{pmatrix}, \tag{2.2}$$

kde $\mathbf{S} \in \mathbb{F}^{64\times64}$, $\mathbf{T} \in \mathbb{F}^{96\times96}$. Všechny podmatice mají rozměr $\mathbb{F}^{32\times32}$. Matice $\mathbf{O}$ je matice nul a $\mathbf{I}$ je matice identity. Dále $\mathbf{T}^{(4)} := \mathbf{T}^{(1)}\mathbf{T}^{(3)} - \mathbf{T}^{(2)}$. Centrální zobrazení je stejné jako v kapitole 1.1.1.

Protože se pohybujeme nad tělesem $GF(16)$, má jeden element 4 bity. Implementace je určená pro 32bitový procesor, tedy do 32bitového slova se vejde 8 prvků. Tyto prvky mohou být vynásobeny stejným prvkem paralelně – tj. operace násobení vektoru skalárem.

2.2.1 Implementace působení lineárních zobrazení S a T

Náš útok cílíme na zpracování násobení matice $\mathbf{S}^{-1}$, resp. $\mathbf{T}^{-1}$, se vstupním, resp. výstupním, vektorem. Analýza implementace nám může odhalit, na které mezivýpočty lze útok cílit. Působení lineárního zobrazení je implementováno jako násobení vektoru příslušnou maticí, která reprezentuje zobrazení. Násobení se skládá ze dvou částí. První se zpracují netriviální podmatice a následně se přičte samotný vektor k výsledku, který představuje jedničky na diagonále matice. Pořadí zpracování zde hraje důležitou roli. Algoritmus 2 popisuje výpočet $\mathbf{S}^{-1}\mathbf{h}$ a algoritmus 3 popisuje $\mathbf{T}^{-1}\mathbf{y}$. Obě funkce používají maticové násobení popsané v algoritmu 4. Vstupní inverzní matice do algoritmů jsou předpočítány a uloženy v soukromém klíči.

Algoritmus 2 Násobení matice $\mathbf{S}^{-1}$ vektorem $\mathbf{h}$

vstup: matice $\mathbf{S}' \in \mathbb{F}^{o_1\times o_2}$, vektor $\mathbf{h} \in \mathbb{F}^m$
výstup: vektor $\mathbf{x} = (\mathbf{S}^{-1}\mathbf{h}) \in \mathbb{F}^m$

1: $\mathbf{x} = \mathbf{h}$
2: $\mathbf{t} = \text{gfmat\_prod}(\mathbf{S}', \mathbf{h}_{o_1:o_1+o_2})$
3: $\mathbf{x}_{o_1:o_1+o_2} = \mathbf{x}_{o_1:o_1+o_2} + \mathbf{t}$
4: **vrať** $\mathbf{x}$

Algoritmus 3 Násobení matice $\mathbf{T}^{-1}$ vektorem $\mathbf{y}$

vstup: matice $\mathbf{T}^{(1)} \in \mathbb{F}^{v_1 \times o_1}$, $\mathbf{T}^{(4)} \in \mathbb{F}^{v_1 \times o_2}$, $\mathbf{T}^{(3)} \in \mathbb{F}^{o_1 \times o_2}$, vektor $\mathbf{y} \in \mathbb{F}^n$
výstup: vektor $\mathbf{z} = (\mathbf{T}^{-1}\mathbf{y}) \in \mathbb{F}^n$

1: $\mathbf{z} = \mathbf{y}$
2: $\mathbf{t} = \text{gfmat\_prod}(\mathbf{T}^{(1)}, \mathbf{y}_{1:v_1})$
3: $\mathbf{z}_{1:v_1} = \mathbf{z}_{1:v_1} + \mathbf{t}$
4: $\mathbf{t} = \text{gfmat\_prod}(\mathbf{T}^{(4)}, \mathbf{y}_{v_1+1:v_2})$
5: $\mathbf{z}_{1:v_1} = \mathbf{z}_{1:v_1} + \mathbf{t}$
6: $\mathbf{t} = \text{gfmat\_prod}(\mathbf{T}^{(3)}, \mathbf{y}_{v_1+1:v_2})$
7: $\mathbf{z}_{v_1+1:v_2} = \mathbf{z}_{v_1+1:v_2} + \mathbf{t}$
8: **vrať** $\mathbf{z}$

Algoritmus 4 `gfmat_prod`: Působení matice na vektor :

vstup: matice $\mathbf{A} \in \mathbb{F}^{s,u}$, vektor $\mathbf{x} \in \mathbb{F}^u$
výstup: vektor $\mathbf{t} = (\mathbf{A}\mathbf{x}) \in \mathbb{F}^s$
pozn.: v implementaci není $\mathbf{t}$ výstupem, ale je předán jako reference

1: $\mathbf{t} = \mathbf{0} \in \mathbb{F}^s$
2: $i = 1$
3: **dokud** $i \leq u$ **prováděj**
4: $\quad \mathbf{t} = \mathbf{t} + x_i \mathbf{A}_{:i}$
$\quad$ *pozn.:* i-tý sloupec matice $\mathbf{A}$ se násobí i-tou složkou vektoru $\mathbf{x}$
5: $\quad i = i + 1$
6: **konec dokud**
7: **vrať** $\mathbf{t}$

Z algoritmu 4 je patrné, že je zkonstruován, aby násobení mohlo být počítáno paralelně. Sloupec délky 32 elementů může být procesorem zpracován jako 4 paralelní násobení vektoru (o 8 prvcích) jedním skalárem. K tomu je přizpůsobeno i uložení samotného klíče, kde všechny matice jsou uloženy po sloupcích a ne po řádcích. Tento způsob výpočtu je důležitý pro odběrovou analýzu. Rovněž je důležité postřehnout (alg. 2 a 3), že diagonála matic se přičítá až na konci výpočtu. Pokud se v rámci této práce budeme bavit o paralelním zpracování, myslíme tím vždy zpracování více prvků v rámci většího registru/slova. Nemyslíme tím vícevláknový program.

2.3 Analýza zabezpečení proti postranním útokům

Další částí práce je návrh protiopatření, které budou v jisté míře zamezovat útokům pomocí postranní analýzy. Zde bych možná řešení rozdělil do dvou kategorií. Do první bych zařadil mechanismy, u kterých nepotřebujeme znát samotný algoritmus. To jsou obecné postupy skrytí postranního kanálu jako je skrývání v čase, v amplitudě, generování dodatečného šumu apod. [16, 17]

Tyto techniky vyžadují provádění nadbytečných operací, použití speciálního hardwaru apod.

Zaměříme se na protiopatření, která jsou postavena na znalosti samotného podpisového schématu a lze je implementovat změnou algoritmu. Základním takovým protiopatřením je aditivní maskování [9], které je velmi jednoduché a účinné. Z tohoto důvodu ji určitě zahrneme do této práce.

Další zajímavý koncept může být randomizace samotného klíče. Klíč nám zde hraje roli tajné informace a pokud nalezneme způsob, jakým lze klíč měnit, aniž bychom změnili celkový výsledek, dostaneme velice zajímavé schéma. Samotný proces podepisování je matematicky jednoduchý – obsahuje dvě lineární zobrazení a jedno kvadratické zobrazení. Pouze na tomto základě jde odvodit jednoduché maskování vstupu, které jsme zahrnuli v článku [18]. V této práci popíšeme složitější maskování, kde budeme generovat ekvivalentní klíče.

2.4 Analýza aplikovaných protiopatření

Po implementaci protiopatření, bude potřeba otestovat jejich účinnost. Zde se nám nabízí celá škála způsobů LA. Základní testy se zakládají na testovacích statistikách, jako je testování vzájemné informace, t-test [10] či χ^2-test [19].

Relativní novinkou na poli LA je využití neuronových sítí a hlubokého učení, které se teoreticky i prakticky dokáží naučit rozeznávat záznamy dle unikající informace [20]. Jedná se o klasifikační problém, kde nemusíme znát samotný algoritmus. Tuto metodu nepoužijeme, protože nám neposkytne mnoho kvantitativních informací.

Testování vzájemné informace testuje všechny statistické momenty, kdežto t-test pouze vybraný statistický moment. Jelikož jsme si vybrali pro maskování techniku aditivního maskování, pak má smysl volit t-test, kde tento test budeme cílit na patřičný řád.

Zde nastává největší problém této práce. Tím je paměťová a výpočetní složitost měření a vyhodnocení. Pro testování se musí naměřit řádově milióny záznamů o spotřebě a následné zpracování LA vyžaduje velký výkon. Pro urychlení snížíme parametry Rainbow z $(v_1, o_1, o_2) = (32, 32, 32)$ na $(8, 8, 8)$ (viz implementace testování 5.2).

Samotné snížení parametru nestačí. Jeden podpis (se sníženými parametry) trvá přibližně 15 ms při frekvenci 7,37 MHz, tím dostáváme přibližně 110000 bodů v jednom záznamu. Pokud naměříme milión záznamů a každý bod bude reprezentován 2 B, pak dostáváme na jedno měření zhruba 220 GB dat. Jelikož bychom rádi naměřili více prováděných algoritmů (pro nezabezpečenou verzi a pro jednotlivé opatření), dostáváme se na terabyty dat.

Z těchto důvodů musíme LA omezit jen na kritická místa algoritmu. Kritickým místem zde myslíme lineární zobrazení, na která cílíme útok. Jelikož aplikace lineárních zobrazení jsou velmi podobná, pak můžeme měřit pouze jedno zobrazení, kde pro jednoduchost vybereme zobrazení S. Samotné náso-

bení $\mathbf{S}^{-1}\mathbf{h}$ trvá přibližně 270 us. Při stejné konfiguraci tak dostáváme kapacitu přibližně 4 GB na milión záznamů.

Kapitola 3

Návrh útoku

Základní myšlenkou útoku je odhalit lineární matice soukromého klíče a následně dopočítat centrální zobrazení [14]. To znamená, že cílíme na matice $\mathbf{S}^{-1}$ a $\mathbf{T}^{-1}$, které mají pevně definovanou strukturu. Tajnou informací jsou samotné podmatice těchto matic. Náš útok se tím redukuje na odhalení netriviálních podmatic.

Útok jsme rozdělili na tři samostatné části, které se opakují pro každou podmatici. Útok I odhalí řádek i dané podmatice, následně útok II zjistí index odhaleného řádku při útoku I (determinuje index i). Tímto způsobem mohou být odhaleny pouze specifické řádky. Útok III využívá již odhalené řádky a rozšiřuje na ně svoji hypotézu, tedy útočíme na paralelizaci. Prvně se budeme zabývat útokem na podmatici $\mathbf{S}'$.

3.1 Útok na podmatici S′

Při výpočtu $\mathbf{x} = \mathbf{S}^{-1}\mathbf{h}$ využijeme znalosti algoritmu 2 a 4 a odvodíme, že náš útok musíme cílit na výpočet:

$$\mathbf{x}_{1:32} = \mathbf{S}' \cdot \mathbf{h}_{33:64} + \mathbf{h}_{1:32}, \tag{3.1}$$

kde každý prvek vektoru $\mathbf{x}_{1:32}$ můžeme vyjádřit jako

$$\forall i \in \{1, \ldots, 32\} : x_i = \sum_{j=1}^{32} (S'_{i,j} \cdot h_{j+32}) + h_i. \tag{3.2}$$

Nejdříve je vektor $\mathbf{x}$ nastaven na nulové hodnoty, následně se paralelně postupně přičítají násobky sloupců podmatice $\mathbf{S}'$ s daným skalárem h_{j+32} a na závěr se paralelně přičte vektor $\mathbf{h}_{1:32}$. Nyní zanedbáme paralelizaci a provedeme útok. Tabulka 3.1 popisuje hodnoty, na které cílíme naši hypotézu.

V celé tabulce používáme i pro index řádku. To děláme z toho důvodu, že nelze od sebe jednotlivé řádky odlišit. Žádnou hodnotou, kterou známe

Tabulka 3.1: Útok I: Odhalení řádku matice

Cílový prvek	Hodnota pro hypotézu
$S'_{i,1}$ and $S'_{i,2}$	$S'_{i,1} \cdot y_{33} + S'_{i,2} \cdot h_{34}$
$S'_{i,3}$	$\sum_{j=1}^{2}(S'_{i,j} \cdot h_{j+32}) + S'_{i,3} \cdot h_{35}$
$S'_{i,4}$	$\sum_{j=1}^{3}(S'_{i,j} \cdot h_{j+32}) + S'_{i,4} \cdot h_{36}$
⋮	⋮
$S'_{i,32}$	$\sum_{j=1}^{31}(S'_{i,j} \cdot h_{j+32}) + S'_{i,32} \cdot h_{64}$

(vektor $\mathbf{h}$) nedokážeme řádky od sebe rozeznat. Jinak řečeno první řádek je násoben stejným $\mathbf{h}$ jako řádek druhý atd.

Ze všeho nejdřív útočíme na dva prvky: $S'_{i,1}$ a $S'_{i,2}$. Útočit pouze na jeden prvek nemá smysl, protože v prvním sloupci matice $\mathbf{S}'$ se patrně nacházejí všechny prvky tělesa $GF(16)$. To znamená, že bychom dostali korelaci pro všechny hypotetické klíče (tj. 16 hypotéz na 32 prvků). Rozšířením útoku o jeden prvek dostáváme 256 hypotéz na 32 dvojic prvků. Zde vybereme nejsilnější korelaci a pokračujeme v útoku po jednom prvku, dokud neodhalíme celý řádek.

Při útoku na první dva prvky řádku může nastat chyba. Předpokládejme, že $S'_{i,2} = 0$. Pro prvky nad tělesem dostáváme hodnotu pro hypotézu:

$$S'_{i,1} \cdot y_{33} + S'_{i,2} \cdot h_{34} = S'_{i,1} \cdot y_{33} + 0 \cdot h_{34} = S'_{i,1} \cdot y_{33}. \tag{3.3}$$

Jinými slovy útočíme pouze na první prvek, což vede na jistou korelaci pro všechny hodnoty (viz výše). Naopak, pokud bude $S_{i,1} = 0$, útočíme pouze na druhý prvek, který bude rovněž korelovat pro všechny prvky, protože v jistou chvíli musí být tato hodnota vypočtena procesorem a to pro všechny řádky. Jedná se o okamžik, než se tento člen připočte k celkové sumě. Z tohoto dostáváme, že na začátku útoku I (první řádek tabulky 3.1) nemůžeme uvažovat nulové prvky.

Stejné tomu je i u ostatních prvků řádku, na který cílíme (tj. $S'_{i,j}, j \in \{3, 4, \ldots, 32\}$). Zde ale máme jednoduché řešení. V hypotéze stačí neuvažovat nulu a pokud žádná korelace není pozorována, pak je daný prvek nulový. Základem je fakt, že první dva prvky řádku (které již známe) nám zafixují index řádku, na který útočíme a zároveň znemožní útok na samotný j-tý člen sumy.

Nyní předpokládejme úspěšný útok I, tedy známe prvky jednoho řádku, ale neznáme samotný index řádku. Tento problém řeší útok II. Dle algoritmu 2 a rovnice 3.1 se po vypočítání maticového násobení připočte k výsledku vektor $\mathbf{h}_{1,32}$, který známe. Právě tyto prvky budou tvořit jádro útoku II. Tabulka 3.2 popisuje hodnoty, na které útočíme. Pokud bude kolerovat nějaké k, pak musí jistě platit, že $i = k$, čímž zjistíme index řádku. Pokud žádné k nekoreluje, patrně nastala chyba při útoku I. Tímto zároveň dostáváme i validaci odhaleného řádku.

Tabulka 3.2: Útok II: Odhalení indexu řádku

Cílový prvek	Hodnota pro hypotézu
$k \in \{1, \ldots, 32\}$	$\sum_{j=1}^{32}(S'_{i,j} \cdot h_{j+32}) + h_k$

Útok I a II provádíme opakovaně, dokud je to možné. Tímto způsobem odhalíme v průměru 28 řádků. Tato hodnota vychází z pravděpodobnosti výskytu dvou nenulových prvků za sebou (tj. $\left(\frac{15}{16}\right)^2$). Následně průměrný počet takových řádků v matici $\mathbf{S}'$ je

$$\text{Mean}\left(\text{BinomialDistribution}\left(32, \left(\frac{15}{16}\right)^2\right)\right) = 28{,}125. \tag{3.4}$$

Reálně tímto útokem není důležité odhalit takový počet řádků. Stačí nám odhalit dostatečný počet řádků, abychom mohli využít paralelizace pro útok III.

Pro útok III si musíme zavést pomocné značení, které odráží paralelizaci. Prvně si rozdělíme indexy řádků do množin, kde řádky v jedné množině jsou zpracovávány paralelně. Jelikož máme 32 bitovou implementaci a pracujeme se čtyř bitovými prvky, procesor zpracovává osm prvků zároveň:

$$Set_l \coloneqq \{8 \cdot (l-1) + 1, \ldots, 8 \cdot l\}. \tag{3.5}$$

Pro tyto množiny budeme používat index $l \in \{1, 2, 3, 4\}$. Pro kompaktnější zápis si označme hypotézu H_j^l pro danou množinu řádků l a pro j-tý sloupec:

$$H_j^l \coloneqq \sum_{i \in Set_l} \text{HW}\left(\sum_{k=1}^{j}(S'_{i,k} \cdot h_{k+32})\right). \tag{3.6}$$

H_j^l nám říká, jaká je Hammingova váha 32 bitového registru (suma přes osm řádků) v určitém mezivýsledku u j-tého sloupce při násobení $\mathbf{S}' \cdot \mathbf{h}_{32:64}$. Pokud je dané $S'_{i,k}$ neznámé (ještě nebylo odhaleno útokem), pak předpokládáme nulu, čímž se nám v hypotéze neprojeví. Do této hypotézy nezahrnujeme aktuální řádek, na který útočíme.

Útok III pro i-tý řádek, kde l je takové, aby $i \in Set_l$ popisuje tabulka 3.3. Útok se liší pouze v části odhalení samotných prvků řádku, kdežto identifikace indexu řádku probíhá zcela stejně jako při útoku II.

3.2 Útok na matici $\mathbf{T}^{-1}$

Matice $\mathbf{T}^{-1}$ se skládá ze tří netriviálních podmatic, které jsou součástí soukromého klíče. Následující rovnice popisují výpočet výsledného vektoru $\mathbf{z}$, které vycházejí ze struktury matice $\mathbf{T}^{-1}$ popsané v rovnici 2.2 a z algoritmu 3:

$$\begin{aligned} \mathbf{y}_{1:32} + \mathbf{T}^{(1)} \cdot \mathbf{y}_{33:64} + \mathbf{T}^{(4)} \cdot \mathbf{y}_{65:96} &= \mathbf{z}_{1:32}, \\ \mathbf{y}_{33:64} + \mathbf{T}^{(3)} \cdot \mathbf{y}_{65:96} &= \mathbf{z}_{33:64}, \\ \mathbf{y}_{65:96} &= \mathbf{z}_{65:96}. \end{aligned} \tag{3.7}$$

Tabulka 3.3: Útok III: Odhalení zbývajících řádků

Cílový prvek	Mezivýsledek I	Hodnota pro hypotézu
$S'_{i,1}$ and $S'_{i,2}$	$S'_{i,1} \cdot y_{33} + S'_{i,2} \cdot y_{34}$	$H_2^l + \mathrm{HW}(I)$
$S'_{i,3}$	$\sum_{j=1}^{2}(S'_{i,j} \cdot y_{j+32}) + S'_{i,3} \cdot y_{35}$	$H_3^l + \mathrm{HW}(I)$
$S'_{i,4}$	$\sum_{j=1}^{3}(S'_{i,j} \cdot y_{j+32}) + S'_{i,4} \cdot y_{35}$	$H_4^l + \mathrm{HW}(I)$
$\vdots$	$\vdots$	$\vdots$
$S'_{i,32}$	$\sum_{j=1}^{31}(S'_{i,j} \cdot y_{j+32}) + S'_{i,32} \cdot y_{64}$	$H_{32}^l + \mathrm{HW}(I)$
$k \in Set_l$	$\sum_{j=1}^{32}(S'_{i,j} \cdot y_{j+32}) + y_k$	$\mathrm{HW}(I)$

Tento výpočet se provádí na konci procesu podepisování a vektor $\mathbf{z}$ je jeho výstup, který tedy známe. Na rozdíl od útoku na matici $\mathbf{S}^{-1}$, zde neznáme vstup $\mathbf{y}$, ale díky poslední rovnici lze odvodit její část: $\mathbf{y}_{65:96}$.

Postup je stejný jako při předchozím útoku. Nejdřív musíme zjistit, kterým vektorem je daná podmatice násobena, následně útočit na řádky a identifikovat indexy řádků.

3.2.1 Útok na podmatici $\mathbf{T}^{(3)}$

Popišme si relevantní rovnici pro tento útok a identifikujme, které prvky známe, neznáme nebo chceme zjistit:

$$\underbrace{\mathbf{T}^{(3)}}_{\text{tajný klíč}} \cdot \underbrace{\mathbf{y}_{65:96}}_{\text{známe}} + \underbrace{\mathbf{y}_{33:64}}_{\text{neznáme}} = \underbrace{\mathbf{z}_{33:64}}_{\text{známe}}. \tag{3.8}$$

Díky konstrukci matice $\mathbf{T}^{-1}$ známe vektor $\mathbf{y}_{65:96}$, který se násobí tajným klíčem a pomocí kterého můžeme útok realizovat. Problém nastává při identifikaci řádku pomocí útoku II. Zde neznáme vektor $\mathbf{y}_{33:64}$, který se přičítá k výsledku maticového násobení. Pokud odhalíme řádek matice $\mathbf{T}^{(3)}$, pak můžeme s pomocí $\mathbf{z}_{33:64}$ dopočítat $\mathbf{y}_{33:64}$. Pokud bychom řádek odhalili špatně, pak se rovněž dopočítáme výsledku, ale hodnota nám nebude korelovat. Pro správný řádek dostaneme korelaci v závislosti na indexu řádku. Hodnota, pomocí které zjistíme index řádku, je:

$$\sum_{j=1}^{32}(T_{i,j}^{(3)} \cdot y_{j+65}) + z_{i+32} = y_{i+32}. \tag{3.9}$$

Podobně jako u podmatice $\mathbf{S}'$ můžeme využít útok III pro odhalení celé podmatice $\mathbf{T}^{(3)}$, kde po jejím odhalení můžeme dopočítat prostřední část vektoru $\mathbf{y}$:

$$\mathbf{y}_{33:64} = \mathbf{T}^{(3)} \cdot \mathbf{y}_{65:96} + \mathbf{z}_{33:64}. \tag{3.10}$$

3.2.2 Útok na podmatici $\mathbf{T}^{(4)}$

Postup je velmi podobný předchozímu útoku. Relevantní rovnice je:

$$\underbrace{\mathbf{y}_{1:32} + \mathbf{T}^{(1)} \cdot \mathbf{y}_{33:64}}_{\text{neznáme}} + \underbrace{\mathbf{T}^{(4)}}_{\text{tajný klíč}} \cdot \underbrace{\mathbf{y}_{65:96}}_{\text{známe}} = \underbrace{\mathbf{z}_{1:32}}_{\text{známe}}. \tag{3.11}$$

Opět dostáváme stejné schéma útoku. Tajnou matici násobíme známým vektorem a přičítáme neznámý vektor, kde známe hodnotu pravé strany. Opět využijeme útoku I, kde odhalíme jeden řádek. Následně z odhaleného řádku vypočteme hodnotu, která se přičítá a tím odhalíme index řádku. Tento proces opakujeme, dokud je to možné. Nakonec využijeme útok III a zaútočením na celé 32 bitové slovo odhalíme zbylé řádky.

3.2.3 Útok na podmatici $\mathbf{T}^{(1)}$

Předchozí rovnici si zde trochu upravíme, abychom dostali stejný tvar jako v předešlých útocích:

$$\begin{aligned} \mathbf{y}_{1:32} + \mathbf{T}^{(1)} \cdot \mathbf{y}_{33:64} + \mathbf{T}^{(4)} \cdot \mathbf{y}_{65:96} &= \mathbf{z}_{1:32}, \\ \underbrace{\mathbf{T}^{(1)}}_{\text{tajný klíč}} \cdot \underbrace{\mathbf{y}_{33:64}}_{\text{známe}} + \underbrace{\mathbf{y}_{1:32}}_{\text{neznáme}} &= \underbrace{\mathbf{z}_{1:32} + \mathbf{T}^{(4)} \cdot \mathbf{z}_{65:96}}_{\text{známe}}. \end{aligned} \tag{3.12}$$

Opět známe vektor násobený s maticí, známe pravou stranu a pro identifikaci řádku využijeme odhad $\mathbf{y}_{1:32}$.

3.3 Extrakce centrálního zobrazení F

Nyní předpokládejme úspěšné odhalení lineárních zobrazení S a T. Centrální zobrazení můžeme snadno získat jedním ze dvou způsobů.

Extrakce se znalostí veřejného klíče

Můžeme využít znalosti veřejného klíče $P = S \circ F \circ T$. Pro intuitivní představu lze říct, že P, stejně jako F, je 3D pole. V první dimenzi indexujeme jednotlivé polynomiální formy, ve zbylých dvou jednotlivé kvadratické členy. Prvek $F_{2,3,4}$ tedy označuje koeficient termu x_3x_4 ve druhé kvadratické formě centrálního zobrazení. Aplikací lineárního zobrazení T docílíme změnu báze vstupního vektoru a aplikací lineárního zobrazení S získáme změnu báze na výstupu. Odstranění lineárních zobrazení z P je tedy otázka maticového násobení. Pokud bychom měli dvě kvadratické formy, reprezentovány maticemi

$\mathbf{Q}_1$ a $\mathbf{Q}_2$, pak by přechod mezi bázemi vypadal takto:

$$\begin{aligned} F &= \begin{pmatrix} \mathbf{Q}_1 \\ \mathbf{Q}_2 \end{pmatrix}, \\ F \circ T &= \begin{pmatrix} \mathbf{T}^T\mathbf{Q}_1\mathbf{T} \\ \mathbf{T}^T\mathbf{Q}_2\mathbf{T} \end{pmatrix}, \\ S \circ F \circ T &= \begin{pmatrix} S_{1,1}\mathbf{T}^T\mathbf{Q}_1\mathbf{T} + S_{1,2}\mathbf{T}^T\mathbf{Q}_2\mathbf{T} \\ S_{2,1}\mathbf{T}^T\mathbf{Q}_1\mathbf{T} + S_{2,2}\mathbf{T}^T\mathbf{Q}_2\mathbf{T} \end{pmatrix}. \end{aligned} \tag{3.13}$$

Překvapivě může působit krok s aplikací zobrazení T. Matice $\mathbf{Q}_k$ není soustava lineárních polynomů, ale jedna kvadratická forma. Odtud plyne:

$$\mathbf{Q}_k(\mathbf{x}) = \mathbf{x}^T \cdot \mathbf{Q}_k \cdot \mathbf{x}. \tag{3.14}$$

Změnou báze vektoru $\mathbf{x}$ pomocí lineární transformace reprezentovanou maticí $\mathbf{T}$ dostáváme:

$$\mathbf{Q}_k(\mathbf{Tx}) = (\mathbf{Tx})^T \cdot \mathbf{Q}_k \cdot \mathbf{Tx} = \mathbf{x}^T\mathbf{T}^T\mathbf{Q}_k\mathbf{Tx} = (\mathbf{T}^T\mathbf{Q}_k\mathbf{T})(\mathbf{x}). \tag{3.15}$$

Zjištění F z P pomocí matic $\mathbf{S}$ a $\mathbf{T}$ je tedy snadný úkol.

Extrakce bez znalostí veřejného klíče

Druhou možností, jak získat centrální zobrazení F, je řešením soustavy lineárních rovnic. Během měření zcela jistě zaznamenáme mnoho vstupů a odpovídajících výstupů do algoritmu. Na základě nich a se znalostí matic $\mathbf{S}$ a $\mathbf{T}$ spočteme vstupy a výstupy centrálního zobrazení F, čímž dostáváme množinu rovnic ve tvaru:

$$y_{k-v_1} = \sum_{\substack{i,j \in V_l \\ i \leq j}} \alpha_{i,j}^{(k)} x_i x_j + \sum_{\substack{i \in V_l \\ j \in O_l}} \beta_{i,j}^{(k)} x_i x_j, \tag{3.16}$$

kde $l \in \{1, 2\}$. Jelikož známe vektor $\mathbf{x}$ i vektor $\mathbf{y}$, dostáváme lineární rovnice, kde neznámé jsou α a β. Takový systém umíme řešit i pro velké počty neznámých, ale v tomto případě lze rovnice rozdělit do nezávislých množin lineárních soustav a řešit neznámé postupně a zvlášť. K této metodě veřejný klíč vůbec nepotřebujeme.

Kapitola 4

Návrh protiopatření

Zde se zaměříme pouze na protiopatření založeném na maskování zpracovávaných hodnot úpravou algoritmu. Tyto metody jsou postavené na randomizaci mezivýsledků, které vedou v nemožnost odhadnout hodnotu v registru.

Prvním protiopatřením je randomizace samotného klíče, kde odvodíme ekvivalentní klíč, tedy klíč představující stejnou soustavu kvadratických rovnic, ale s jinými koeficienty. Pokud by útočník prováděl dynamickou odběrovou analýzu, neútočil by stále na stejný klíč, ale na množinu klíčů.

Další protiopatření jsou postavené na aditivním maskování d-tého řádu, tedy na rozdělení informace (hodnoty, na kterou útočník cílí) na $(d+1)$ rozdílných hodnot. Zde d sdílených hodnot nenesou informaci o původní hodnotě. Útočník tedy musí cílit na všech $(d+1)$ hodnot. Toto maskování je formulováno v kapitole 1.3.

Upozornění: V zájmu jednoduchosti mírně pozměníme indexaci prvků vektorů $\mathbf{h}$ a $\mathbf{y}$:

$$\mathbf{h} = \begin{pmatrix} h_{v_1+1} \\ h_{v_1+2} \\ \vdots \\ h_n \end{pmatrix} \qquad \mathbf{y} = \begin{pmatrix} y_{v_1+1} \\ y_{v_1+2} \\ \vdots \\ y_n \end{pmatrix} \tag{4.1}$$

Doteď jsme pracovali odděleně s lineárním zobrazením S nebo T. V následující části již pracujeme rovněž s centrálním zobrazením, kde právě kvadratické formy jsou indexovány tímto způsobem. Tím dostáváme jednodušší zápis $f^{(k)} = y_k$, místo $f^{(k)} = y_{k-v_1}$, kde $k \in \{v_1+1, v_1+2, \ldots, n\} = O_1 \cup O_2$. Tato notace zůstane po zbytek této práce.

4.1 Ekvivalentní klíč

Předpokládejme soukromý klíč $SK = (\mathbf{S}^{-1}, F, \mathbf{T}^{-1})$ a veřejný klíč $PK = (P)$. Pro účely maskování musíme zobecnit matice reprezentující lineární zobra-

zení, které byly definovány v sekci 2.2. Zobecnění se týká diagonál, kde nebudeme vyžadovat neutrální prvky, ale libovolné nenulové prvky. Zobrazení $S^{-1} : \mathbb{F}^{(o_1+o_2)} \to \mathbb{F}^{(o_1+o_2)}$ budeme reprezentovat maticí $\mathbf{S}^{-1}$ ve tvaru:

$$\begin{aligned}
\mathbf{S}^{-1} &= \begin{pmatrix} \mathbf{A}_1 & \mathbf{S}' \\ \mathbf{0} & \mathbf{A}_2 \end{pmatrix}, \\
\mathbf{A} &= \begin{pmatrix} \mathbf{A}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_2 \end{pmatrix}, \\
\mathbf{A}_1 &= \mathrm{diag}(a_{v_1+1}, a_{v_1+2}, \ldots, a_{v_2}), \\
\mathbf{A}_2 &= \mathrm{diag}(a_{v_2+1}, a_{v_2+2}, \ldots, a_n),
\end{aligned} \tag{4.2}$$

kde $\mathbf{S}' \in \mathbb{F}^{o_1 \times o_2}$, dále $\mathbf{A}_1, \mathbf{A}_2$ jsou diagonální matice s nenulovými prvky na diagonále, neboli $\forall i \in O_1 \cup O_2 : a_i \in \mathbb{F} \setminus \{0\}$ a $\mathbf{0}$ značí nulovou matici odpovídajících rozměrů. Lze si povšimnout, že prvky na diagonále jsou rovněž indexovány od hodnoty $v_1 + 1$.

Matici $\mathbf{T}^{-1}$ zobecníme stejným způsobem:

$$\begin{aligned}
\mathbf{T}^{-1} &= \begin{pmatrix} \mathbf{B}_1 & \mathbf{T}^{(1)} & \mathbf{T}^{(4)} \\ \mathbf{0} & \mathbf{B}_2 & \mathbf{T}^{(3)} \\ \mathbf{0} & \mathbf{0} & \mathbf{B}_3 \end{pmatrix}, \\
\mathbf{B} &= \begin{pmatrix} \mathbf{B}_1 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{B}_2 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{B}_3 \end{pmatrix}, \\
\mathbf{B}_1 &= \mathrm{diag}(b_1, b_2, \ldots, b_{v_1}), \\
\mathbf{B}_2 &= \mathrm{diag}(b_{v_1+1}, b_{v_1+2}, \ldots, b_{v_2}), \\
\mathbf{B}_2 &= \mathrm{diag}(b_{v_2+1}, b_{v_2+2}, \ldots, b_n),
\end{aligned} \tag{4.3}$$

kde $\mathbf{T}^{(1)} \in \mathbb{F}^{v_1 \times o_1}, \mathbf{T}^{(4)} \in \mathbb{F}^{v_1 \times o_2}, \mathbf{T}^{(3)} \in \mathbb{F}^{o_1 \times o_2}$ a $\mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_3$ jsou diagonální matice s nenulovými prvky na diagonále, tedy $\forall i \in \{1, \ldots, n\} : b_i \in \mathbb{F} \setminus \{0\}$.

Dále sjednotíme značení koeficientů centrálního zobrazení. Rovnice 1.3 obsahuje koeficienty $\alpha_{i,j}^{(k)}$ a $\beta_{i,j}^{(k)}$, které budeme nyní značit jednotným značením $\lambda_{i,j}^{(k)}$, kde:

$$\lambda_{i,j}^{(k)} := \begin{cases} \alpha_{i,j}^{(k)} & (k \in O_1), (i,j \in V_1), (i \leq j) \\ \beta_{i,j}^{(k)} & (k \in O_1), (i \in V_1), (j \in O_1) \\ \alpha_{i,j}^{(k)} & (k \in O_2), (i,j \in V_2), (i \leq j) \\ \beta_{i,j}^{(k)} & (k \in O_2), (i \in V_2), (j \in O_2) \\ 0 & jinak \end{cases} \tag{4.4}$$

Tímto získáváme jednodušší zápis kvadratické formy:

$$f^{(k)}(\mathbf{x}) = \sum_{i,j \in \{1,\ldots,n\}} \lambda_{i,j}^{(k)} \cdot x_i \cdot x_j. \tag{4.5}$$

Ekvivalentním soukromým klíčem nazveme klíč $\overline{SK}$, pokud splňuje, že jeho veřejný klíč $\overline{PK}$ je shodný s původním PK, tedy $\overline{PK} = PK$. Naše maskování bude fungovat na principu vygenerování ekvivalentního klíče, které následně můžeme použit místo původního klíče.

Maskou, která generuje ekvivalentní klíč, definujeme dvojici diagonálních matic $(\overline{\mathbf{A}}, \overline{\mathbf{B}})$, které mají nenulové diagonální prvky:

$$\begin{aligned} \overline{\mathbf{A}} &:= \mathrm{diag}(\overline{a}_{v_1+1}, \overline{a}_{v_1+2}, \ldots, \overline{a}_n), \\ \overline{\mathbf{B}} &:= \mathrm{diag}(\overline{b}_1, \overline{b}_2, \ldots, \overline{b}_n). \end{aligned} \tag{4.6}$$

Tyto diagonální matice jsou zavedeny stejně jako matice $\mathbf{A}$ a $\mathbf{B}$ v rovnicích 4.2 a 4.3. Stejným způsobem je dělíme na diagonální podmatice $\overline{\mathbf{A}}_1, \overline{\mathbf{A}}_2, \overline{\mathbf{B}}_1, \overline{\mathbf{B}}_2, \overline{\mathbf{B}}_3$.

Důkaz existence inverze matice $\overline{\mathbf{B}}$. Předpokládejme, že

$$\overline{\mathbf{B}}^{-1} := diag(\overline{b}_1^{-1}, \overline{b}_2^{-1}, \ldots, \overline{b}_n^{-1}). \tag{4.7}$$

$\overline{\mathbf{B}}^{-1}$ existuje, protože nenulové prvky nad tělesem mají inverzi a z definice $\overline{\mathbf{B}}$ jsou všechny prvky b_i nenulové. Následně lze vypočítat, že:

$$\overline{\mathbf{B}} \cdot \overline{\mathbf{B}}^{-1} = \overline{\mathbf{B}}^{-1} \cdot \overline{\mathbf{B}} = \mathbf{I}. \tag{4.8}$$

□

Matice $\overline{\mathbf{A}}$, resp $\overline{\mathbf{B}}^{-1}$, je určena pro zamaskování vektoru $\mathbf{y}$, resp. $\mathbf{x}$. Maskováním zde rozumíme prosté maticové násobení vektoru:

$$\begin{aligned} \mathbf{y} &\to \overline{\mathbf{A}} \cdot \mathbf{y}, \\ \mathbf{x} &\to \overline{\mathbf{B}}^{-1} \cdot \mathbf{x}, \end{aligned} \tag{4.9}$$

kde $\to$ naznačuje přechod z původní hodnoty do maskované hodnoty.

4.1.1 Generování ekvivalentního klíče

Mějme ekvivalentní klíč $\overline{SK} := (\overline{\mathbf{S}}^{-1}, \overline{F}, \overline{\mathbf{T}}^{-1})$ generovaný maskou $(\overline{\mathbf{A}}, \overline{\mathbf{B}})$. Pro splnění požadavku na maskování v rovnici 4.9 dostáváme pro matici $\overline{\mathbf{S}}^{-1}$ následující:

$$\begin{aligned} \mathbf{y} &= \mathbf{S}^{-1} \cdot \mathbf{h}, \\ \overline{\mathbf{A}} \cdot \mathbf{y} &= \overline{\mathbf{S}}^{-1} \cdot \mathbf{h}. \end{aligned} \tag{4.10}$$

Pokud si do druhé rovnice dosadíme $\mathbf{y}$ z první rovnice, dostáváme

$$\overline{\mathbf{A}} \cdot \mathbf{S}^{-1} \cdot \mathbf{h} = \overline{\mathbf{S}}^{-1} \cdot \mathbf{h}. \tag{4.11}$$

Pokud má být rovnice 4.11 splněna pro každé $\mathbf{h}$, pak musí platit:

$$\overline{\mathbf{S}}^{-1} = \overline{\mathbf{A}} \cdot \mathbf{S}^{-1} = \begin{pmatrix} \overline{\mathbf{A}}_1 \cdot \mathbf{A}_1 & \overline{\mathbf{A}}_1 \cdot \mathbf{S}' \\ \mathbf{0} & \overline{\mathbf{A}}_2 \cdot \mathbf{A}_2 \end{pmatrix}. \tag{4.12}$$

Stejným způsobem můžeme odvodit maskování matice $\mathbf{T}^{-1}$:

$$\begin{aligned} \mathbf{z} &= \mathbf{T}^{-1} \cdot \mathbf{x}, \\ \mathbf{z} &= \overline{\mathbf{T}} \cdot (\overline{\mathbf{B}}^{-1} \cdot \mathbf{x}), \end{aligned} \tag{4.13}$$

po dosazení a pro každé $\mathbf{x}$ musí platit:

$$\overline{\mathbf{T}}^{-1} = \mathbf{T}^{-1} \cdot \overline{\mathbf{B}} = \begin{pmatrix} \mathbf{B}_1 \cdot \overline{\mathbf{B}}_1 & \mathbf{T}^{(1)} \cdot \overline{\mathbf{B}}_2 & \mathbf{T}^{(4)} \cdot \overline{\mathbf{B}}_3 \\ \mathbf{0} & \mathbf{B}_2 \cdot \overline{\mathbf{B}}_2 & \mathbf{T}^{(3)} \cdot \overline{\mathbf{B}}_3 \\ \mathbf{0} & \mathbf{0} & \mathbf{B}_3 \cdot \overline{\mathbf{B}}_3 \end{pmatrix}. \tag{4.14}$$

Zbývá nám vyřešit pouze maskování centrálního zobrazení. Opět vyjdeme z původních kvadratických rovnic a z požadavku na maskování v rovnici 4.9. Jednotlivé maskované kvadratické formy budeme značit $\overline{f}^{(k)}, k \in \{v_1{+}1, \ldots, n\}$ a musí splňovat:

$$\overline{f}^{(k)}(\underbrace{\overline{\mathbf{B}}^{-1} \cdot \mathbf{x}}_{\text{maskovaný vstup}}) = [\underbrace{\overline{\mathbf{A}} \cdot \mathbf{y}}_{\text{maskovaný výstup}}]_k = \overline{a}_k \cdot y_k, \tag{4.15}$$

kde

$$\overline{f}^{(k)}(\mathbf{x}) = \sum_{i,j \in \{1,\ldots,n\}} \overline{\lambda}_{i,j}^{(k)} \cdot x_i \cdot x_j. \tag{4.16}$$

Pro splnění předchozích rovnic si $\overline{\lambda}$ zadefinuje jako:

$$\overline{\lambda}_{i,j}^{(k)} \coloneqq \overline{a}_k \cdot \overline{b}_i \cdot \overline{b}_j \cdot \lambda_{i,j}^{(k)}. \tag{4.17}$$

Důkaz validnosti rovnice 4.15 s použitím $\overline{\lambda}$ definované v rovnici 4.17.

$$\begin{aligned} [\overline{F}(\overline{\mathbf{B}}^{-1} \cdot \mathbf{x})]_k = \overline{f}^{(k)}(\overline{\mathbf{B}}^{-1} \cdot \mathbf{x}) &= \sum_{i,j \in \hat{n}} \overline{\lambda}_{i,j}^{(k)} \cdot (\overline{b}_i^{-1} \cdot x_i) \cdot (\overline{b}_j^{-1} \cdot x_j) = \\ &= \sum_{i,j \in \hat{n}} \overline{a}_k \cdot \overline{b}_i \cdot \overline{b}_j \cdot \lambda_{i,j}^{(k)} \cdot (\overline{b}_i^{-1} \cdot x_i) \cdot (\overline{b}_j^{-1} \cdot x_j) = \\ &= \sum_{i,j \in \hat{n}} \overline{a}_k \cdot \lambda_{i,j}^{(k)} \cdot x_i \cdot x_j = \overline{a}_k \cdot \sum_{i,j \in \hat{n}} \lambda_{i,j}^{(k)} \cdot x_i \cdot x_j = \\ &= \overline{a}_k \cdot f^{(k)}(\mathbf{x}) = \overline{a}_k \cdot y_k = [\overline{\mathbf{A}} \cdot \mathbf{y}]_k, \end{aligned} \tag{4.18}$$

kde $\hat{n} = \{1, 2, \ldots, n\}$ a $k \in \{v_1 + 1, \ldots, n\}$. Jelikož rovnice 4.18 platí pro každé k, pak platí:

$$\overline{F}(\overline{\mathbf{B}}^{-1} \cdot \mathbf{x}) = \overline{\mathbf{A}} \cdot \mathbf{y}. \tag{4.19}$$

□

4.1.2 Rekapitulace a zhodnocení randomizace klíče

Navrhli jsme způsob generování ekvivalentního soukromého klíče

$$\overline{SK} = (\overline{\mathbf{S}}^{-1}, \overline{F}, \overline{\mathbf{T}}^{-1}) \tag{4.20}$$

k původnímu klíči $SK = (\mathbf{S}^{-1}, F, \mathbf{T}^{-1})$, pomocí dvojice $(\overline{\mathbf{A}}, \overline{\mathbf{B}})$, kde

$$\begin{aligned}
\overline{\mathbf{A}} &\coloneqq \mathrm{diag}(\overline{a}_{v_1+1}, \overline{a}_{v_1+2}, \ldots, \overline{a}_n), \\
\overline{\mathbf{B}} &\coloneqq \mathrm{diag}(\overline{b}_1, \overline{b}_2, \ldots, \overline{b}_n), \\
\overline{\mathbf{S}}^{-1} &\coloneqq \overline{\mathbf{A}} \cdot \mathbf{S}^{-1}, \\
\overline{\mathbf{T}}^{-1} &\coloneqq \mathbf{T}^{-1} \cdot \overline{\mathbf{B}}, \\
\overline{F} &\coloneqq (\overline{f}^{(v_1+1)}, \overline{f}^{(v_1+2)}, \ldots, \overline{f}^{(n)}), \\
\overline{f}^{(k)}(\mathbf{x}) &\coloneqq \sum_{i,j \in \{1,\ldots,n\}} \overline{\lambda}_{i,j}^{(k)} \cdot x_i \cdot x_j, \\
\overline{\lambda}_{i,j}^{(k)} &\coloneqq \overline{a}_k \cdot \overline{b}_i \cdot \overline{b}_j \cdot \lambda_{i,j}^{(k)},
\end{aligned} \tag{4.21}$$

kde diagonální prvky $\overline{\mathbf{A}}$ a $\overline{\mathbf{B}}$ jsou nenulové.

Pokud chceme implementovat randomizaci klíče, musíme lehce pozměnit referenční implementaci, aby podporovala zobecněné lineární matice. Konkrétně se jedná o přidání dvou násobení vektorů po složkách během počítání lineárních zobrazení. Další implementace je zapotřebí pro samotný přepočet klíče. Každý prvek klíče je potřeba vynásobit danou hodnotou, tedy implementace je jednou průchozí přes celý klíč. Klíč se zobecněnými $\mathbf{S}$ a $\mathbf{T}$ je paměťově náročnější o $m + n$ prvků, tedy pro parametry $(v_1, o_1, o_2) = (32, 32, 32)$ dostáváme navíc $(96 + 64) * 4/8 = 80$ bytů. Oproti velikosti klíče, který má při těchto parametrech 93 kB, je rozdíl zanedbatelný. Nutno rovněž podotknout, že kvadratické polynomy musí být kvadratické formy, tedy nesmí obsahovat lineární nebo absolutní členy.

4.1.3 Ekvivalentní klíče jako multiplikativní maskování

Během procesu generování ekvivalentního klíče musíme každou hodnotu soukromého klíče vynásobit náhodně vygenerovanou nenulovou hodnotou. V případě zobrazení $\mathbf{S}$ násobíme každý řádek náhodnou maskou $\overline{a}_i$ a v případě matice $\mathbf{T}$ násobíme každý sloupec náhodnou hodnotou $\overline{b}_i$. Centrální zobrazení je vynásobeno trojicí hodnot, které dohromady dávají nenulovou náhodnou hodnotu. Tímto dostáváme multiplikativní masku diskutovanou v kapitole 1.4. Jedná se o masku pouze prvního řádu, už jen proto, že zde máme mnoho hodnot, které jsou vynásobeny stejnou maskou.

V kapitole 1.4 jsme diskutovali rovněž útok na nulové prvky. Tomuto omezení se zde neubráníme a výsledkem je, že na ekvivalentní klíče lze zaútočit útokem prvního řádu, ale pouze na nulové hodnoty. Tento únik informace bychom měli pozorovat v následném testování.

4.2 Maskování lineárních zobrazení

Během útoku využíváme znalosti vektoru, který se násobí konstantní (tajnou) maticí. Díky této znalosti následně dokážeme odhalit samotnou matici, která tvoří část soukromého klíče. Pro znemožnění tohoto útoku můžeme buď randomizovat vektor tak, abychom ho v době násobení neznali, nebo můžeme randomizovat samotnou matici, aby nebyla konstantní. Podívejme se nejdřív na randomizaci samotné matici pomocí aditivního maskování.

4.2.1 Maskování lineární matice

Maskování si zavedeme obecně. Mějme matici $\mathbf{A}$, vstupní vektor $\mathbf{x}$ a libovolné $n \in \mathbb{N}$. Dále uvažujme lineární transformaci:

$$\mathbf{y} = \mathbf{A}\mathbf{x}, \tag{4.22}$$

kde vektor $\mathbf{x} \in \mathbb{F}^n$ známe a matice $\mathbf{A} \in \mathbb{F}^{n \times n}$ je tajná matice. S využitím aritmetického maskování nad lineární transformací dostáváme rozdělení samotné matice na $d+1$ sdílených matic:

$$\mathbf{A} = \bigoplus_{i=1}^{d+1} \mathbf{A}_i. \tag{4.23}$$

Pak triviálně platí:

$$\mathbf{y} = \mathbf{A}\mathbf{x} = (\bigoplus_{i=1}^{d+1} \mathbf{A}_i)\mathbf{x} = \bigoplus_{i=1}^{d+1} (\mathbf{A}_i\mathbf{x}). \tag{4.24}$$

Tímto způsobem můžeme maskovat matice $\mathbf{S}^{-1}$ a $\mathbf{T}^{-1}$.

4.2.2 Maskování vstupu do lineární transformace

Nyní použijeme stejnou transformaci, ale místo maskování citlivých dat randomizujme vstup. Tzn., vytvořme sdílené hodnoty pro samotný vstupní vektor:

$$\mathbf{y} = \mathbf{A}\mathbf{x} = \mathbf{A}\bigoplus_{i=1}^{d+1} \mathbf{x}_i = \bigoplus_{i=1}^{d+1} (\mathbf{A}\mathbf{x}_i). \tag{4.25}$$

Toto schéma má ovšem zásadní slabinu, přes kterou bude unikat informace. Předpokládejme dvě různé matice $\mathbf{A}, \mathbf{B} \in \mathbb{F}^{n \times n}, \mathbf{x}, \mathbf{v}, \mathbf{w} \in \mathbb{F}^n, n \in \mathbb{N}$ a platí:

$$A_{i,j} = 0 \quad \wedge \quad B_{i,j} \neq 0 \quad \wedge \quad \mathbf{x} = \mathbf{v} + \mathbf{w}, \tag{4.26}$$

kde $i, j \in \{1, \ldots, n\}$. Matice $\mathbf{A}, \mathbf{B}$ představují různé tajné klíče, vektor $\mathbf{x}$ je vstup a vektory $\mathbf{v}$ a $\mathbf{w}$ jsou sdílené hodnoty, mezi které je randomizován vstup.

Během počítání $\mathbf{A}\mathbf{x} = \mathbf{A}\mathbf{v} + \mathbf{A}\mathbf{w}$ a $\mathbf{B}\mathbf{x} = \mathbf{B}\mathbf{v} + \mathbf{B}\mathbf{w}$ je potřeba v jistém okamžiku vypočítat $A_{i,j} \cdot v_j = 0 \cdot v_j = 0$ a pro druhý klíč hodnotu $B_{i,j} \cdot v_j \in GF(16)$.

V případě prvního klíče **A** máme výslednou hodnotu z konstantního rozdělení (při každém vstupu) a v případě klíče **B** dostáváme výsledek z rovnoměrného rozdělení. Tyto rozdílné rozdělení lze využít pro získání informace ze zařízení a Welchův t-test by tyto rozdíly detekoval.

Z výše popsaných důvodu jsme se rozhodli implementovat pouze sdílené hodnoty nad samotnou maticí, kde bychom neměli pozorovat žádný únik informace. Verze, kde je sdílená hodnota vytvořená nad vektorem, vede na stejnou zranitelnost jako máme u ekvivalentních klíčů.

Kapitola 5

Implementace

5.1 Implementace útoku

Pro účely útoku využíváme zařízení ChipWhisperer-Lite, který obsahuje 32bitový mikrokontrolér STM32F303 postavený na ARM Cortex-M4. ChipWhisperer obsahuje integrovaný 10 bitový A/C převodník se vzorkovací frekvencí 105 MS/s, využívající metodu synchronního vzorkování [21]. Tento převodník je určen právě pro zaznamenávání spotřeby připojeného mikrokontroléru.

Jelikož pracujeme s výkonným 32bitovým RISC procesorem, referenční implementace může být použita téměř v původní verzi. Provedené změny pouze zmenšují velikost programu a urychlují podpis. První změnou je odebrání náhodného generátoru postaveného na AES kryptosystému. Tento generátor nahrazujeme lineárním kongruenčním generátorem, jehož počáteční stav nastavuje řídící počítač, ke kterému je ChipWhisperer připojen. Druhá změna se týká randomizace vstupu. Dle rovnice 1.7 se nejdřív randomizuje vstup pomocí hašovací funkce, který se následně zpracovává. Tento krok vynecháváme, ale na oplátku musíme zaručit, že náš vstup bude z rovnoměrného rozdělení (nesmíme si nastavit námi zvolený vstup a ani nesmí být z jiného pravděpodobnostního rozdělení). Třetí změnou bylo odebrání cyklů s variabilním počtem průchodů. Toto je důležité pro zarovnání záznamů o spotřebě. Díky posledním dvou změnám je možné, že výsledný podpis nebude validní (pro daný vstup neexistuje řešení centrálního zobrazení). Toto je potřeba řešit pouze v případě měření centrálního zobrazení. V našem případě útočíme pouze na lineární zobrazení, které mají vždy řešení, jejich výpočet je vždy validní a nepoužíváme žádné hodnoty, které by tímto mohly být ovlivněny.

Samotný útok je implementován v systému Jupyter, postaveném na jazyku Python. ChipWhisperer je dodáván se základními funkcemi pro práci se zařízením, jako je nahrávání programu, komunikace se zařízením přes sériovou linku a stahování naměřených dat. Samotné měření je tedy otázka volání knihovních funkcí. Naměřené záznamy jsou následně uloženy na disku a vý-

počet klíče se provádí odděleně offline. Největší potíž s měřením byl fakt, že úložný prostor pro záznamy je příliš malý pro zaznamenání celého maticového násobení. Proto samotné měření muselo být provedeno mnohonásobně, kde se vždy měřila jiná část násobení matice s vektorem. Násobení jedné podmatice o rozměrech 32×32 prvků muselo být provedeno třikrát. Tento fakt značně prodloužil dobu měření.

Výpočet klíče je rovněž vytvořen v Jupyter notebooku. Útok je implementován přesně podle popisu v návrhu útoku.

5.2 Implementace protiopatření

Protiopatření byla implementována do samotné referenční implementace. Pro účely LA jsme nuceni snížit parametry programu, které určují bezpečnost Rainbow. Zde mluvím o parametrech (v_1, v_2, v_3), které mají přímý vliv na velikost matic lineárních zobrazení a velikost centrálního zobrazení. Snížením parametrů jsme dosáhli radikálního zmenšení doby výpočtu (teoreticky se jedná o kubické zrychlení). Samotná implementace má tyto parametry nastavené jako konstanty v kódu, takže tato úprava nebyla příliš náročná.

Parametry jsme nastavili z doporučených hodnot $(v_1, o_1, o_2) = (32, 32, 32)$ na nejnižší možné parametry, které lze bez úprav implementovaných protiopatření použít, tedy $(8, 8, 8)$. Tímto jsme zachovali i plné paralelní zpracování.

5.2.1 Výpočet ekvivalentního klíče

Výpočet ekvivalentního (maskovaného) klíče může být prováděn mimo hlavní část algoritmu, kde pouze namísto původního klíče vložíme klíč ekvivalentní.

Pro uložení zobecněné verze soukromého klíče je potřeba dodatečné místo. Jedná se o dvě diagonální matice $\overline{\mathbf{A}}$ a $\overline{\mathbf{B}}$ o rozměrech $m \times m$ a $n \times n$. Jelikož se jedná o diagonální matice, tak se dají uložit jako pole o rozměrech m a n. SK je tedy paměťově vetší právě o $m + n$ prvků $GF(16)$.

Matice $\mathbf{S}^{-1}$ a $\mathbf{T}^{-1}$ je dají přepočítat velmi jednoduše. V paměti jsou všechny podmatice lineárních transformací uloženy po sloupcích.

Výpočet $\overline{\mathbf{A}} \cdot \mathbf{S}^{-1}$ zahrnuje dva výpočty. Nejdříve se vynásobí diagonála původní matice diagonálou $\overline{\mathbf{A}}$. Poté se přepočítá podmatice $\mathbf{S}'$, podle $\overline{\mathbf{A}}_1 \cdot \mathbf{S}'$. Celková složitost tohoto výpočtu je $(m + o_1 o_2)/p$, kde p je počet prvků v jednom 32bitovém registru.

Výpočet $\overline{\mathbf{T}}^{-1}$ zahrnuje rovněž vynásobení diagonál $\mathbf{T}^{-1}$ a $\overline{\mathbf{B}}$. Následně se násobení diagonální podmatice a podmatice $\mathbf{T}^{-1}$ musí provádět třikrát (pro každou podmatici zvlášť). Celková složitost je tedy $(n + v_1 o_1 + v_1 o_2 + o_1 o_2)/p$.

Mnohem zajímavější je přepočet centrálního zobrazení. V návrhu protiopatření máme přesně uvedeno, co je potřeba vypočítat. Začneme tím, jak je centrální zobrazení uloženo v paměti. V rovnici 1.6 máme uvedenou maticovou reprezentaci centrálního zobrazení. Tímto způsobem jsou nenulové prvky

uloženy v paměti po sloupcích, ale ukládají se stejné prvky za sebou napříč všemi maticemi. Tzn., že pokud si indexy koeficientů vyjádříme jako tří ciferné číslo v pořadí (i, j, k), pak jsou seřazeny od nejmenšího po největší. To ale platí pouze pro první vrstvu. Ukládání centrálního zobrazení je poměrně komplikované pro vysvětlení, proto čtenáře odkážeme na samotný projekt, kde je oficiální dokumentace [22]. Komplikovanost ukládání zde rozhodně není na škodu. Díky specifickému pořadí může být centrální zobrazení dobře zpracováno a i náš ekvivalentní klíč může být rychle vypočítán. Celý přepočet centrálního zobrazení zahrnuje opakované volání jedné funkce kubické složitosti. Celý přepočet je jednou průchozí přes celé centrální zobrazené, proto složitost jednoduše odvodíme podle velikosti centrálního zobrazení jako $\approx 7(o_{max})^3$, kde $o_{max} = max(v_1, o_1, o_2)$.

5.2.2 Algebraické maskování lineární transformace

Samotná implementace algebraického maskování prvního řádu je jednoduchá. Chceme vytvořit dvě sdílené matice $\mathbf{S}'_1, \mathbf{S}'_2$ pro matici $\mathbf{S}'$, která tvoří naší citlivou hodnotu. Algoritmus 5 popisuje generování masky a algoritmus 6 popisuje aplikování lineární transformace se sdílenými hodnotami.

Algoritmus 5 Generování sdílených matic

vstup: citlivá informace ve formě matice $\mathbf{S}' \in \mathbb{F}^{o_1 \times o_2}$
výstup: dvě sdílené matice $\mathbf{S}'_1, \mathbf{S}'_2 \in \mathbb{F}^{o_1 \times o_2}$

1: $\mathbf{S}'_1 \xleftarrow{R} \mathbb{F}^{o_1 \times o_2}$ (matice s náhodnými prvky)
2: $\mathbf{S}'_2 = \mathbf{S}'_1 - \mathbf{S}'$ (zde máme + jako xor, tedy - je stejná operace jako +)
3: **vrať** $\mathbf{S}'_1, \mathbf{S}'_2$

Algoritmus 6 Aplikování lineární transformace pomocí sdílených hodnot

vstup: sdílené hodnoty ve formě matic $\mathbf{S}'_1, \mathbf{S}'_2 \in \mathbb{F}^{o_1 \times o_2}$,
vstupní vektor $\mathbf{h}_{o_1+1,m} \in \mathbb{F}^{o_2}$
výstup: vektor $\mathbf{y} = \mathbf{S}' \cdot \mathbf{h}_{o_1+1,m}$

1: $\mathbf{y} = \mathbf{S}'_1 \cdot \mathbf{h}_{o_1+1,m}$
2: $\mathbf{t} = \mathbf{S}'_2 \cdot \mathbf{h}_{o_1+1,m}$
3: $\mathbf{y} = \mathbf{y} + \mathbf{t}$
4: **vrať** $\mathbf{y}$

Z předchozích algoritmů 5 a 6 lze vidět, že přidaná složitost je pro vygenerování sdílené hodnoty jedno sčítání matic a pro aplikování, zde máme o jedno násobení maticí s vektorem navíc. To je dvojnásobný nárůst složitosti. V porovnán s celým procesem podepisování, se jedná o malý nárůst. Tato metoda je velice účinná proti útoku prvního řádu.

5.2.3 Porovnání implementací

Celkem jsme implementovali dvě protiopatření: ekvivalentní klíče a aditivní maskování matice $\mathbf{S}^{-1}$ a $\mathbf{T}^{-1}$. Tabulka 5.1 popisuje zpomalení implementace oproti původní nezabezpečené verzi. Jednotlivé údaje jsou měřeny pro různé parametry Rainbow. Tyto údaje jsme měřili na čisté implementaci algoritmu Rainbow tak, jak je popsaný v kapitole 5.1, tedy neobsahuje například počítání hašovací funkce. Hodnoty v tabulce jsou vypočítány jako poměr doby provádění zabezpečeného podpisu vůči době provádění nezabezpečené verze.

Tabulka 5.1: Zpomalení implementací vzhledem k parametrům a původní verzi

(v_1, o_1, o_2)	$(8, 8, 8)$	$(16, 16, 16)$	$(32, 32, 32)$
ekvivalentní klíče	2.3	4.3	5.9
aditivní maskování	1.2	1.02	1.004

Z tabulky 5.1 je vidět, že vytvoření ekvivalentního klíče je náročnější, než samotné generování podpisu. Při zabezpečení pomocí ekvivalentního klíče je zpomalení způsobenou pouze přepočtem klíče na klíč ekvivalentní. To zahrnuje vynásobení každé hodnoty soukromého klíče náhodnou hodnotou, dle kapitoly 4.1.2. Implementace není optimalizována, ale zároveň se nejedná o naivní implementaci.

Zpomalení při zabezpečení pomocí aditivního maskování je způsobeno dvojitým výpočtem násobení matice s vektorem a následného součtu výsledných vektorů. Z tabulky 5.1 vidíme, že s rostoucími parametry je zpomalení kvůli maskování nižší. To je způsobenou výpočtem inverze centrálního zobrazení, které má vyšší výpočetní složitost.

Kapitola 6

Testování

6.1 Realizace útoku

Pro prvotní útok volíme 20000 záznamů o spotřebě pro útok na první dva prvky v řádku a 5000 záznamů na ostatní prvky. Pro tentokrát volíme vyšší počet záznamů, aby byla jistota odhalení klíče. Pokud by klíč nebyl odhalen, byla by chyba pravděpodobně v samotné implementaci útoku. Útok lze rozhodně provést s nižším počtem prvků, ale v prvotní fázi útoku nám implementace nefungovala a zvýšení počtu záznamů je logický krok, jak se ujistit, že chyba není kvůli šumu apod.

Měření musíme provádět mnohokrát, protože velikost záznamu je delší než velikost bufferu v ChipWhispereru. Tabulka 6.1 popisuje naměřená data, včetně velikosti a doby měření. Jeden záznam obsahuje 24000 bodů spotřeby.

Celková doba měření je nepříjemně dlouhá. Na druhou stranu se nejedná o nerealizovatelnou dobu čekání a optimalizace by nutně neušetřila mnoho času, zvláště, když toto úplné měření provádíme pouze jednou.

Útoky I a II jsou schopné odhalit zhruba polovinu řádků z každé množiny řádků, které se zpracovávají paralelně (tyto množiny jsou definované jako set_l v rovnici 3.5). Tím dostáváme dostatečné množství řádků pro útok III.

Útok III správně odhaluje zbylé řádky. Útok je tedy úspěšný a odhalili jsme jak matici $\mathbf{S}^{-1}$, tak matici $\mathbf{T}^{-1}$. Správnost matic kontrolujeme oproti opravdovému soukromému klíči a samotnou centrální mapu nepočítáme.

Výpočet útoku se provádí zhruba 40 minut na jednu podmatici, tedy celý útok trvá zhruba 2,5 hodiny. Žádné významné optimalizační metody nebyly implementovány. Samozřejmě zde byla snaha zbytečně neprovádět redundantní výpočty. Možným radikálním urychlením by byla implementace na nižší úrovni abstrakce, vhodně předzpracovat data, použít paralelní výpočty apod.

Tabulka 6.1: Seznam naměřených dat

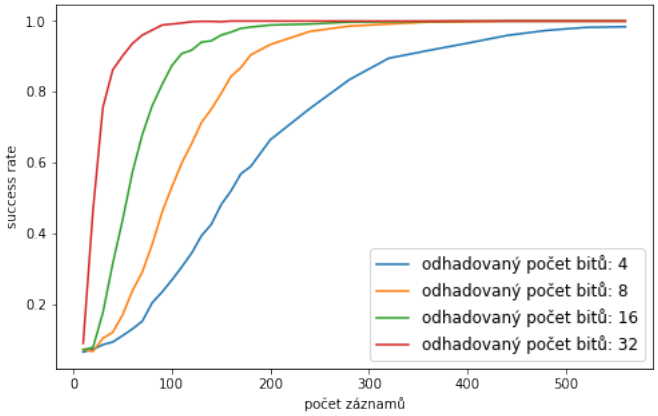
Cíl měření	počet záznamů	doba měření [min]	velikost dat [GB]
mat. $\mathbf{S}'$, sloupce 1-12	20000	180	3,84
mat. $\mathbf{S}'$, sloupce 13-24	5000	45	0,96
mat. $\mathbf{S}'$, sloupce 25-32	5000	45	0,96
mat. $\mathbf{T}^{(1)}$, sloupce 1-12	20000	180	3,84
mat. $\mathbf{T}^{(1)}$, sloupce 13-24	5000	45	0,96
mat. $\mathbf{T}^{(1)}$, sloupce 25-32	5000	45	0,96
mat. $\mathbf{T}^{(4)}$, sloupce 1-12	20000	180	3,84
mat. $\mathbf{T}^{(4)}$, sloupce 13-24	5000	45	0,96
mat. $\mathbf{T}^{(4)}$, sloupce 25-32	5000	45	0,96
mat. $\mathbf{T}^{(3)}$, sloupce 1-12	20000	180	3,84
mat. $\mathbf{T}^{(3)}$, sloupce 13-24	5000	45	0,96
mat. $\mathbf{T}^{(3)}$, sloupce 25-32	5000	45	0,96
celkem	120000	18 hodin	18,2

6.2 Vyhodnocení útoku

Vyhodnocení provedeme pomocí metriky success rate, která je definovaná v kapitole 1.2.1. Pro tyto účely je potřeba naměřit nová data, dle zvolené metodologie. Základním rozdílem je, že musíme naměřit nezávisle různé sady dat s jinými klíči. Jelikož klíč je součástí samotného programu (kvůli paměťovým nárokům nebylo možné uložit klíč na zásobník mikrokontroléru, který používáme pro útok), pak jsme zvolili ulehčení, kde nepoužíváme přímo success rate definovaný v [8], ale za použití pouze jednoho klíče útočíme na náhodně zvolené prvky matice $\mathbf{S}'$ opakovaně v nezávislých sadách.

Pro vyhodnocení jsme použili 20000 záznamů o spotřebě, naměřené nad jedním klíčem. Tyto záznamy jsme rozdělili do nezávislých 33 sad. Nyní pracujeme nad každou sadou zvlášť, kde vypočítáme success rate pro náhodně zvolené prvky. Řád success rate volíme 1, což značí, že experiment je úspěšný právě tehdy, když jsme odhalili správný klíč. Výsledky z jednotlivých sad následně zprůměrujeme. Obrázek 6.1 popisuje success rate v závislosti na počtu použitých záznamů o spotřebě. V grafu jsou vykresleny 4 křivky lišící se počtem bitů v hypotéze. 4 bity značí hypotézu o jednom prvku. To je Útok I na sloupce s indexem > 2. Odhadovaný počet bitů 8 je útok na jeden prvek se znalostí jednoho prvku z předchozího útoku. Toto je útok III. Stejně tak cílení na 16 bitů, kde známe další 3 prvky a 32 bitů, kde známe všechny prvky ze stejného setu.

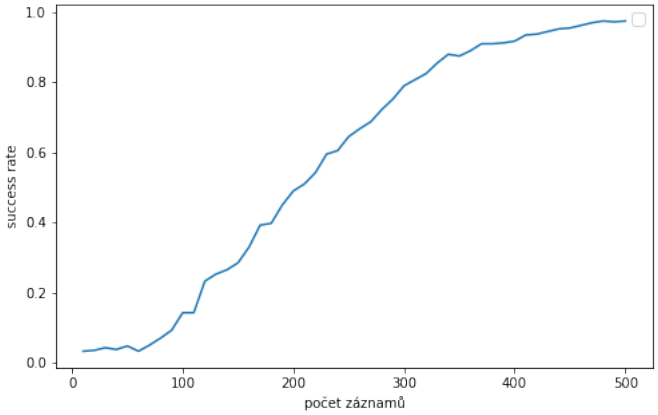
Z obrázku 6.1 je očividné, že cílení na větší počet bitů nám zpřesňuje hypotézu, čímž dostáváme lepší výsledky. Pro útok I s mírou úspěchu 0.75/0.95 potřebujeme změřit průměrně 240/440 záznamů. Pro dosažení míry úspěchu



Obrázek 6.1: Success rate pro útok I a III

0.75/0.95 u útok III s jedním, třemi, nebo sedmi známými prvky, které se zpracovávají paralelně, potřebujeme mít přibližně 140/240, 80/150, nebo 30/70 záznamů o spotřebě.

Zatím jsme opomenuli útok II, to je určení indexu řádku, na který jsme útok provedli. Zde opět využijeme podobného přístupu výpočtu success rate jako u předchozího útoku. Využijeme opět 20000 naměřených záznamů o spotřebě, které rozdělíme do 40 sad. Pro tyto sady budeme měřit míru úspěchu nezávisle a nakonec výsledky zprůměrujeme. Výsledky jsou zobrazeny na obrázku 6.2. Zde máme míru úspěchu 0.75/0.95 přibližně pro 290/440 záznamů o spotřebě. Lze si povšimnout, že výsledky jsou velmi podobné útoku I.



Obrázek 6.2: Success rate pro útok II

6.3 Testování protiopatření

Měříme dvě různé implementace Rainbow s parametry $(v_1, o_1, o_2) = (8, 8, 8)$, viz kapitola 5.2. Jedna implementace náhodně pomocí PRNG volí mezi nezabezpečeným podpisem a mezi podpisem využívající ekvivalentní klíč podle kapitoly 4.1. Druhá implementace náhodně volí mezi podpisem zabezpečeným aditivním rozkladem matic, popsaným v kapitole 4.2, a mezi podpisem zabezpečeným oba způsoby, jak aditivní maskou, tak ekvivalentním klíčem. Obě implementace navíc náhodně při podpisu volí mezi čtyřmi různými soukromými klíči. Celkem tedy měříme čtyři varianty:

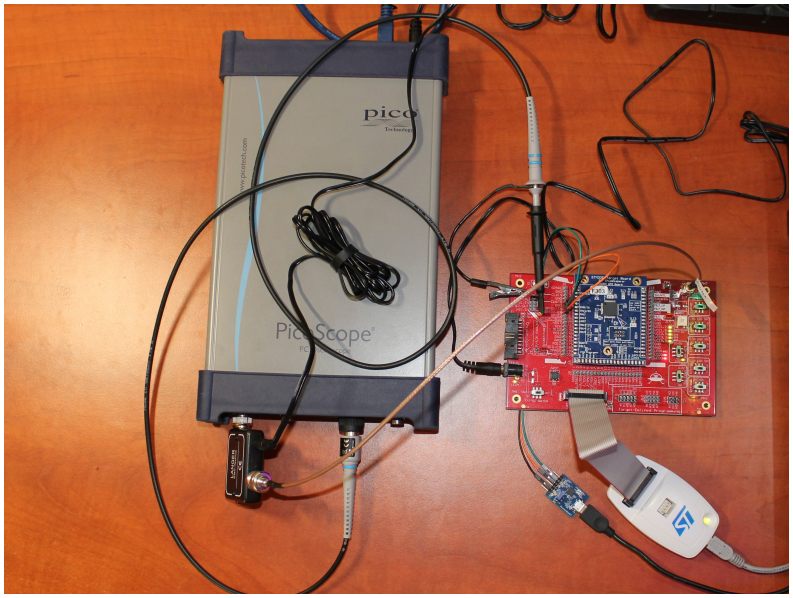
- nezabezpečený podpis,
- podpis zabezpečený generováním ekvivalentního klíče (multiplikativní maska),
- podpis zabezpečený aditivním rozkladem lineárních zobrazení (aditivní maska),
- podpis zabezpečený oběma výše zmíněnými metodami,

každá se čtyřmi různými klíči. Na obou implementacích měříme 515200 podpisů, celkem tedy 1030400 podpisů. Po rozdělení tedy každá sada průběhů obsahuje průměrně 64400 záznamů (4 varianty zabezpečení, 4 klíče). Vyhodnocení úniku informace provádíme pomocí Welchova t-testů s hypotézou o rovnosti středních hodnot dvou sad naměřených s různými klíči. Cílem testu je rozlišení použití různých soukromých klíčů.

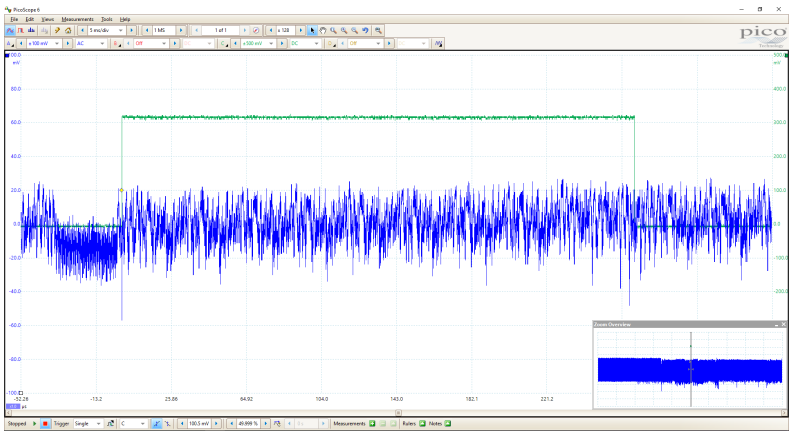
Měřené implementace spouštíme v mikrokontroléru STM32F3 na platformě ChipWhisperer CW308 UFO Board, vybavené krystalem o frekvenci 7,37MHz. Řídící počítač zasílá po sériové lince do mikrokontroléru seedy pro PRNG generátory, na základě kterých je provedeno vždy 368 podpisů. Tímto způsobem je omezena externí komunikace mikrokontroléru pro zvýšení efektivity měření. Spotřebu mikrokontroléru měříme pomocí osciloskopu Picoscope 6404D (osciloskop s dlouhou dobou záznamu), předzesilovače Langer PA 303 a BNC-SMA kabelu připojeného na konektor UFO Boardu. Výsledná sestava je vyfocena na obrázku 6.3. Nastavení osciloskopu volíme následující:

- rozsah kanálu: +2 V až -2 V,
- mód kanálu: DC 50 Ω,
- 25MHz omezovač pásma,
- vzorkovací frekvence: 39 MS/s.

Celý podpis trvá 14,75 ms. Z kapacitních důvodů (viz kapitolu 2.4) jsme změřili a vyhodnotili pouze implementace lineární transformace S, na kterou



Obrázek 6.3: Použitá sestava na měření



Obrázek 6.4: Naměřený záznam o spotřebě během počítání $\mathbf{S}^{-1}\mathbf{h}$

mj. cílí útok popsaný v kapitole 3.1. Výpočet lineárního zobrazení S trvá 267 us bez aditivního rozkladu matic, nebo 502 us s aditivním rozkladem matic. S ohledem na zvolenou vzorkovací frekvenci to znamená průběhy dlouhé 10500, resp. 19650 vzorků. Výsledný vzhled záznamu vykresleného do grafu je vidět na obrázku 6.4. Měření jsme provedli pomocí nástroje SICAK [23].

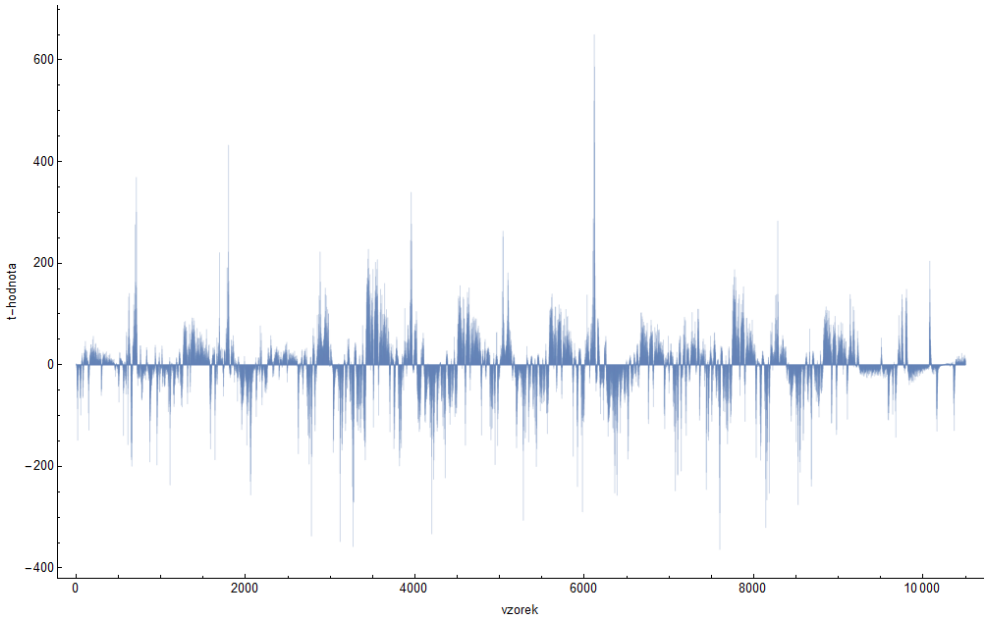
6.4 Vyhodnocení zabezpečení

Nyní nám zbývá vyhodnotit naměřené výsledky pomocí Welchova t-testu. Ten zatím nevyhodnocujeme, ale pouze si vypočteme hodnotu t, popisovanou v ka-

pitole 1.5.1. Pokud hodnota $|t|$ překročí hranici 4.5, zamítáme nulovou hypotézu a vzorky jsou rozlišitelné od sebe. Tedy informace ze zařízení uniká. Hodnoty t vykreslujeme do grafů, kde na horizontální ose máme jednotlivé vzorky (okamžiky v čase) a na svislé ose máme hodnotu t. Naměřili jsme čtyři různé varianty, které si postupně projdeme. Jedno vyhodnocení se provádí nad dvěma různými klíči a jelikož jsme měření prováděli nad čtyřmi klíči, máme ke každé variantě 6 grafů. Pro větší přehlednost a snížení redundance uvedu reprezentativní/důležité grafy (všechny grafy jsou přiloženy na CD).

6.4.1 Vyhodnocení nezabezpečené verze

Na obrázku 6.5 je vykreslená hodnota t v závislosti na číslu vzorku (resp. na času). Je zde zcela zřetelné, že hranice 4.5 je výrazně překračována a to opakovaně.



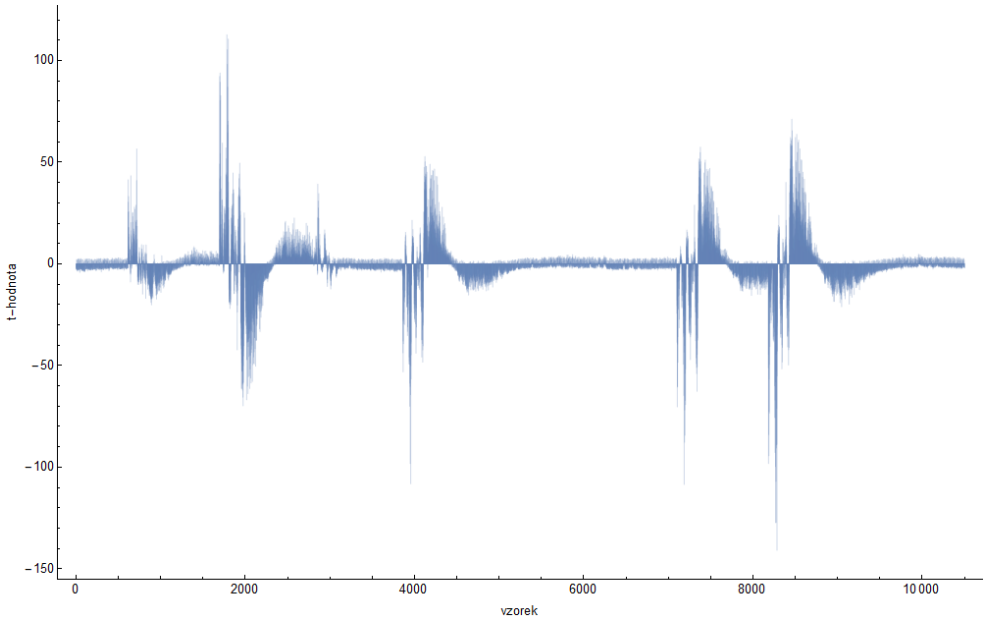
Obrázek 6.5: Nezabezpečená verze, 1. vs 2. klíč

Výsledkem tohoto testu je zamítnutí nulové hypotézy, tedy vzorky lze rozlišit od sebe (v našem případě testování umíme rozlišit, který klíč byl pro podepisování použit). Tím dostáváme jistotu, že ze zařízení uniká informace o klíči.

6.4.2 Vyhodnocení verze s ekvivalentním klíčem

Na obrázku 6.6 vidíme Welchův t-test, který porovnává druhý a třetí klíč. Je zde vidět, že k úniku dochází. U ekvivalentního klíče máme skutečně očekávaný únik informace, a to prostřednictvím nulových prvků (viz kapitola 4.1.3). Z al-

goritmu 4 víme, že maticové násobení se zpracovává po sloupcích, tedy únik informace je závislý na sloupci ve kterém se nachází nula. Pokud bychom měli větší podmatice $\mathbf{S}'$ než 8×8 prvků, pak by záleželo i ve které množině řádků z pohledu paralelizace se nachází. Naše sloupce se přesně vejdou do 32bitového slova, takže jsou zpracovány najednou.



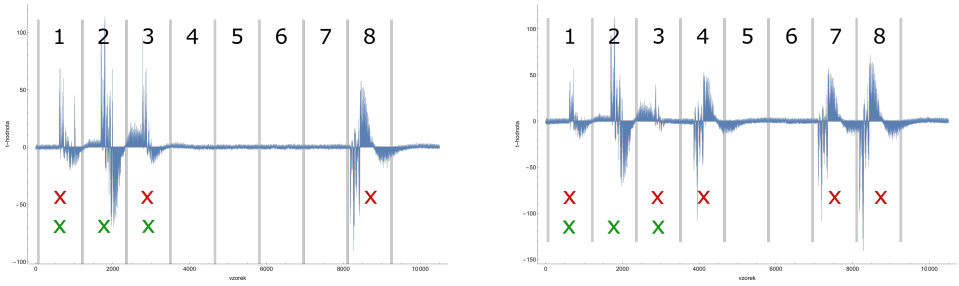
Obrázek 6.6: Verze s ekvivalentními klíči, 2. vs 3. klíč

Obrázek 6.7 popisuje únik informace pro ekvivalentní klíče, kde jsme doplnili dodatečné informace. Čísla v horní části obrázku rozdělují měření na 8 částí. Tyto části zhruba označují právě zpracovávaný sloupec vzhledem k měřeným vzorkům. Následně červené křížky (horní křížky) zvýrazňují, zda první klíč má v daném sloupci matice $\mathbf{S}'$ nulu. Zelené křížky (spodní křížky) zase zda druhý klíč, pro který je daný obrázek počítán, má v daném sloupci nulu. Vidíme, že k úniku informace dochází pouze, pokud jeden z klíčů obsahuje v daném sloupci nulový prvek. Pokud mají nulový prvek ve stejném sloupci a řádku, pak stále dochází k úniku (neplatí, že by se navzájem vyrušily).

Protiopatření realizované čistě pomocí multiplikativního maskování není odolné vůči útoku prvního řádu z důvodu nulových prvků. Je na zvážení, zda lze této informace využít, ale je vhodné používat tuto metodu v kombinaci s jinými metodami, které skryjí i nulové prvky.

6.4.3 Vyhodnocení verze s aditivní maskou

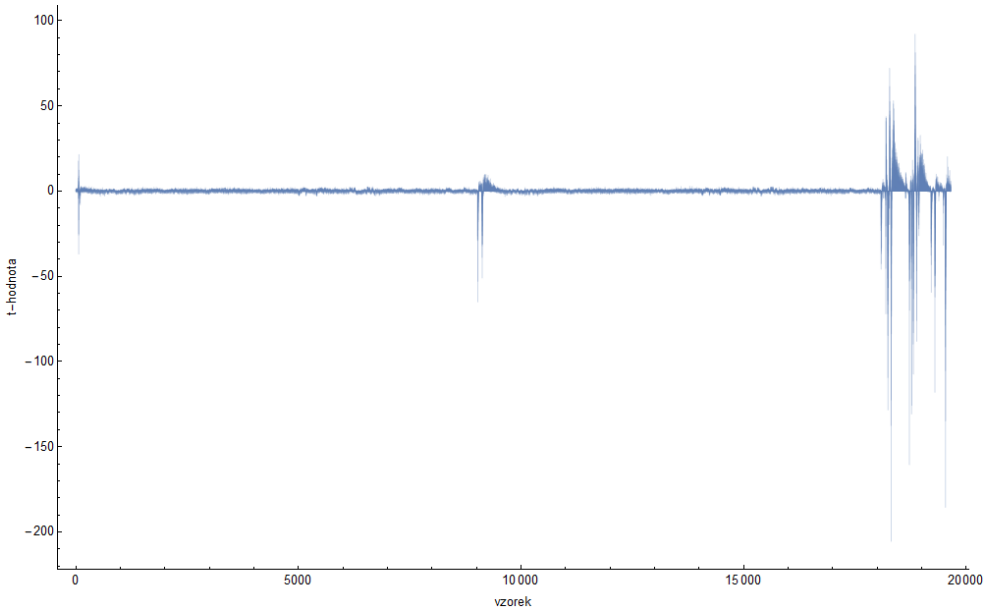
Zde máme již vytvořenou predikci. Citlivou hodnotu rozdělujeme na dvě sdílené hodnoty, kde ani jedna sdílená hodnota nenese informaci o původní hodnotě. Útok prvního řádu nemůže fungovat a Welchův t-test nemůže zamítnou



(a) 1. vs 3. klíč (b) 2. vs 3. klíč

Obrázek 6.7: Porovnání úniku informace s vyznačenými nulovými prvky vzhledem k indexu sloupce matice $\mathbf{S}'$

nulovou hypotézu. Podívejme se na obrázek 6.8, který naše očekávání zcela vyvrací.



Obrázek 6.8: Aditivní maska, 1. vs 2. klíč

Toto měření nedokazuje, že by aditivní maskování nebylo účinné, ale poukazuje na špatnou implementaci maskování. V době, kdy zaznamenáváme spotřebu (trigger je aktivní) nesmíme manipulovat s nechráněnými citlivými daty a v tomto testu nesmíme ani přistupovat k úložišti nechráněného klíče. Ve chvíli, kdy jeden klíč máme pevně na jedné adrese (v průběhu celého měření) a druhý klíč na jiné adrese, pak náš test nalezne tento přístup a na základě adresy, kde se klíč nalézá, dokáže odhalit rozdíl ve spotřebě. Tím dostáváme únik informace.

Toto odhalení přístupu jsme čekali a proto sdílené matice se vždy nacházejí na stejném místě. Podívejme se tedy na únik informace na konci grafu na obrázku 6.8. Na konci počítání lineárního zobrazení (dle algoritmu 2) se má přičíst vstupní vektor (ten se přičítá na základě jedniček na diagonále matice **S**). Z tohoto nemá jak informace unikat. Ovšem naše implementace má rozšířené soukromé klíče (viz. kapitola 4.1), kde na diagonále nutně nemusejí být jedničky. My zde nyní rozšířené matice nepoužíváme, proto vektory v soukromém klíči, které představují diagonálu, jsou nastaveny pevně na vektor jedniček. Ovšem tento přístup ke konstantním jedničkám na jiných adresách způsobuje únik informace.

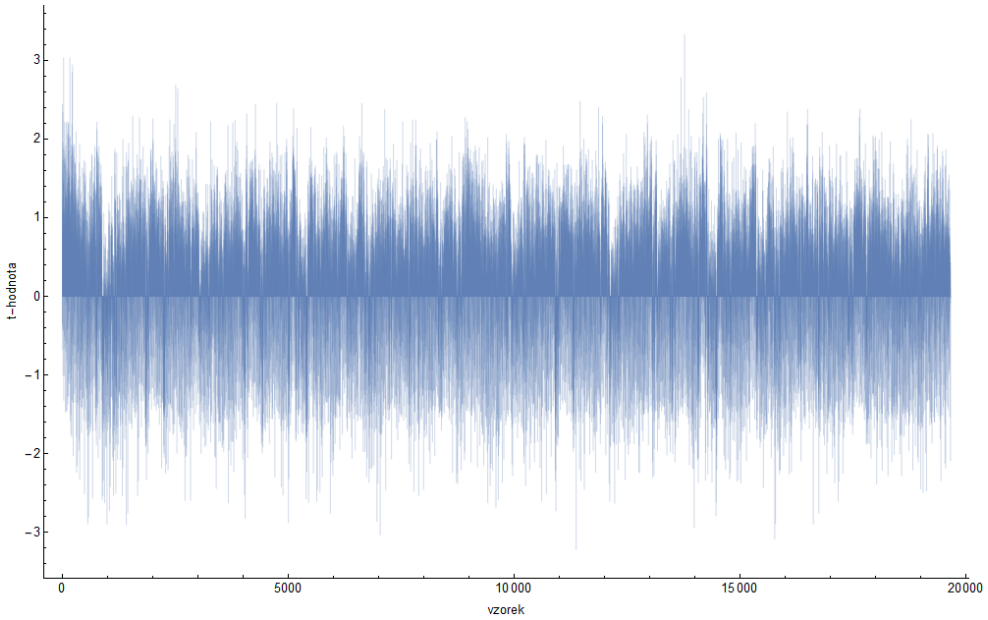
Zbývá nám analyzovat, kde se berou úniky na začátku měření a uprostřed. Na začátku se může jednat o hodnoty, které byly uloženy na zásobníku z předchozích funkcích, které byly závislé na klíči. Každopádně tomu tak nemůže být v prostřední části, kde se volá stejná operace (násobené matice vektorem). Znovu volání stejné funkce (jedná se o plně deterministické funkce s danými parametry) využije stejný prostor na zásobníku, takže touto cestou informace neuniká. Jelikož neznáme zdroj úniku, pak je vhodné opravit implementační chyby popsané výše a znovu data přeměřit.

6.4.4 Vyhodnocení kombinovaného zabezpečení

Na závěr provedeme měření, kde budeme používat současně ekvivalentní klíče a aditivní maskování nad lineárním zobrazením. V obrázku 6.9 vidíme výsledné hodnoty. Na grafu můžeme pozorovat, že hodnota t téměř nepřekročila hodnotu 3. Tím dostáváme výsledek, že nezamítáme nulovou hypotézu. Jinými slovy je zde možnost, že naměřená data mají stejnou střední hodnotu. V rámci našeho měření, síly předchozího zamítání, velikosti šumu a počtu záznamů lze prohlásit, že patrně informace (pro útok prvního řádu) neuniká.

Zde se zkombinovaly dvě vlastnosti, které si dopomohly k úspěšnému testu. Za prvé použití aditivní masky zabránilo, aby nám nulové prvky, skrz ekvivalentní klíč, poskytovaly informace. A za druhé použití ekvivalentního klíče vyžaduje, aby byl klíč přepočítán. Toto přepočítání způsobuje uložení klíče na pevně danou pozici pro všechny klíče. Tímto způsobem všechny soukromé klíče mají jednu adresu (v daném čase výpočtu) a na základě přístupu ke klíči, nelze zjistit, který klíč používáme. Tohle odstraňuje implementační chybu uvedenou v kapitole 6.4.3.

Tato implementace je s jistou pravděpodobností bezpečná vůči útoku řádu jedna. Toto vyhodnocení nám nedává žádnou informaci, zda útok druhého řádu lze realizovat. Jelikož jsme použili aditivní maskování prvního řádu, pak lze předpokládat, že útok druhého řádu bude úspěšný. Rovněž nevíme, zda by byl úspěšný útok prvního řádu s vyšším počtem záznamů o spotřebě.



Obrázek 6.9: Ekvivalentní klíč s aditivní maskou, 1. vs 2. klíč

Závěr

V této práci jsme navrhli a experimentálně ověřili útok postranním kanálem, a dále navrhli a experimentálně ověřili dvě různá protiopatření proti takovému útoku.

Útok na podpisové schéma Rainbow jsme navrhli pomocí odběrové korelační analýzy. Použili jsme 32bitovou referenční implementaci, která byla poskytnutá pro účely standardizace institutem NIST. Tuto implementaci jsme bez podstatných změn využili pro implementaci protiopatření a testování. Navržený útok je postavený na odhalení lineárních částí soukromého klíče. V práci popisujeme způsob, jakým lze po jejich odhalení dopočítat celý soukromý klíč. Během útoku využíváme paralelní zpracování uvnitř implementace, čímž dostaneme lepší výsledky.

Útok jsme úspěšně realizovali, čímž jsme získali celý soukromý klíč ze zařízení. Spotřebu pro korelační analýzu jsme měřili na mikrokontroléru ARM Cortex-M4. Ke kvantitativnímu zhodnocení útoku jsme vybrali metodologii míry úspěchu. Útok bez využití paralelizace potřebuje pro úspěšnost 95% přibližně 440 záznamů o spotřebě. Pro dosažení stejné úspěšnosti, při použití útoku využívajícího paralelismu, potřebujeme se znalostí jednoho, tří nebo sedmi dalších podklíčů 240, 150 nebo 70 záznamů o spotřebě.

V další části jsme navrhovali protiopatření proti útokům postranními kanály. Celkem jsme implementovali dvě protiopatření, kde první je aditivní maskování a druhé opatření je implementováno pomocí ekvivalentních klíčů. Ekvivalentní klíče jsou randomizované soukromé klíče, které částečně představují multiplikativní maskování.

K otestování protiopatření jsme vybrali metodu testování pomocí Welchova t-testu prvního řádu. Z kapacitních a výpočetních důvodů jsme měřili pouze lineární zobrazení, na které jsme cílili náš útok. Posuzovali jsme únik informací ze zařízení u čtyř variant implementace: bez protiopatření, s aditivní maskou, s ekvivalentními klíči a s kombinací obou protiopatření.

Dle očekávání, u nezabezpečené implementace jsme byli schopni detekovat únik informace během celé měřené operace. V případě aditivního maskování

jsme rovněž naměřili únik informací, který byl nicméně způsoben nevhodnou implementací. V případě implementace zabezpečené pomocí ekvivalentních klíčů jsme naměřili únik informace v místech, kde jsou nulové prvky v matici, což je způsobeno principem multiplikativního maskování. Na závěr jsme testovali kombinaci předešlých protiopatření, kde jsme nezaznamenali únik informace, tedy s daným šumem ji můžeme prohlásit za bezpečnou proti útoku prvního řádu. Na základě výsledků testování a porovnání implementační složitosti můžeme prohlásit aditivní maskování za efektivnější, protože má menší výpočetní nároky a nemá zranitelná místa vůči útoku prvního řádu.

Z práce vyplývá, že referenční implementace není bezpečná pro nasazení do reálného světa a je potřeba implementovat vhodná protiopatření proti útokům postranními kanály.

Literatura

[1] Fumaroli, G.; Martinelli, A.; et al. Affine Masking against Higher-Order Side Channel Analysis. In *Selected Areas in Cryptography*, Springer Berlin Heidelberg, 2011, pp. 262–280, doi:10.1007/978-3-642-19574-7_18.

[2] Shor, P. W. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM Journal on Computing*, volume 26, no. 5, Oct 1997: p. 1484–1509, ISSN 1095-7111, doi:10.1137/s0097539795293172. Available from: http://dx.doi.org/10.1137/S0097539795293172

[3] Ding, J.; Schmidt, D. Rainbow, a New Multivariable Polynomial Signature Scheme. In *Applied Cryptography and Network Security*, Springer Berlin Heidelberg, 2005, pp. 164–175, doi:10.1007/11496137_12.

[4] Kipnis, A.; Shamir, A. Cryptanalysis of the oil and vinegar signature scheme. In *Advances in Cryptology — CRYPTO '98*, Springer Berlin Heidelberg, 1998, pp. 257–266, doi:10.1007/bfb0055733.

[5] Kipnis, A.; Patarin, J.; et al. Unbalanced Oil and Vinegar Signature Schemes. In *Advances in Cryptology — EUROCRYPT '99*, Springer Berlin Heidelberg, 1999, pp. 206–222, doi:10.1007/3-540-48910-x_15.

[6] Kocher, P.; Jaffe, J.; et al. Differential Power Analysis. In *Advances in Cryptology — CRYPTO' 99*, Springer Berlin Heidelberg, 1999, pp. 388–397, doi:10.1007/3-540-48405-1_25.

[7] Brier, E.; Clavier, C.; et al. Correlation Power Analysis with a Leakage Model. In *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2004, pp. 16–29, doi:10.1007/978-3-540-28632-5_2.

[8] Standaert, F.-X.; Malkin, T. G.; et al. A unified framework for the analysis of side-channel key recovery attacks. In *Annual international confe-*

rence on the theory and applications of cryptographic techniques, Springer, 2009, pp. 443–461.

[9] Nikova, S.; Rechberger, C.; et al. Threshold Implementations Against Side-Channel Attacks and Glitches. In *Information and Communications Security*, Springer Berlin Heidelberg, 2006, pp. 529–545, doi: 10.1007/11935308_38.

[10] Schneider, T.; Moradi, A. Leakage Assessment Methodology. In *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2015, pp. 495–513, doi:10.1007/978-3-662-48324-4_25.

[11] Garey, M. R.; Johnson, D. S.; et al. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W H FREEMAN WORTH PUB 3PL, Apr. 2011, ISBN 0716710455. Available from: https://www.ebook.de/de/product/3637119/m_r_garey_david_s_johnson_michael_r_garey_computers_and_intractability_a_guide_to_the_theory_of_np_completeness.html

[12] Sakumoto, K.; Shirai, T.; et al. On Provable Security of UOV and HFE Signature Schemes against Chosen-Message Attack. In *Post-Quantum Cryptography*, Springer Berlin Heidelberg, 2011, pp. 68–82, doi:10.1007/978-3-642-25405-5_5.

[13] Standaert, F.-X. How (Not) to Use Welch's T-Test in Side-Channel Security Evaluations. In *Smart Card Research and Advanced Applications*, Springer International Publishing, 2019, pp. 65–79, doi:10.1007/978-3-030-15462-2_5.

[14] Park, A.; Shim, K.-A.; et al. Side-Channel Attacks on Post-Quantum Signature Schemes based on Multivariate Quadratic Equations. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, aug 2018: pp. 500–523, doi:10.46586/tches.v2018.i3.500-523.

[15] National Institute of Standards and Technology. *Submission Requirements and Evaluation Criteriafor the Post-Quantum Cryptography Standardization Process*. Dec. 2016. Available from: https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf

[16] Lu, Y.; O'Neill, M.; et al. Evaluation of Random Delay Insertion against DPA on FPGAs. *ACM Transactions on Reconfigurable Technology and Systems*, volume 4, no. 1, dec 2010: pp. 1–20, doi:10.1145/1857927.1857938.

[17] Ratanpal, G.; Williams, R.; et al. An on-chip signal suppression countermeasure to power analysis attacks. *IEEE Transactions on Dependable and Secure Computing*, volume 1, no. 3, jul 2004: pp. 179–189, doi: 10.1109/tdsc.2004.25.

[18] Pokorný, D.; Socha, P.; et al. Side-channel attack on Rainbow post-quantum signature. *Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE.*, 2021.

[19] Moradi, A.; Richter, B.; et al. Leakage Detection with the x2-Test. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, feb 2018: pp. 209–237, doi:10.46586/tches.v2018.i1.209-237.

[20] Moos, T.; Wegener, F.; et al. DL-LA: Deep Learning Leakage Assessment: A modern roadmap for SCA evaluations. Cryptology ePrint Archive, Report 2019/505, 2019, https://eprint.iacr.org/2019/505.

[21] O'Flynn, C.; Chen, Z. Synchronous sampling and clock recovery of internal oscillators for side channel analysis and fault injection. *Journal of Cryptographic Engineering*, volume 5, no. 1, nov 2014: pp. 53–69, doi: 10.1007/s13389-014-0087-5.

[22] PQCRainbow. Online. Available from: https://www.pqcrainbow.org/

[23] Socha, P.; Miškovský, V.; et al. SICAK: An open-source SIde-Channel Analysis toolKit. *8th Workshop on Trustworthy Manufacturing and Utilization of Secure Devices (TRUDEVICE)*, May 2019.

Příloha A

Zkratky

AES Advanced encryption standard

CMOS Complementary metal–oxide–semiconductor

CPA Correlation power analysis

EUF-CMA Existential unforgeability under chosen message attack

FDH Full domain hash scheme

GF Galois field

HFE Hidden fields equations

LA Leakage assessment

MQ Multivariate quadratic

MS/s Mega samples per second

NIST National institute of standards and technology

OV Oil and vinegar

PK Public key

PRNG Pseudorandom number generator

RISC Reduced instruction set computer

RSA Rivest, Shamir, Adleman

SCA Side-channel attack

SK Secret key

SPA Simple power analysis

TVLA Test vector leakage assessment

Příloha B

Obsah přiloženého CD

- `readme.md` ..... popis obsahu CD
- `thesis.pdf` ..... diplomová práce v .pdf formátu
- `data` ..... naměřená data během procesu LA
- `grafs` ..... grafy s LA, které byly přislíbeny v kapitole 6.4
- `projects` ..... souhrnná složka s projekty
  - `readme.md` ..... popis projektů
  - `implementation-attack` ..... zdrojový kód - nezabezpečená verze pro jupyter-attack
  - `implementation-meas` ..... zdrojový kód - impl. protiopatření pro jupyter-meas
  - `jupyter-attack` ..... jupyter soubory pro útok
  - `jupyter-meas` ..... jupyter soubory - gen. klíče a testování
- `thesis` ..... složka se zdrojovým LaTeX kódem této diplomové práce