MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY

Odběrová analýza čipových karet s využitím osciloskopu PicoScope

BAKALÁŘSKÁ PRÁCE

Ondřej Koutský

Brno, jaro 2012

Prohlášení

Prohlašuji, že tato bakalářská práce je mým původním autorským dílem, které jsem vypracoval samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování používal nebo z nich čerpal, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

Ondřej Koutský

Vedoucí práce: RNDr. Petr Švenda, PhD.

Poděkování

Rád bych poděkoval panu RNDr. Petru Švendovi, PhD. za poskytování cenných připomínek, rad a nápadů při vedení této bakalářské práce.

Shrnutí

Odběrová analýza je dnes považována za jeden z nejúčinnějších útoků proti čipovým kartám. Využívá se při ní skutečnosti, že spotřeba energie zařízení je závislá na výpočetní složitosti právě vykonávané instrukce a na datech, která tato instrukce zpracovává. Množství úspěšně realizovaných útoků vedlo výrobce čipových karet k nutnosti zavést do svých zařízení množství obranných mechanizmů. Tato práce se zabývá odběrovou analýzou několika čipových karet, s jejíž pomocí je analyzováno, jakým způsobem se tyto mechanizmy chovají a jak jsou implementovány.

Klíčová slova

odběrová analýza, čipová karta, PicoScope, JavaCard, RSA, maskovací algoritmy

Obsah

Kapitola 1

Úvod

Čipové karty se v dnešní době staly nedílnou součástí každodenního života většiny z nás. Oblast jejich využití je široká, sahá od bankovních platebních karet, přes GSM SIM karty v mobilních telefonech, až po bezkontaktní karty používané jako oprávnění pro vstup do budov nebo ve veřejné dopravě.

Díky zabudovanému mikroprocesorovém čipu je karta schopna přechovávat a také zpracovávat nejrůznější data, jejichž utajení a zabezpečení je často stěžejním problémem. Vážný bezpečnostní problém pro čipové karty představují útoky využívající informací unikajících z tzv. postranních kanálů. Nejvýznamnějšími postranními kanály umožňujícími tento druh útoku jsou elektromagnetické vlnění, chování karty po úmyslném zavedení chyby a především pak odběr energie. Právě analyzováním dat získaných při měření odběru energie se zabývá odběrová analýza, která je náplní i této práce.

Cílem bakalářské práce bylo realizovat odběrovou analýzu s jednoduchým osciloskopem PicoScope. Měření se uskutečnilo s využitím API dodávaného výrobcem, pomocí kterého byl osciloskop ovládán. Odběrová analýza byla provedena na několika kartách s technologií Java Card při průběhu šifrování algoritmem RSA.

Tato práce je rozdělena do šesti kapitol. První tři jsou věnovány stručnému popisu principů a vlastností čipových karet, blíže objasněny metody odběrové analýzy a představeny obecné mechanizmy ochrany proti tomuto druhu útoku. Následující kapitola se zabývá algoritmem RSA a jeho možným chráněným implementacím v čipových kartách. Další kapitola seznamuje s ovládáním osciloskopu PicoScope pomocí dodaného API a popisuje základní prvky vytvořeného ovládacího programu. Poslední část práce je věnována analýze výsledků provedených experimentů.

Kapitola 2

Čipové karty

Pojmem čipová karta [1] rozumíme kartové zařízení kapesní velikosti se zabudovaným mikroprocesorovým čipem, díky němuž je karta schopna provádět kryptografické operace, autentizovat se, ukládat a zpracovávat citlivá data.

V této kapitole bude postupně vysvětleno rozdělení čipových karet, objasněna jejich architektura a stručně popsána technologie JavaCard, jejíž principů je využíváno v dalších částech práce. Poslední podkapitola je věnována seznamu karet použitých později při měření v praktické části.

2.1 Rozdělení čipových karet

Čipové karty mohou být rozděleny do dvou základních skupin na karty paměťové, které mohou data pouze uchovávat, a karty mikroprocesorové, schopné data dále zpracovávat [1]. Z hlediska komunikačního rozhraní jsou karty dále rozděleny na kontaktní a bezkontaktní.

Paměťové karty
Paměťové karty poskytují jen omezenou funkcionalitu. Jejich bezpečnostní mechanizmy sice dokáží zabránit neoprávněné manipulaci s uloženými daty, karta už však není schopna provádět jakékoliv kryptografické operace[1]. Nevýhodou běžných paměťových karet je i jejich snadné zkopírování. Výhodou naopak zůstává nízká cena a snadná dostupnost.

Mikroprocesorové karty
Mikroprocesorové karty jsou aktivní karty se zabudovaným mikroprocesorovým čipem a instalovaným operačním systémem. Jejich hlavní přínos spočívá ve schopnosti bezpečně ukládat tajná data a provádět potřebné

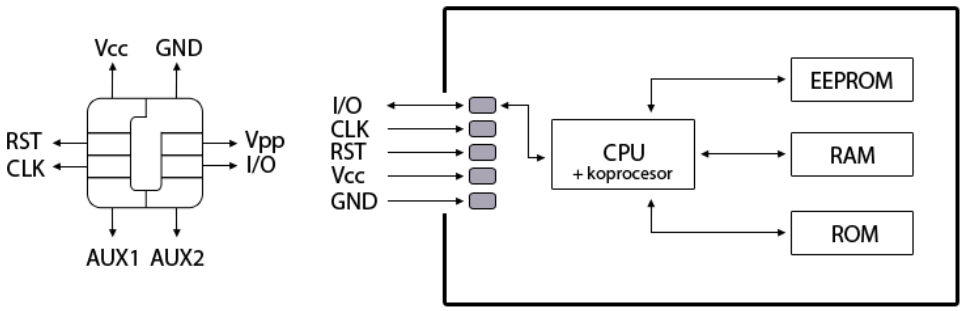
1. Podle [1] ve skutečnosti existují paměťové čipy obsahující komplexnější bezpečnostní logiku umožňující například jednoduché šifrování. Možnosti takovýchto mechanizmů jsou však velmi omezené.

kryptografické operace. Moderní karty jsou navíc vybaveny kryptografickým koprocesorem nebo například kvalitním hardwarovým generátorem náhodných čísel.

Bezkontaktní karty
Bezkontaktní čipové karty jsou zařízení schopná přenášet svá data bez nutnosti elektrického kontaktu se čtecím zařízením. V současnosti jsou jako bezkontaktní využívány jak paměťové, tak mikroprocesorové karty. Paměťové bezkontaktní karty jsou obvykle schopné komunikovat na vzdálenost přibližně jednoho metru, ty mikroprocesorové na délku do několika centimetrů [1].

2.2 Architektura čipových karet

Architekturu čipové karty a její technické vlastnosti specifikuje norma ISO 7816. Obrázek 2.1 ukazuje zjednodušené schéma architektury čipové karty a jejího rozhraní tak, jak jej tato norma popisuje. Podrobnějšímu rozboru schématu jsou věnovány následující podkapitoly.



Obrázek 2.1: Architektura čipové karty a schéma komunikačního rozhraní

2.2.1 Komponenty

Stěžejními komponentami čipové karty jsou CPU (Central Processing Unit) a tři druhy pamětí – RAM (Random Access Memory), ROM (Read Only Memory) a EEPROM (Electrically Erasable Programmable Read Only Memory):

- **CPU** – U dnešních čipových karet je jako CPU použit nejběžněji 8bitový mikroprocesor často podporovaný kryptografickým koproceso-

rem urychlujícím náročné kryptografické operace. Běžné jsou v současnosti také karty využívající 16bitové a 32bitové mikroprocesory.

- **RAM** – Operační paměť, která slouží zejména k ukládání mezivýpočtů procesoru. Její velikost se obvykle pohybuje v řádech kilobytů.
- **ROM** – Jedná se o permanentní paměť, do které je při výrobě čipu nahrán operační systém. Její velikost je řádově 10–100 kB.
- **EEPROM** – Tato paměť slouží v čipové kartě k ukládání většiny programů a aplikací. Je přepisovatelná a energeticky nezávislá s velikostí v řádech desítek kilobytů.

2.2.2 Komunikační rozhraní

Jak ukazuje obrázek 2.1, komunikační rozhraní čipových karet se skládá z osmi kontaktů:

- **Vcc** a **GND** – Zajišťují napájení karty elektrickým proudem a uzemnění (GND – Ground).
- **Vpp** – Kontakt Vpp v minulosti sloužil pro zásobování karty energií používanou pro programování paměti EEPROM. V současnosti si většina karet Vpp vyrábí sama a kontakt tak zůstává nevyužitý.
- **CLK** – Slouží jako vstup hodinového signálu z terminálu. Podle tohoto signálu je následně stanovena rychlost komunikace na I/O konektoru.
- **I/O** – Vstupně-výstupní konektor pro asynchronní sériovou komunikaci mezi kartou a čtečkou.
- **RST** – Pomocí konektoru RESET je k mikroprocesoru vysílán signál sloužící k zahájení resetovací sekvence instrukcí.
- **AUX1** a **AUX2** – Tyto konektory jsou prozatím nevyužity a připraveny pro budoucí použití.

2.3 JavaCard

JavaCard [7] je technologie – programovací platforma – umožňující čipovým kartám a podobným zařízením disponujícím omezenou výpočetní kapacitou bezpečný běh malých aplikací zvaných applety.

Platforma JavaCard je založená na redukované[2] formě jazyka a technologie Java. Díky tomu jsou JavaCard applety nezávislé na použité platformě a kompatibilní mezi kartami různých výrobců. Kompilace appletu je prováděna standardním Java kompilátorem, poté je převeden pomocí JavaCard konvertoru a následovně nahrán a instalován na kartu. Spuštění appletu je řízeno terminálem zavoláním spouštěcí metody *Select()*. Běh celého systému je spravován pomocí JavaCard virtual machine.

2. Podporovány jsou pouze základní primitivní datové typy boolean, byte a short, chybí například podpora vláken. Kompletní přehled podporovaných prvků je popsán v [3].

Kapitola 3

Odběrová analýza

Vážný problém pro bezpečnost čipových karet představují útoky postranním kanálem. Při těch se útočník na rozdíl od klasické kryptoanalýzy nesnaží objevit chyby v matematické struktuře kryptografických algoritmů, ale využívá informace, které unikají ze samotné fyzické implementace systému při běhu těchto algoritmů [4]. Nejvýznamnějšími útoky postranním kanálem jsou časová [6], chybová [8] a odběrová analýza.

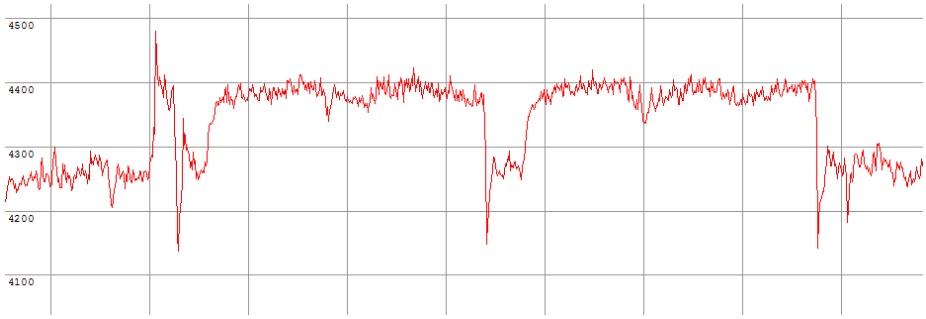
Při odběrové analýze se využívá skutečnosti, že spotřeba energie zařízení je závislá na výpočetní složitosti právě vykonávané instrukce a na datech, která tato instrukce zpracovává. Měřením spotřeby tak útočník může získat informace nejen o operacích, které v zařízení probíhají, ale mohou vést až k odhalení tajného klíče, se kterým se provádí kryptografické algoritmy [4]. Útok odběrovou analýzou je účinný zejména proti nízkoenergetickým zařízením, které nemají vlastní baterii a jsou tedy napájeny přímo ze čtecího zařízení. Jejich spotřeba je tak snadno měřitelná. Takovým zařízením jsou i čipové karty [1].

Podle způsobu zpracování naměřených dat rozdělujeme odběrovou analýzu na dva základní typy – Jednoduchou odběrovou analýzu (Simple Power Analysis – SPA) a Diferenciální odběrovou analýzu (Differential Power Analysis – DPA).

3.1 Jednoduchá odběrová analýza (SPA)

SPA je technika útoku založená na přímém vyhodnocování dat získaných z měření odběru energie [2]. Hodnoty naměřených dat a tím i tvar výstupní stopy jsou výrazně závislé na instrukcích, které jsou během odběru na zařízení prováděny, a na datech, se kterými tyto instrukce manipulují. Použitím SPA je tak útočník schopen identifikovat sekvence těchto instrukcí a tím například určit typ a implementaci použitého šifrovacího algoritmu. Za určitých podmínek může dojít i k odhalení informací o použitém tajném klíči, například Hammingově váze některých jeho částí nebo dokonce jeho jed-

notlivých bitů. Příklad odběrové stopy, která může být výstupem SPA, je obrázek 3.1. Stopa odpovídá průběhu šifrování algoritmem DES na čipové kartě.



Obrázek 3.1: SPA průběhu šifrování algoritmem DES. Zdroj: vlastní měření.

3.2 Diferenciální odběrová analýza

DPA představuje mnohem silnější nástroj než SPA. Na rozdíl od SPA není založena na přímém vyhodnocování množství spotřebované energie, ale využívá k útoku statistické metody. Díky tomu je odstraněn šum vzniklý při měření a mohou být zřetelné i velmi malé odchylky ve spotřebě objevující se v závislosti na zpracovávaných datech. K úspěšnému útoku navíc není nutná znalost přesné implementace šifrovacích algoritmů [9].

K útoku pomocí DPA je zapotřebí velké množství (v řádech tisíců) měření se stejným šifrovacím klíčem a různými datovými vstupy. Útočník odhaluje tajný klíč po jednotlivých bytech. Pro každý byte vyzkouší všech 256 možností a pro každou možnost provede velké množství měření. Naměřené stopy jsou následně podle určité rozřazovací funkce (např. podle Hammingovy váhy výsledku) rozděleny do dvou skupin a hodnoty dat jsou v obou skupinách zprůměrovány. Průměrné hodnoty z obou skupin se odečtou. Pokud byl útočníkův odhad bytu klíče špatný, je výsledná diferenční stopa téměř rovná, pokud správný, objeví se v ní výrazné vrcholy, které značí závislost na klíči [10].

Kapitola 4

Ochrana proti odběrové analýze

Útok odběrovou analýzou je založen na skutečnosti, že odběr energie kryptografického zařízení je závislý na prováděných operacích a na datech, se kterými tyto operace pracují. Cílem každého protiopatření je tedy tuto závislost přerušit nebo alespoň snížit na minimum. Představeno bylo už mnoho možných řešení. Podle [12] mohou být opatření rozdělena na ta, co závislosti *skrývají* a ta, co je ovlivňují *maskováním* tajných dat.

4.1 Skrývání závislostí (Hiding)

Cílem skrývacích technik je zastavit přímé ovlivňování spotřeby energie hodnotami zpracovávaných dat a prováděnými operacemi. Toho lze dosáhnout, spotřebovává-li zařízení v každém hodinovém cyklu buď *konstantní*, nebo *náhodné* množství energie. Je zřejmé, že úplné dosažení takových podmínek je v reálném prostředí nemožné, existují však návrhy řešení, které se jejich splnění blíží. V zásadě je můžeme rozdělit do dvou skupin na ty, které zasahují *časové charakteristiky* spotřeby energie a na ty, které ovlivňují *amplitudové charakteristiky*.

4.1.1 Časové charakteristiky

Jednou z podmínek úspěšného útoku diferenciální odběrovou analýzou je předpoklad, že se spotřeba energie každé operace promítne do výsledné stopy pokaždé na stejné pozici. Pokud tato podmínka není splněna, vyžaduje DPA mnohonásobně vyšší počet měření a výrazně je tak útočníkovi znesnadněna. Protiopatření zaměřená proti takovému principu jsou obvykle založena na snaze co nejvíce *randomizovat* výpočet kryptografických operací. K randomizaci bývají nejčastěji použity následující dvě techniky:

1. **Vkládání „falešných“ operací** – Do průběhu výpočtu kryptografických operací jsou náhodně vkládány „falešné“ operace, které nemají

jiný účel, než prodloužit výpočet o útočníkovi neznámou délku. Při použití opatření tohoto druhu je důležité zajistit, aby během každého běhu algoritmu bylo použito stejné množství „falešných“ operací, a aby byly tyto operace od ostatních útočníkem nerozlišitelné.

2. **Náhodné zpřeházení pořadí operací** – Sekvence operací kryptografických výpočtů, které mohou být provedeny v libovolné posloupnosti, proběhnou v náhodném pořadí. Množství sekvencí, jejichž pořadí může být zaměněno, se liší podle druhu použitého algoritmu.

4.1.2 Amplitudové charakteristiky

Opatření zasahující časové charakteristiky jsou založena na co největší randomizaci běhu kryptografických výpočtů. Opatření týkající se amplitudových charakteristik se na rozdíl od nich zabývají spíše způsoby, jak ovlivnit odběrové vlastnosti těchto operací přímo.

Jedním z možných řešení spadajících do této kategorie je vhodná volba programových instrukcí při softwarové implementaci kryptografických algoritmů. Různé instrukce spotřebovávají různé množství energie. Jejich výběr by tedy měl být omezen jen na ty, při jejichž vykonávání uniká pouze minimální množství informací.

4.2 Maskování tajných dat (Masking)

Základní myšlenkou maskování je překrýt zpracovávanou hodnotu x (typicky plaintext nebo klíč) náhodnou hodnotou m zvanou maska. Odběrová stopa zařízení pak není ovlivněna skutečnými tajnými daty, ale pouze jejich maskovaným obrazem. Maska je generována uvnitř kryptografického zařízení a její hodnota se s každým novým výpočtem mění. Podle způsobu, jakým jsou data v x maskována dělíme maskování na *booleovské* a *aritmetické*.

V případě booleovského maskování je hodnota x překryta maskou m pomocí logické operace XOR: $x_m = x \oplus m$. Aritmetické maskování je naproti tomu nejčastěji založeno na aritmetické operaci modulárního sčítání: $x_m = x + m \ (mod\ n)$ nebo modulárního násobení: $x_m = x \times m \ (mod\ n)$. V obou případech je modulo n vždy voleno v závislosti na vlastnostech použitého algoritmu.

Kapitola 5

Algoritmus RSA a jeho implementace

Kryptosystém RSA je první veřejně publikovaný šifrovací algoritmus založený na asymetrické kryptografii. Představen byl už v roce 1978 v článku [11] autory Rivestem, Shamirem a Adlemanem, využíván je však i v současnosti a to jak pro šifrování komunikace, tak pro digitální podepisování dat. Bezpečnost RSA je založena na předpokladu obtížnosti rozkladu velkých čísel na prvočinitele.

Tato kapitola se ve své první části věnuje bližšímu popisu algoritmu RSA, ve druhé pak jeho možným implementacím chráněným proti útoku odběrovou analýzou.

5.1 Princip RSA

Ustanovení klíčů. Průběh algoritmu RSA začíná ustanovením *soukromého* a *veřejného* šifrovacího klíče. Jedna z komunikujících stran provede následující kroky:

1. Určí dvě náhodná prvočísla p a q.
2. Vypočte $n = pq$ a hodnotu Eulerovy funkce $\phi(n) = (p-1)(q-1)$.
3. Zvolí celé číslo e takové, že $1 \leq e \leq \phi(n)$ a $nsd(e, \phi(n)) = 1$.
4. Nalezne kladné číslo d takové, že $ed \equiv 1(mod\ \phi(n))$

Veřejným klíčem jsou ustanoveny čísla n a e nazývána modulo a veřejný šifrovací exponent. Soukromým klíčem je číslo d, které nazýváme soukromý dešifrovací exponent a číslo $\phi(n)$.

Šifrování zprávy. Strana A chce straně B poslat zprávu M. Potom musí A provést tyto kroky:

1. Zjistit veřejný klíč B, tedy dvojici (n, e).

2. Převést zprávu M na celé číslo m z intervalu $[1, (n-1)]$.
3. Vypočítat $c = m^e (mod\ n)$.
4. Zaslat šifrovanou zprávu c straně B.

Dešifrování zprávy. Strana B obdržela od A šifrovanou zprávu c a chce získat původní zprávu M. Postup dešifrování je následující:

1. S použitím soukromého klíče d vypočítat zprávu $m = c^d (mod\ n)$.
2. Převést zprávu m zpět do srozumitelné podoby na původní M.

5.2 Chráněné implementace RSA

Ústřední operací algoritmu RSA je *modulární umocňování*. Jeho výpočet je ve většině čipových karet řešený relativně efektivními umocňovacími algoritmy založenými na metodě *„square-and-multiply"*. Obecný pseudokód této metody je následující:

Algorithm *Square-and-multiply*
Input: g, $e = (e_t, e_{t-1}...e_1, e_0)_2$
Output: g^e
1. $R_0 \leftarrow 1$, $R_1 \leftarrow g$;
2. **for** $i \leftarrow t-1$ **downto** 0 **do**
3. $\quad R_0 \leftarrow R_0^2$;
4. $\quad$ **if** $e_i = 1$ **then** $R_0 \leftarrow R_0 R_1$;
5. **return** A.

Z řádku 4 vyplývá, že provedení podmíněné větve přímo závisí na hodnotě exponentu, kterým je v případě RSA veřejný nebo tajný klíč. Takováto implementace algoritmu je snadno napadnutelná jednoduchou odběrovou analýzou. Existuje sice upravená verze algoritmu, ve které je čas výpočtu konstantní, ani ta však nezaručí odolnost vůči útoku pomocí DPA [20].

Většina dosavadně publikovaných metod ochrany algoritmů asymetrické kryptografie je založena na randomizaci výpočtů a dat pomocí multiplikativního maskování [13]. V zásadě známe dvě možnosti, jak této randomizace dosáhnout:

1. **Maskováním vstupních dat**. Základní myšlenkou opatření z této kategorie je randomizovat vstupní data, obvykle klíč i plaintext, ještě před samotným průběhem šifrovacího algoritmu. Do této kategorie řadíme metodu *exponent blinding* a částečně metodu *trojitého maskování*.

2. **Randomizací algoritmu pro modulární umocňování**. Právě algoritmy pro výpočet modulárního umocňování jsou často nejnáchylnějším místem pro útoky postranním kanálem. Vhodnou implementací s využitím náhodného maskování lze však množství unikajících informací minimalizovat. Bližší pozornost bude věnována randomizované verzi algoritmu *Montgomery ladder* a třem maskovaným implementacím window metody (dále WM): *Overlapping WM*, *Randomized Table WM* a *Hybrid Randomizing WM*.

5.2.1 Zaslepení exponentu (Exponent blinding)

Exponent blinding je technika maskování navržená a popsaná P. Kocherem v [6]. Před samotným modulárním umocňováním zprávy se spočítá dvojice čísel (v_i, v_f) taková, že v_f je náhodné a $v_i = (v_f^{-1})^e \ mod \ n$, kde e je veřejný exponent. Zpráva m se zamaskuje vynásobením s v_i, provede se modulární umocňování pro zašifrování a poté se šifrovaná zpráva překryje číslem v_f.

Pro další zvýšení randomizace výpočtů může být maskován i samotný veřejný exponent (resp. v případě dešifrování soukromý exponent) a to přičtením násobku $\phi(n)$. Výše popsaný postup lze shrnout v následujících krocích:

1. Zamaskování zprávy m: $m' = (v_i \times m) \ mod \ n$.
2. Zamaskování veřejného exponentu e: $e' = e + r\phi(n)$, kde r je náhodné.
3. Šifrování pomocí modulárního umocňování: $c' = m'^{e'}$.
4. Odmaskování výsledku: $c = (v_f \times c') \ mod \ n$.

Z hlediska ochrany proti odběrové analýze je důležité, aby maskovací dvojice (v_i, v_f) byla pro každé šifrování unikátní. Výpočet inverzí modulo n je však složitá a pomalá operace a generování nového náhodného maskovacího páru pro každé šifrování by bylo velmi neefektivní. Možným řešením tohoto problému je vypočítat novou dvojici z páru předchozího, například $v_i' = v_i^2$ a $v_f' = v_f^2$.

5.2.2 Trojité maskování (Triple masking)

Metoda *trojitého maskování* popsaná v [16] navazuje jak na principy techniky *exponent blinding*, tak na myšlenku randomizace exponenciálního výpočtu. Na rozdíl od exponent blinding jsou zde kromě plaintextu a exponentu

maskovány i samotné instrukce algoritmu během modulárního umocňování. Hlavním cílem trojitého maskování je vkládáním náhodných instrukcí do výpočtu algoritmu square-and-multiply smazat rozdíly ve spotřebě mezi operacemi umocňování na druhou a násobením. Náhodné instrukce jsou vkládány na základě bitových hodnot náhodného maskovacího vektoru $N = (N_{K-1}, N_{K-2} \dots N_0)$. Pseudokód algoritmu trojitého maskování je popsán v příloze B.

5.2.3 Metoda překrývání oken (O–WM: Window overlapping method)

Metoda *Overlapping Window* je první ze tří technik představených K. Itohem et al. v [14]. Základním algoritmem, ze kterého všechny tři maskované implementace vychází, je tzv. sliding window metoda (WM). WM je efektivní technika používaná pro modulární umocňování při výpočtech v asymetrické kryptografii. Stručně je popsána níže, podrobněji například v [15].

Metoda oken (Window method). Jedná se o efektivní řešení speciálního případu tzv. m-ární metody (zobecněné square-and-multiply) umocňování [15], kde základ $m = 2^k$. Algoritmus nahlíží na mocnitel e jako na binární číslo, kterým je základ mocniny b postupně umocňován po k-bitových částech zvaných okna. Komentovaný pseudokód obecného algoritmu je obsažen v příloze B.

Jak už bylo řečeno, na technice WM staví i metoda *Overlapping Window*. *O–WM* randomizuje výpočet tím, že dvě po sobě následující okna w_i a w_{i+1} náhodně překrývá[1]. Fixní exponent je tedy pokaždé zpracováván jiným způsobem, vždy v náhodném množství oken. Počet oken, jejich bitové pozice a tím i hodnoty vnitřních dat se při každém výpočtu liší a jsou tak pro útočníka nepředvídatelné.

Výpočet začíná zvolením náhodné hodnoty q určující počet oken, v nichž bude exponent zpracováván. Dále vybereme náhodná čísla h_i, která stanoví délku překrytí mezi okny w_i a w_{i+1}. Ta by měla splňovat podmínky $0 < h_0, h_1 \dots h_{q-2} < q$ a $q \times k + h_0 + h_1 + \cdots + h_{q-2} = u$, kde k značí délku okna a u bitovou délku exponentu. Z hlediska ochrany proti SPA je doporučeno stanovit $h \geq k/2$. Ochranu před DPA zajistí právě randomizace počtu oken a jejich náhodné překrývání. Komentovaný pseudokód je obsažen v příloze B.

1. Překlad z anglického *„overlap“* – překrývat

5.2.4 Metoda oken s randomizovanou tabulkou (RT–WM: Randomized Table Window Method)

Základní algoritmus WM využívá k urychlení procesu umocňování g^e tabulku předpočítaných základních mocnin g. Technika *RT–WM* je založena na randomizaci právě těchto tabulkových hodnot. Místo předpočítaných lichých mocnin g^i ($g^3 \dots g^{2^k-1}$) vypočítáme randomizované hodnoty: $g^{i \times 2^b + r}$, kde i je číslo indexu a r je b-bitové náhodné číslo. Pro získání konečného výsledku $g^e \mod n$ je nutné výstupní data zpět „normalizovat“. Detailní postup algoritmu je i s komentáři obsažen v příloze B.

5.2.5 Hybridně randomizující metoda oken (HR–WM: Hybrid Randomizing Window)

HR–WM je založena na kombinaci obou předešlých metod. Předvýpočet tabulkových hodnot probíhá randomizovaně podle *RT–WM*. Překrývání oken je převzato z *O–WM*, rozdíl je pouze v hloubce překrytí. To je u HR–WM vždy konstantní délky h.

5.2.6 Randomizovaný algoritmus Montgomery ladder

Tato podkapitola bude věnována randomizované variantě další efektivní techniky modulárního umocňování tzv. *montgomery ladder* (ML). Metoda ML byla poprvé publikována v [18] a upravena v [17]. Pseudokód základního algoritmu je následující:

Algorithm *Montgomery ladder*
Input: g, $e = (e_{t-1}, e_t ... e_1, e_0)_2$
Output: g^e
1. $R_0 \leftarrow 1$, $R_1 \leftarrow g$;
2. **for** $i \leftarrow t-1$ **downto** 0 **do**
3. $\quad R_{\neg e_i} \leftarrow R_0 R_1$;
4. $\quad R_{e_i} \leftarrow (R_{e_i})^2$;
5. **return** R_0.

Umocňování pomocí ML je už v této základní podobě odolné proti útoku SPA. Přímým rozborem odběrové stopy lze jen velmi obtížně rozeznat hodnoty použitého exponentu, útoku DPA však daná implementace nezabrání. Řešení nabízí randomizovaná verze navržená G. Fumarolim et al. v [19]. V té se autoři rozhodli klást hlavní důraz na maskování základu mocniny x. Maskování je provedeno vynásobením x náhodným číslem r. Hodnota

x^d je poté vypočítána jako $(xr)^d \times (r^{-1})^d$. Pseudokód takto randomizovaného algoritmu má potom tuto podobu:

Algorithm *Randomized montgomery ladder*
Input: g, $e = (e_{t-1}, e_t ... e_1, e_0)_2$
Output: g^e
1. Zvolte náhodné r.
2. $R_0 \leftarrow r$, $R_1 \leftarrow rg$, $R_2 \leftarrow r^{-1}$;
3. **for** $i \leftarrow t-1$ **downto** 0 **do**
4. $\quad R_{\neg e_i} \leftarrow R_0 R_1$;
5. $\quad R_{e_i} \leftarrow (R_{e_i})^2$;
6. $\quad R_2 \leftarrow R_2^2$
7. **return** $R_2 R_0$.

Kapitola 6

Ovládání osciloskopu PicoScope

Ke sběru dat pro odběrovou analýzu byl v rámci této práce použit osciloskop PicoScope 4224. Jedná se o externí PC zařízení komunikující s počítačem pomocí USB rozhraní. Tento typ osciloskopu nabízí možnost měření na dvou kanálech současně při maximální vzorkovací frekvenci 80 MHz. Zařízení je schopno během jednoho odběru uchovat až 32 milionů snímků, což při nejvyšší frekvenci znamená až 400 ms nepřetržitého sběru dat.

K ovládání svých osciloskopů nabízí výrobce dvě alternativy. První možností je využití aplikačního softwaru *PicoScope 6*. Měření v jeho grafickém rozhraní je však poměrně časově náročné a pro potřeby této práce, nevhodné. Mnohem flexibilnější řešení poskytuje balík *SDK* (Software Development Kit[1]) dodávaný spolu s osciloskopem a zároveň dostupný z webu výrobce[5]. Součástí SDK je i dynamická knihovna `ps4000.dll`, která slouží jako ovladač API (Application Programming Interface[2]) pro ovládání osciloskopu při tvorbě vlastního softwaru.

Následující podkapitoly jsou věnovány vysvětlení základních vlastností a prvků tohoto API a dále pak popisu programu vytvořeného pro ovládání osciloskopu v rámci této práce.

6.1 Seznámení s API pro sérii osciloskopů PicoScope 4000

Jak již bylo zmíněno výše, základním ovladačem výrobcem dodávaného API je dynamická knihovna `ps4000.dll`. Tato knihovna umožňuje programovat osciloskopy série PicoScope 4000 v jazyce C (nebo v jiném příbuzném jazyce, např. C++) pomocí volání knihovních API funkcí. Dostupných funkcí je 56, jejich význam a použití jsou popsány v programovacím manuálu, který je součástí balíku SDK.

Správnou komunikaci uživatelské aplikace s osciloskopem zajišťují dva hlavičkové soubory rovněž přiložené v balíku SDK. Jedná se o soubor `pi-`

1. Překlad autora: *Sada pro vývoj software*
2. Překlad autora: *Aplikační programovací rozhraní*

`coStatus.h`, který pomocí série maker definuje možné normální a chybové stavy osciloskopu, a soubor `ps4000Api.h` obsahující definice několika datových typů potřebných pro snadnější nastavení parametrů měření a definici standardních volání knihovních API funkcí. Propojení vlastního programu s ovladačem API je zajištěno přidáním `ps4000Api.h` direktivou `#include` na začátek souboru.

Programování pomocí API dává uživateli možnost vybrat si z několika různých vzorkovacích režimů osciloskopu. Osciloskopy série PicoScope 4000 nabízí režimy tři:

- **Blokový režim** (Block mode)
- **Rychlý blokový režim** (Rapid block mode)
- **Streamovací režim** (Streaming mode)

V blokovém režimu jsou data ukládána do interní paměti RAM a po dokončení odběru přenesena do PC. Data jsou ztracena při opětovném spuštění osciloskopu. Měření v tomto režimu umožňuje využít maximální vzorkovací frekvenci osciloskopu, z toho důvodu je využit i pro odběrovou analýzu v této práci.

Použitím rychlé varianty blokového režimu jsou minimalizovány časové mezery při navazování nových bloků. Toho využijeme, chceme-li sbírat data ve větším objemu, než nám umožňuje velikost jednoho bloku interní paměť osciloskopu.

Ve streamovacím režimu nejsou data ukládána v interní RAM, ale jsou přenášena do počítače přímo. Uživateli je tak umožněn dlouhodobý odběr za cenu snížení vzorkovací frekvence.

Podrobnějšímu popisu typické struktury ovládacího programu, jeho základních prvků a nejdůležitějších API funkcí je věnována příloha A.

6.2 Vytvoření ovládacího programu

Účelem programu vytvořeného v rámci této práce bylo pomocí dodávaného API připravit osciloskop pro měření, nadefinovat vhodnou spouštěcí událost (trigger), s danými parametry naměřit data a uložit je do formátu vhodného pro vizualizaci v softwaru SACC-Lite (Smart Card Analysis Control Center) nebo Matlab.

Pro programování aplikace byl zvolen jazyk C. Hlavním důvodem pro výběr byla možnost nahlédnout a čerpat z ukázkových příkladů, které jsou součástí balíku SDK, a které jsou rovněž implementovány v jazyce C.

Kompletní zdrojový kód aplikace je obsažen na přiloženém DVD, v této podkapitole jsou uvedeny pouze jeho dva hlavní aspekty:

- **Nastavení vstupních kanálů** – Osciloskop PicoScope 4224 umožňuje odběr na dvou kanálech současně. Pro samotnou odběrovou analýzu na čipové kartě by bylo dostačující využít pouze jednoho z nich, z důvodu omezení šumu měření jsou však zapojeny oba. Vznikne tak matematický kanál, ve kterém je výsledný vzorek spočítán jako rozdíl hodnot na kanálu A a B. Způsob fyzického zapojení a vysvětlení souvislostí jsou popsány v 7.1. Rozsah napětí je na obou kanálech nastaven na $\pm$ 1V.

- **Nastavení spouštěcí události** – Po sérii pokusných měření bylo jako nejvhodnější spouštěcí událost vybráno jednoduché překročení hranice 300mV ve stoupajícím směru. Spouštění probíhá na kanálu A a samotné měření je zahájeno 480 ms po prvním výskytu triggeru.

Přestože dostupný osciloskop umožňuje v blokovém režimu během jednoho odběru naměřit až 32 milionů snímků (rozdělených mezi aktivní kanály), pro pokrytí celého měření v rámci této práce při maximální frekvenci stačilo využít pouze 6,4 milionu pro každý kanál. Výrazně se tím zmenšila velikost výstupního souboru. Výstupní data jsou uspořádána do formy textového souboru (s příponou `.dat`), ve které každému řádku odpovídá právě jedna hodnota naměřeného vzorku. K hodnotě každého řádku je po skončení odběru přičtena konstantní hodnota 4000 mV. Tento krok byl zaveden z důvodu občasného výskytu záporných bodů ve výsledné stopě, které způsobovaly pády softwaru SACC Lite, v němž byla data zpracovávána.

Kapitola 7

Získávání dat a analýza výsledků

Jako součást této práce byl implementován a do několika čipových karet nainstalován JavaCard applet, který měl za úkol vyvolat na kartě šifrování algoritmem RSA. Hlavním úkolem bylo pomocí odběrové analýzy sledovat průběh tohoto šifrování a pokusit se na základě naměřených dat určit, jakým způsobem je karta proti danému druhu útoku chráněna.

Tato kapitola je věnována přípravě měření pro odběrovou analýzu a především pak prezentaci výsledků praktické části práce.

7.1 Příprava měření

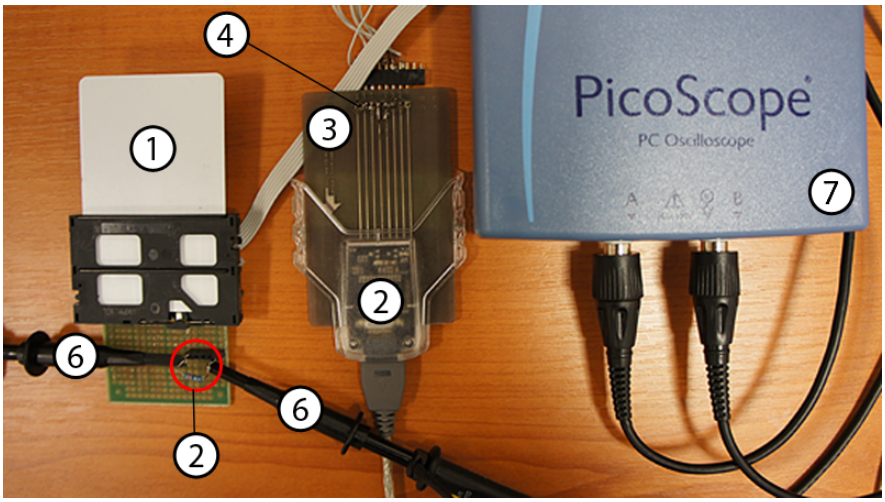
7.1.1 Zapojení měřícího zařízení

Způsob zapojení měřícího zařízení pro potřeby odběrové analýzy nejlépe ilustruje obrázek 7.1. Jednotlivé komponenty soustavy jsou na obrázku očíslovány (dále v textu čísla v závorkách). K fyzickému propojení karty (1) a čtecího zařízení (2) je využita inverzní čtečka karet (3), která umožňuje vložit mezi vodič spojující konektor GND (4) malý rezistor (5). Před a za tímto rezistorem jsou připojeny měřící sondy (6) osciloskopu (7).

7.1.2 Omezení šumu a čistota měření

Problémem, se kterým se často potýkají elektrická měření nejen při odběrové analýze, jsou velké nepřesnosti v získaných datech způsobené zaneseným šumem. Měření nejvýznamněji ovlivňují dva typy šumu. Prvním typem je šum vznikající uvnitř měřící soustavy, druhý je na měřený objekt přenášen z okolního prostředí ve formě elektromagnetického vlnění. K zpřesnění měřených dat je nutné implementovat určité filtrovací mechanizmy. Několik takových opatření bylo použito i v této práci.

Prvního omezení šumu je dosaženo využitím principu tzv. diferenciální sondy popsaného např. v [21]. Obě dostupné sondy jsou zapojeny tak, jak je vidět na obrázku 7.1. Sonda A odebírá data před rezistorem, sonda B



Obrázek 7.1: Obrázek zapojené měřící soustavy.

za ním. Elektromagnetické vlnění interferuje s vodičem na obou stranách rezistoru, pokud ale hodnoty z kanálu A i B odečteme, získáme data blížící se skutečnému odběru a většina interferencí je odstraněna.

Druhým zavedeným opatřením bylo zkrácení vodičů spojující čtecí zařízení s inverzní čtečkou na co nejmenší možnou délku. Snahou bylo minimalizovat na nich interference elektromagnetického vlnění z okolí.

7.2 Analýza výsledků měření

Jedním ze stěžejních úkolů této práce bylo provést na několika čipových kartách měření pro odběrovou analýzu a pokusit se ze získaných dat odvodit co nejvíce informacích vypovídajících o použitých ochranných mechanizmech bránících tomuto druhu útoku.

Hlavním cílem bylo na dostupných kartách potvrdit použití maskování tajných dat, zejména pak tajného exponentu algoritmu RSA. Postupným rozborem naměřených stop však bylo nalezeno i několik dalších zajímavých souvislostí. Prezentaci všech praktických výsledků práce jsou věnovány následující podkapitoly.

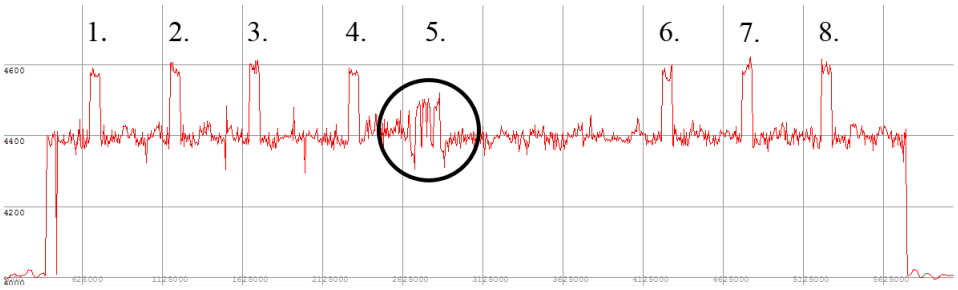
7.2.1 Odběrová analýza čipové karty Gemplus GXP E64 PK

Gemplus GXP E64 PK je karta, které byla v rámci této práce věnována největší pozornost.

První seznámení s odběrovou stopou

Za účelem prvního pozorování odběrové stopy dané karty byl vytvořen jednoduchý JavaCard applet, který sledovanou operaci šifrování algoritmem RSA ohraničil z obou stran trojicí generování náhodných čísel. Tato generování by měla díky své výpočetní náročnosti zanechávat výraznou odběrovou stopu a pomoci tak snazší identifikaci cílové operace. Daný předpoklad se ukázal jako pravdivý.

Na výsledné stopě zobrazené na obrázku 7.2 je zřetelný nárůst spotřeby energie z 0 na 400 mV (resp. ze 4000 na 4400 mV, vysvětlení viz. 6.2) ve chvíli, kdy byl applet spuštěn. Výrazné jsou rovněž další oblasti reprezentované zvýšeným odběrem. Vrcholy označené v obrázku čísly 1 – 3 a 6 – 8 představují oblast generování náhodných čísel, číslo 4 může díky své podobnosti s předchozími patřit generování náhodné masky pro překrytí šifrované zprávy a exponentu. Číslem 5 a černým kruhem je zvýrazněna identifikovaná *oblast šifrování algoritmem RSA*.

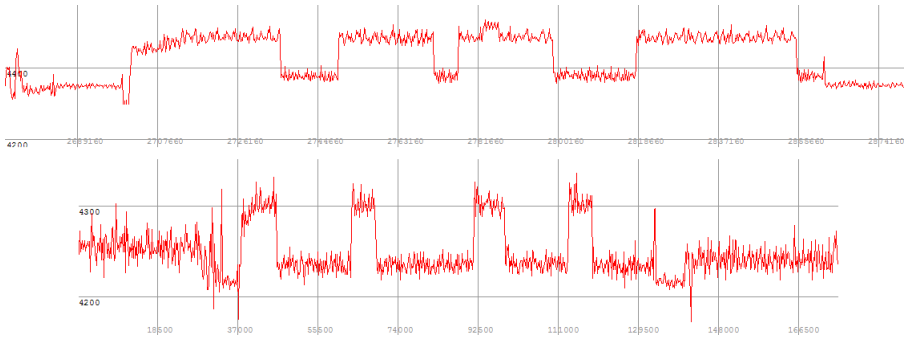


Obrázek 7.2: Celkový průběh odběru energie při vykonávání instrukcí jednoduchého appletu s identifikovanou oblastí šifrování RSA.

Poté, co je odhalen tvar odběrové stopy cílové operace a určena její pozice, je možné se na operaci zaměřit podrobněji.

Pozorování výřezu operace šifrování

Už při pouhém pohledu na detailnější výřez oblasti šifrování (obrázek 7.3) jsou patrné 4 výrazné vrcholy oddělené mezerami se sníženým odběrem. Tato vlastnost platí jak pro klíč délky 1024 bitů, tak délky 512 bitů. Jelikož šifrování vykazuje na obou délkách klíčů velmi podobné chování, bylo za účelem snížení objemu zpracovávaných dat rozhodnuto vdalší fázi experimentu pokračovat s délkou klíče 512 bitů.



Obrázek 7.3: Výřez oblasti šifrování RSA. Horní stopa pro klíč délky 1024 bitů, dolní 512 bitů. Délky obou operací jsou ve stejném poměru, výšky nikoliv.

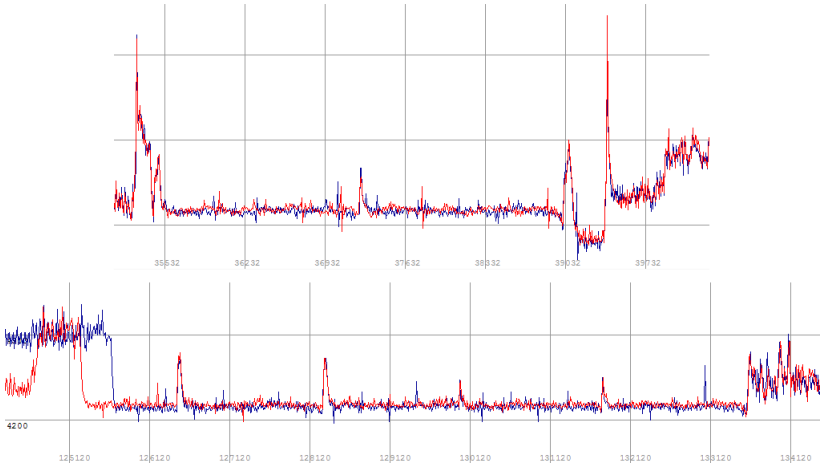
Třemi základními prvky odběrové stopy, kterým byla věnována bližší pozornost jsou:

1. **Jednotlivé vrcholy** – Zvýšený odběr energie představuje oblast, kdy na kartě probíhá výpočet kryptografického koprocesoru. Jelikož víme, že karta provádí algoritmus RSA, jedná se pravděpodobně o výpočet modulárního umocňování.

2. **Mezery oddělující jednotlivé vrcholy** – Výpočet modulárního umocňování viditelně neprobíhá souvisle, ale je přerušován oblastmi s výrazně sníženým odběrem. Na kryptografickém koprocesoru je výpočet dočasně ukončen a karta vykonává energeticky méně náročné operace.

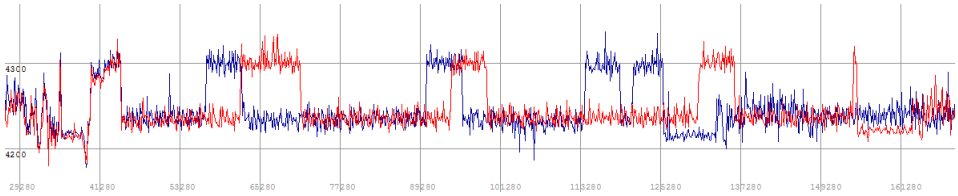
3. **Počátek a konec šifrovací oblasti** – Obrázek 7.4 zobrazuje odběrovou stopu charakterizující oblast, kdy karta vstupuje do procesu šifrování. Zajímavou vlastností pozorovanou na tomto obrázku je velká podobnost dvou stop (jedna modrá, druhá červená), které pocházejí z různých měření. Hladina napětí v daných oblastí je navíc vůči okolí nízká, lze tedy usuzovat, že karta před začátkem měření a po jeho ukončení provádí vždy stejné jednoduché zaváděcí operace.

Opakováním měření bylo vypozorováno několik dalších vztahů mezi těmito jednotlivými prvky. Počet vrcholů, do kterých je rozděleno modulární umocňování, není pokaždé konstantní. Jak je vidět na obrázku 7.5, může stopa obsahovat 4 nebo 5 (výjimečně 3) těchto vrcholů. Data pocházejí ze šifrování, pro které byl použit vždy stejný klíč a stejná šifrovaná data. Zajímavá je tedy i výrazně rozdílná délka celého průběhu RSA. Konstantní na-



Obrázek 7.4: Výřez počátku (nahoře) a konce (dole) šifrovací oblasti s výraznou podobností dvou stop pocházejících z různých měření.

víc není ani délka jednotlivých vrcholů, ani mezer, které je oddělují. Příčin takovéhoto chování karty může být více, vyvozovat závěry z několika málo provedených měření by však bylo velmi předčasné. Z tohoto důvodu bylo úkolem další fáze experimentu provést velké množství měření a k analýze získaných dat využít některé statistické metody.



Obrázek 7.5: Ukázka rozdílnosti dvou šifrování stejných dat se stejným klíčem. Stopy jsou synchronizované na začátek prvního vrcholu.

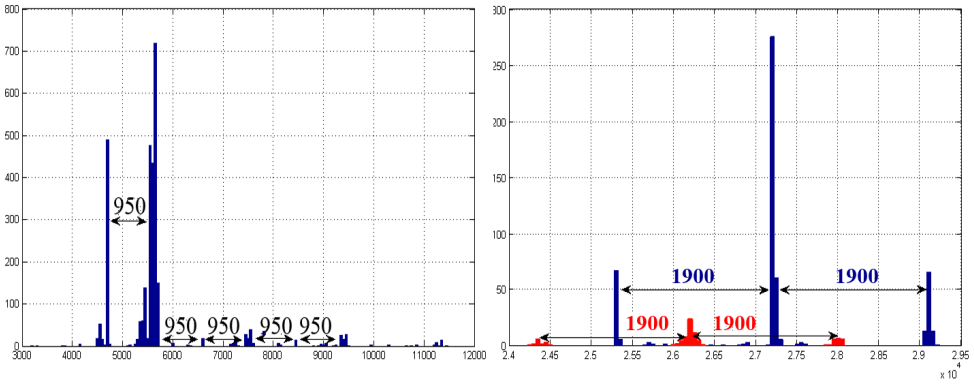
Statistické zpracování naměřených dat

Základní hypotéza, kterou mělo statistické zpracování dat potvrdit nebo vyvrátit, sledovala vztahy mezi délkami vrcholů a mezer, které tyto vrcholy oddělují. Předpokladem bylo, že celé modulární umocňování probíhá uvnitř kryptografického koprocesoru a jeho jinak souvislý výpočet je několikrát přerušen a proložen náhodně dlouhými instrukcemi, které na

konečný výsledek šifrování nemají vliv. K této hypotéze vedlo několik faktů patrných z výše uvedených obrázků, například výrazně odlišná délka mezer. Pokud by se v těchto oblastech zpracovávaly nikoliv pouze náhodné instrukce, ale určitá data vyprodukovaná předešlým modulárním umocňováním, měl by být jejich objem a tím i délka operace vždy podobné.

K ověření předešlé hypotézy bylo provedeno 600 měření odběru RSA při šifrování stejných dat stejným šifrovacím klíčem a 100 měření při šifrování náhodných dat stejným klíčem. Z naměřených stop byly vyřezány všechny vrcholy i oddělující mezery a do samostatných souborů byly zaznamenány jejich délky. Kvůli jejich rozsahu nejsou konkrétní hodnoty vypsány v textu práce, ve formátu `xls` se však nacházejí ve složce `namerena_data` na přiloženém DVD.

První zpracovaná statistika se zabývala délkami jednotlivých vrcholů a jejich součtů. Sledovanou vlastností bylo především rozdělení četností jednotlivých hodnot. K tomuto účelu vznikly dva histogramy[1] zobrazené na obrázku 7.6.



Obrázek 7.6: Histogramy délek jednotlivých vrcholů (vlevo) a jejich součtů (vpravo) při šifrování stejných dat stejným klíčem se znázorněnou vzdáleností mezi jednotlivými centry.

Z obou grafů na obrázku je patrné, že rozložení délek ani jejich součtů není rovnoměrné. Jednotlivé hodnoty jsou koncentrovány v okolí několika „center“, jejichž vzdálenost je konstantní. Výsledky histogramu délek jednotlivých vrcholů napovídají, že celkový výpočet modulárního umocňování není přerušován náhodně, nýbrž v místech určených jistým násobkem

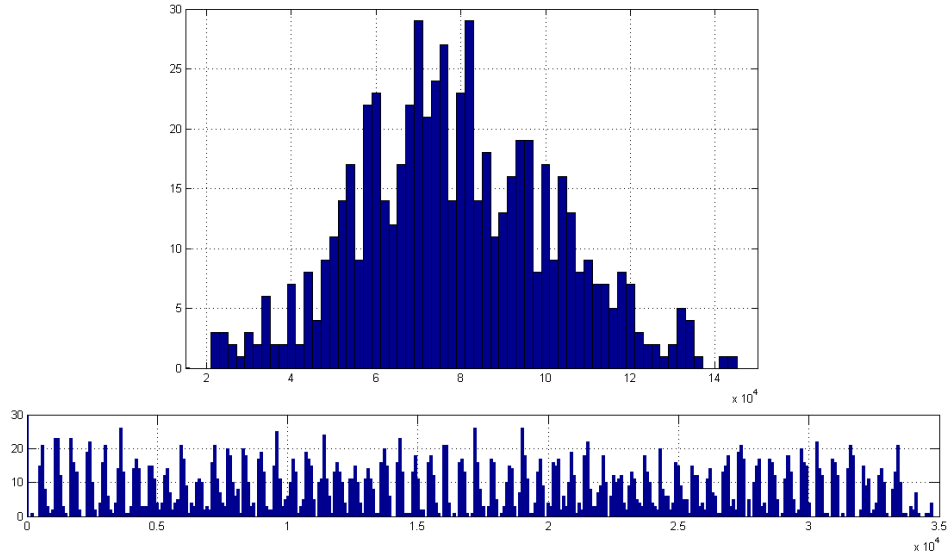
1. Histogram je grafické znázornění distribuce dat pomocí sloupcového grafu se sloupci stejné šířky, vyjadřující šířku intervalů (tříd), přičemž výška sloupců vyjadřuje četnost sledované veličiny v daném intervalu. [22]

čísla 950. Můžeme tedy usuzovat, že se tento výpočet skládá z jednotlivých operací, z nichž každá má délku 950 (vzorků).

Z druhého grafu navíc vyplývá, že ani celkových součet délek není rozložen náhodně. Vzdálenosti center se rovněž pohybují v násobku konstantního čísla, tentokrát 1900. Za zdůrazněné stojí fakt, že $1900 = 2 \times 950$.

Druhý histogram dále ukázal vztah dvou různých trojic vrcholů, kdy je první z nich zvýrazněna červeně, druhá modře. Centra v nich mají stejné vzdálenosti, vzájemně jsou však posunuty. Možným vysvětlením by mohlo být implementování techniky *exponent blinding* (5.2) jako ochrany proti časové a odběrové analýze. Exponent by v takovém případě byl před každým výpočtem prodloužen o jiný násobek $\phi(n)$. O tento násobek se pak změní i celkový výpočet.

Stejnou metodou byla zpracována i data získaná změřením délek oddělujících mezer. Oba vygenerované histogramy jsou zobrazeny na obrázku 7.7.



Obrázek 7.7: Histogramy délek jednotlivých mezer (dole) a jejich součtů (nahoře) při šifrování stejných dat stejným klíčem.

Už při prvním pohledu se tyto grafy od předchozích výrazně liší. Patrně nejzajímavější vlastnost vykazuje graf součtů délek mezer. Rozložení četností jednotlivých hodnot se blíží tzv. normálnímu rozložení, které je charakteristické pro náhodné veličiny a jeho graf má tvar typické Gaussovy křivky. Toto zjištění potvrzuje hypotézu stanovenou v úvodu bloku

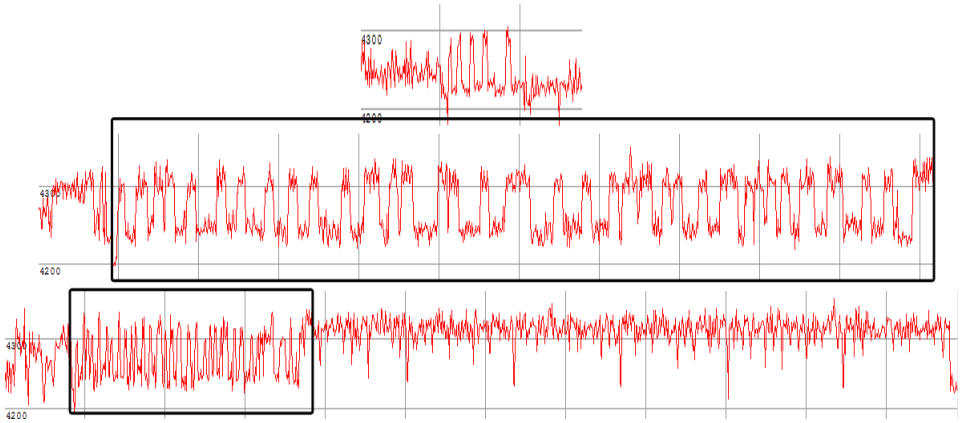
„Statistické zpracování naměřených dat“, tedy že průběh modulárního umocňování je přerušován náhodně dlouhými falešnými instrukcemi.

Graf jednotlivých délek má naopak rozložení hodnot velmi rovnoměrné a pravidelné, nikoliv však souvislé. Můžeme tedy usuzovat, že je celková náhodná délka rozdělena vždy v určitém poměru do jednotlivých bloků, kterými je modulární umocňování v koprocesoru prokládáno.

Je pravděpodobné, že se výrobce karty snaží výše uvedenými mechanizmy zamezit útočníkovi v útoku diferenciální odběrovou analýzou. Předpokládáme-li však, že instrukce, kterými je výpočet modulárního umocňování prokládán, jsou náhodné a nemají vliv na výsledek šifrování, můžeme je z odběrové stopy vyřezat, jednotlivé vrcholy spojit a provést DPA na nich. Jeden obranný mechanizmus karty by tak byl zcela eliminován.

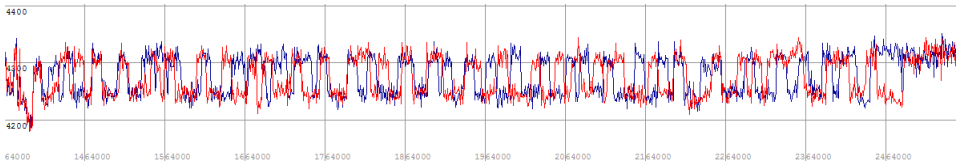
Pozorování maskování soukromého exponentu

Předchozí fáze experimentu sledovaly algoritmus RSA v průběhu šifrování dat. Následující měření byla zaměřena na postup opačný, tedy na jejich dešifrování a digitální podepisování. Hlavní rozdíl v obou operacích z hlediska odběrové stopy spočívá v délce celého výpočtu. Zatímco veřejný klíč bývá obvykle malý (typicky hodnota 65 537), soukromý klíč nabývá délky blížící se délce modulu (v tomto experimentu 512 bitů). Úměrně tomu je prodloužena i doba výpočtu a délka odběrové stopy. Toto tvrzení potvrzují i stopy na obrázku 7.8



Obrázek 7.8: Průběh šifrování (nahoře), část (uprostřed) a celé (dole) dešifrování algoritmem RSA. Délka i výška dvou horních stop jsou ve stejném poměru, poslední jim neodpovídá.

Stopa celkového průběhu dešifrování (na obrázku zcela dole) se skládá ze třech hlavních částí. První je označena černým oknem a pravděpodobně odpovídá oblasti vrcholů a oddělujících mezer známé z šifrovacího směru algoritmu. Počet těchto vrcholů, délka mezer mezi nimi i celková doba operace se opět liší (dvě proložené a synchronizované stopy jsou ukázány na obrázku 7.9). To ukazuje na skutečnost, že i zde implementoval výrobce karty velmi podobné ochranné mechanizmy popsané výše v této kapitole.



Obrázek 7.9: Výřez oblasti modulárního umocňování při dešifrování algoritmem RSA ukazující rozdílnost dvou stop z různých měření..

Druhá ze zmiňovaných oblastí označenému oknu předchází. Jedná se o tři oddělené vrcholy s poměrně vysokým odběrem energie. Vzhledem k tomu, že se tyto vrcholy ve stopě objevují i během šifrování, avšak s výrazně nižší hladinou odběru a délkou, lze usuzovat, že se jedná o jisté zpracovávání klíče před vstupem do modulárního umocňování. S jistou pravděpodobností se tak může jednat o maskování klíče před vstupem do procesu dešifrování.

Třetí oblast označené okno následuje. Je složena z šesti částí s výraznou spotřebou oddělených krátkou mezerou. Pro jejich výskyt zatím nebylo nalezeno přesvědčivé vysvětlení. Vzhledem k množství energie, které tyto operace spotřebovávají se pravděpodobně jedná o výpočty uvnitř kryptografického koprocesoru.

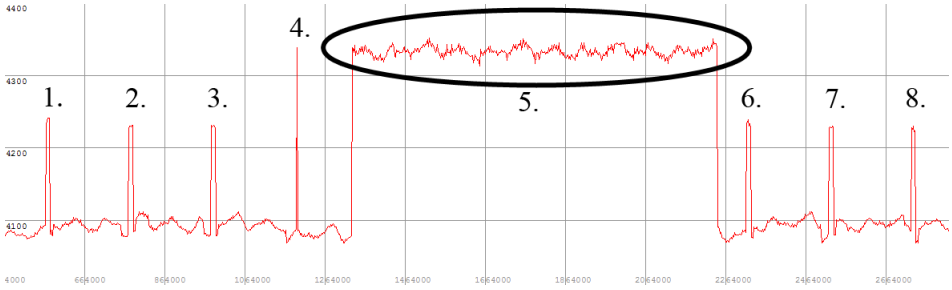
7.2.2 Odběrová analýza čipové karty Gemplus Twin GCX4 72k PK

Druhou kartou, na které byla provedena odběrová analýza je karta Gemplus Twin GCX4 72k PK. Důvodem pro výběr právě této karty jsou její zcela odlišné odběrové vlastnosti oproti Gemplus GXP E64 PK.

První seznámení s odběrovou stopou

Pro první pozorování odběrové stopy byl opět použit jednoduchý JavaCard applet, který sledovanou operaci šifrování RSA ohraničil vždy trojicí generování náhodných čísel.

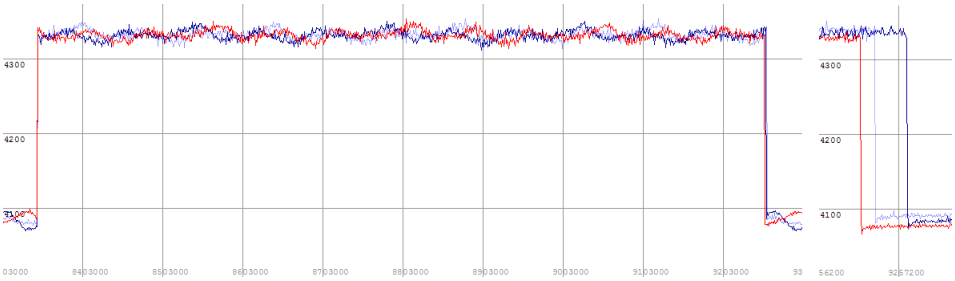
Na obrázku 7.10 jsou i tentokrát zřetelné oblasti generování náhodných čísel (1 – 3 a 6 – 8), oblast generování náhodné masky pro překrytí zprávy a exponentu (4). Odlišnost oproti předchozí kartě je zřejmá zejména při pohledu na oblast šifrování RSA, která je na obrázku označena číslem 5 a černým kruhem.



Obrázek 7.10: Celkový průběh odběru energie při vykonávání instrukcí jednoduchého appletu s identifikovanou oblastí šifrování RSA.

Pozorování výřezu operace šifrování

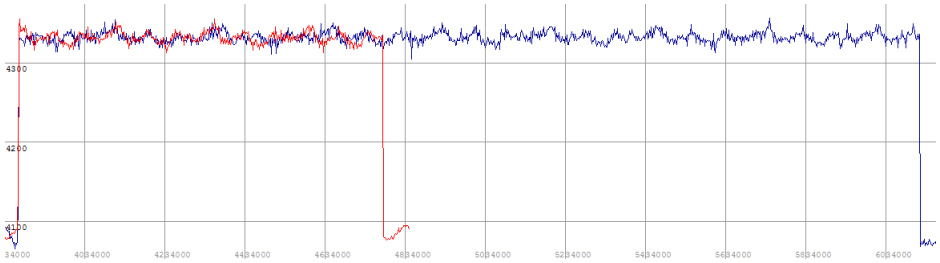
Základní myšlenkou, jak útočníkovi znesnadnit odběrovou analýzu, je zřejmě snaha udržet všechny významné odběrové charakteristiky konstantní. Z výřezů oblastí šifrování na obrázku 7.11 je vidět, že rozdíly v délce celé operace jsou pozorovatelné až při velkém přiblížení a pohybují se pouze v řádech desítek vzorků. V průběhu šifrování navíc nejsou vidět žádné extrémní výkyvy odběru proudu jaké byly přítomné na předchozí kartě.



Obrázek 7.11: Celkový průběh odběru energie při vykonávání instrukcí jednoduchého appletu s identifikovanou oblastí šifrování RSA.

Pozorování operace dešifrování

Kromě pozorování algoritmu RSA v šifrovacím směru byla pozornost věnována i odběrové analýze dešifrování, kdy se do výpočtu zapojil soukromý exponent. Jak je však vidět na obrázku 7.12, jediným rozdílem oproti odběrové stopě šifrování je v délce celé operace, ostatní výše charakterizované vlastnosti zůstávají stejné.



Obrázek 7.12: Celkový průběh odběru energie při vykonávání instrukcí jednoduchého appletu s identifikovanou oblastí šifrování RSA.

Kapitola 8

Závěr

Velké množství úspěšně provedených a publikovaných útoků postranním kanálem je důkazem, jak velkou hrozbu pro čipové karty představují. S rostoucím propojením karet a nejrůznějších aplikací a systémů je stále patrnější nutnost implementace účinných obranných mechanizmů.

Hlavním cílem této práce bylo provést na několika čipových kartách odběrovou analýzu algoritmu RSA. Na jejím základě se pak pokusit odvodit co největší množství informací vypovídajících o nasazení a implementaci ochranných mechanizmů. Závěry analýzy dat na všech testovaných kartách ukazují snahu výrobců maximálně znesnadnit útočníkovi vyzrazení citlivých dat. Jak však naznačují prezentované výsledky, nemusí být všechna opatření zcela účinná.

Nejzajímavějších výsledků bylo dosaženo především s využitím metod statistické analýzy na velkém vzorku dat. Na jedné z karet byl například díky porovnání velkého množství naměřených stop odhalen nevhodný způsob obrany proti diferenciální odběrové analýze. Ten je realizován pomocí vkládání náhodně dlouhých falešných instrukcí do výpočtu algoritmu RSA. Podklady pro tvrzení, že je délka instrukcí skutečně náhodná, přinesla právě statistická analýza.

Kromě své hlavní náplně se práce věnuje i některým k ní přidruženým tématům. Podrobnějšího zpracování se tak dostalo především rozboru a popisu ovládání osciloskopu PicoScope pomocí výrobcem dodávaného API. Shromážděné informace o jednotlivých API funkcích a struktuře typického ovládacího programu mohou začínajícím uživatelům posloužit jako návod a uvést jej do problematiky programování osciloskopu.

V průběhu provádění experimentu se stále objevovaly nové zajímavé náznaky, na jejichž důkladnější analýzu nebyl v průběhu vytváření práce dostatečný prostor. V naměřených odběrových stopách se například objevují jisté opakující se vzory vykazující různé výkyvy ve své délce a poloze. Jejich význam se prozatím nepodařilo objasnit. Autor by rád navázal

na již odvozené závěry a v rozboru obranných mechanizmů čipových karet nadále pokračoval. Prostor pro další práci se nachází především v analýze výše zmíněných náznaků a především v rozšíření odběrové analýzy na další typy čipových karet.

Literatura

[1] RANKL, W., EFFING W. Smart card handbook. 4th ed. Překlad Kenneth Cox. Chichester: John Wiley, 2010, 1043 s. ISBN 978-0-470-74367-6. 3, 4, 7

[2] QUISQUATER, J. *Side channel attacks – State-of-art* [online]. 1.10.2002, [cit. 13.11.2011]. Dostupné na: <http://www.ipa.go.jp/security/enc/CRYPTREC/fy15/doc/1047_Side_Channel_report.pdf> 7

[3] CHEN, Z. *Technology for Smart Cards: Architecture and Programmer's Guide*. Addison-Wesley, MA, USA, 2000. 6

[4] *Útok postranním kanálem* [online], aktualizace 16.12.2010, [cit. 13.11.2011], Wikipedie. Dostupné na: <http://cs.wikipedia.org/wiki/Útok_postranním_kanálem> 7

[5] Pico Technology. *Drivers and Examples for Data Loggers and Oscilloscopes* [online]. [cit. 13.4.2012]. Dostupné na: <http://www.picotech.com/drivers.html> 17

[6] KOCHER, P. *Timing Attacks on Implementation of Diffie-Hellman, RSA, DSS and Other Systems* [online]. [cit. 20.2.2012]. Dostupné na: <http://www.cryptography.com/public/pdf/TimingAttacks.pdf> 7, 13

[7] Sun Developer Network. *Java Card Technology Overview* [online]. [cit. 5.3.2012]. Dostupné na: <http://java.sun.com/javacard/overview.jsp> 5

[8] BONEH, D., DeMILLO, R.A., LIPTON, R.J. *On the Importance of Checking Cryptographic Protocols for Faults*. EUROCRYPT, volume 1233 of Lecture Notes in Computer Science, pp. 37–51. Springer, December 1997. 7

[9] KŮR, J. *Algoritmy pro differenciální odběrovou analýzu kryptografických čipových karet (DPA)* [online]. 2006 [cit. 2012-05-12]. Bakalářská práce. Masarykova univerzita, Fakulta informatiky. Vedoucí práce Petr Švenda. Dostupné z: <http://is.muni.cz/th/98692/fi_b/bc_thesis_Jiri_Kur.pdf>. 8

[10] KOCHER, P., JAFFE, J., JUN, B. *Introduction to Differential Power Analysis and Related Attacks* [online]. 1998, [cit. 20.2.2012]. Dostupné na: <http://www.cryptography.com/public/pdf/DPATechInfo.pdf> 8

[11] RIVEST, R. L., SHAMIR, A., ADLEMAN, L. *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*. Communications of the ACM 21, 1978, str. 120–126. 11

[12] MANGARD, S., OSWALD, E., POPP, T. *Power analysis attacks: revealing the secrets of smart cards*. New York: Springer, c2007. xxiii, 337 s. ISBN 978-0-387-30857-9. 9

[13] MEDWED, M., HERBST, Ch. *Randomizing the Montgomery Multiplication to Repel Template Attacks on Multiplicative Masking*. COSADE 2010 Workshop Proceedings, 2010, str. 56 – 71. 12

[14] ITOH, K., YAJIMA J., TAKENAKA, M., TORII, N. *DPA Countermeasures by Improving the Window Method*. Cryptographic Hardware and Embedded Systems – CHES 2002. Lecture Notes in Computer Science, 2003, Volume 2523/2003, pp. 303–317. 14

[15] KOÇ, ÇETIN K., ed. *Cryptographic engineering*. New York: Springer, 2009. xxii, 517 s. ISBN 978-0-387-71816-3. 14

[16] JIN, J., LU, E., Gao, X. *Resistance DPA of RSA on Smartcard*. Information Assurance and Security, 2009. IAS '09. Fifth International Conference on Information Assurance and Security, vol.2, no., pp.406-409, 18-20 Aug. 2009 13

[17] JOYE, M. a S. YEN. *The Montgomery Powering Ladder*. Cryptographic hardware and embedded systems–CHES 2002: 4th international workshop, Redwood Shores, CA, USA, August 13-15, 2002 : revised papers. New York: Springer-Verlag, c2002, č. 2523, s. 291. 15

[18] MONTGOMERY, P. L. *Speeding the Pollard and elliptic curve methods of factorization*. Mathematics of computation. 1987, roč. 48, č. 177, s. 243–264 15

[19] FUMAROLI, G. a D. VIGILANT. *Blinded Fault Resistant Exponentiation*. Fault diagnosis and tolerance in cryptography: third international workshop, FDTC 2006, Yokohama, Japan, October 10, 2006 : proceedings. Berlin: Springer, c2006, s. 62–70. 15

[20] CORON, J. *Resistance Against Differential Power Analysis For Elliptic Curve Cryptosystems*. Cryptographic hradware and embedded systems. Berlin: Springer Verlag, 1999, vol. 1717. 12

[21] OTT, H. *Noise reduction techniques in electronic systems*. 2nd ed. New York: John Wiley, 1988, 426 s. ISBN 04-718-5068-3. 20

[22] KUPKA, K. *Statistické řízení jakosti*. Pardubice: TriloByte, 1997, 191 s. ISBN 80-238-1818-X. 25

Příloha A

Struktura programu pro ovládání osciloskopu

Typický program pro ovládání osciloskopu pomocí volání knihovních API funkcí se skládá z následujících kroků:

1. **Spuštění měřící jednotky**

Každý program pro ovládání osciloskopu musí začít **spuštěním měřící jednotky** voláním funkce `ps4000OpenUnit()`. Její návratovou hodnotou je stejně jako u všech ostatních API funkcí jedna ze stavových konstant PICO_STATUS definovaná v souboru `picoStatus.h`. Jediným argumentem funkce je proměnná `handle` typu ukazatel na `short`, do které se po zavedení osciloskopu uloží přidělené číslo pro zařízení. Toto číslo je využíváno v dalším běhu programu pro správnou identifikaci osciloskopu.

2. **Nastavení vstupních kanálů**

Dalším důležitým krokem ovládacího programu je **nastavení vstupních kanálů**. Všechny kanály jsou před začátkem měření deaktivovány. Jako parametry funkce `ps4000SetChannel()` jsou jim předány rozhodnutí o aktivování a požadavky na rozsah napětí. Je také třeba rozhodnout, zda chceme měřit v AC (Alternating current – Střídavý proud) nebo DC (Direct current – Stejnosměrný proud) režimu.

3. **Nastavení spouštěcí události**

Následně po aktivování a nastavení vstupních kanálů vyžaduje osciloskop **nastavení spouštěcí události (trigger)**. Měření může být zahájeno okamžitě po spuštění osciloskopu nebo může čekat na výskyt spouštěcí události zvané trigger. V obou případech je třeba využít tyto tři funkce:

- Voláním `ps4000SetTriggerChannelConditions()` definujeme, na kterých kanálech očekáváme výskyt určité spouštěcí podmínky. Na různých kanálech je možné nastavit různé podmínky, výsledná spouštěcí událost je výsledek logické operace AND mezi nimi. Tato

funkce slouží pouze k určení kanálů využívaných pro triggrování, konkrétní hodnoty nastavuje funkce následující.

- Funkce `ps4000SetTriggerChannelProperties()` slouží ke konkrétnímu nastavení hodnot jednotlivých spouštěcích podmínek. Požadované hodnoty jsou funkci předávány jako argument typu ukazatel na pole struktur. Každá struktura v tomto poli nese informace pro jeden kanál. Uživateli je takto především umožněno nastavit hraniční hodnoty napětí (horní a dolní hranici) a hodnotu, o kterou je toto napětí potřeba překonat. Dále pak umožňuje vybrat typ spouštěcí události. Možnosti nabízí API dvě: klasický typ LEVEL, kdy měření začne po překročení hraničního napětí, a typ WINDOW, kdy osciloskop čeká, dokud se aktuální odběr nedostane do „okna“ ohraničeného horní a dolní hranicí spouštěcího napětí.

- Poslední ze základních triggrovacích funkcí je funkce `ps4000SetTriggerChannelDirections()`. Pomocí jejích argumentů předáváme osciloskopu pro každý kanál informaci, v jakém směru chceme, aby byla hraniční hodnota napětí překonána. Kompletní seznam možností je vypsán v programovacím manuálu, základními hodnotami jsou RISING, tedy stoupající směr, a FALLING, směr klesající.

Poslední možností, jak ovlivnit spouštěcí událost, je nastavení zpoždění začátku měření po jejím výskytu. K tomuto účelu slouží funkce `ps4000SetTriggerDelay()`.

Pro úplnost je třeba dodat, že výrobce nabízí k tomuto komplexnějšímu a složitějšímu způsobu nastavení spouštěcí události jednodušší alternativu. Tou je funkce `ps4000SetSimpleTrigger()`. Jedním jejím voláním vybereme správný kanál, nastavíme hodnotu hraničního napětí, určíme spouštěcí směr a definujeme zpoždění měření po výskytu triggeru.

4. **Zahájení sběru dat**

Ve chvíli, kdy je správně nastavena spouštěcí událost, můžeme začít se samotným sběrem dat. Výběr ovládací funkce se odvíjí od zvoleného vzorkovacího režimu. Měříme-li v jednom z blokových režimů, zahájíme měření funkcí `ps4000RunBlock()`, pokud ve streamovacím režimu, volíme funkci `ps4000RunStreaming()`. Oběma je jako hlavní argument předána hodnota určující vzorkovací frekvenci měření.

5. **Čekání na dokončení sběru dat**

U blokových režimů, kdy jsou výstupní data měření nejprve ukládána do interní paměti osciloskopu, je důležité počkat, dokud není celý blok naplněn. Teprve poté je možné data překopírovat do PC. Informaci o dokončení měření předává funkce `ps4000BlockReady()`. Není ji však nutné v programu volat přímo, ovladač API ji automaticky registruje při volání `ps4000RunBlock()`.

6. **Získání dat**

Jakmile je měření dokončeno, je třeba data uložená v interní paměti osciloskopu překopírovat do uživatelského počítače. K tomu účelu slouží funkce `ps4000GetValues()`, před jejím voláním je však nutné pomocí `ps4000SetDataBuffer()` určit pro každý kanál oblast paměti, do které chceme data uložit.

6. **Uzavření měřící jednotky**

Jakmile jsou všechna data překopírována, ukončíme funkcí `ps4000Stop()` měření.

Výše uvedený postup ukazuje pouze základní a nejdůležitější body tvorby ovládacího programu. API dodávané výrobcem však nabízí velké množství dalších funkcí, kterými je možné parametry měření ovlivnit. Jejich dokumentace a bližší popis je obsahem programovacího manuálu, který je součástí balíku SDK.

Příloha B

Pseudokódy

Algorithm *Triple masking*
Input: Plaintext m, celé náhodné číslo n, které je náhodné číslo pro plaintext, exponent $e = (e_t, e_{t-1}...e_1, e_0)_2$, náhodný pár čísel (v_i, v_d) splňující podmínku $v_d = (v^{-1})^e \ (mod\ n)$, celé náhodné číslo r, které se využije pro randomizace operace umocňování, náhodné číslo pro maskování instrukcí $N = (N_{k-1}, N_{k-2})...N_0$.
Output: m^e

1. /* Předpočítej maskování plaintextu. */
2. $m' = mv_i \ (mod\ n)$;
3. /* Předpočítání randomizace exponentu. */
4. $e' = e + r\phi(n)$;
5. Inicializace $k = 0$;
6. $s = 1$;
7. **for** $(i = t - 1; i > 0; i - -)$ **do**
8. a) **if** $(N_k == 1)$ spusť A_s ;
9. **else** spusť A_m ;
10. k++ ;
11. b) $s = s^2 \ (mod\ n)$;
12. c) **if** $(N_k == 1)$ spusť A_s ;
13. **else** spusť A_m ;
14. k++ ;
15. d) **if** $(e'_i == 1)\ s = sm \ (mod\ n)$;
16. e) **if** $(N_k == 1)$ spusť A_s ;
17. **else** spusť A_m ;
18. $s = v_d s \ (mod;\ n)$;
19. **return**;

Kde řádek 3 a operace a) a d) jsou operace modulárního umocňování a a), c), e) jsou vložené náhodné maskovací instrukce. A_s a A_m jsou operace bez efektu.

Algorithm *Sliding Window Exponentiation*
Input: g, $e = (e_t, e_{t-1}...e_1, e_0)_2$ s $e_t = 1$, a celé číslo $k \geq 1$
Output: g^e
1. $g_1 \leftarrow g$, $g_2 \leftarrow g^2$;
2. **for** $i \leftarrow 1$ **to** $(2^{k-1} - 1)$ **do** : $g_{2i+1} \leftarrow g_{2i-1} \cdot g_2$;
($*$ Předpočítá liché mocniny g^3... g^{2^k-1} $*$)
3. $A \leftarrow 1, i \leftarrow t$;
4. **while** $i \geq 0$ **do**
5. **if** $e_i = 0$ **then do** : $A \leftarrow A^2, i \leftarrow i - 1$;
6. **else do** : najdi nejdelší bitový řetězec $e_i e_{i-1}...e_l$ takový, že $i - l + 1 \leq k$ a $e_l = 1$ a proveď následující:
7. $A \leftarrow A^{2^{i-l+1}} \cdot g_{(e_i e_{i-1}...e_l)_2}$, $i = l - 1$;
8. **return** A.
($*$ Řádky 5–7 říkají: Pokud je bit e_i roven 0, vypočítej A^2, pokud je $*$)
($*$ roven 1, najdi bitový řetězec maximální délky k (velikost okna) $*$)
($*$ takový, že začíná a končí 1 a postupuj podle řádku 7. $*$)

Algorithm *Overlaping Window Method*
1. /* Předpočítání tabulkových dat. */
2. **for** $(i = 0; i < 2^k; i++)$ $tab[i] = a^i \ (mod\ n)$;
3. /* Vytváření okna w_i a délky překrytí h_i */
4. /* Vygeneruj náhodná čísla q a $0 < h_0, h_1, ..., h_{q-2} < k$,
5. která splňují $q \times k + (h_0 + h_1 + ...h_s) = u$.
6. Pro zabezpečení proti SPA jsou h_i doporučena
7. jako fixní hodnota $h \geq k/2$. */
8. $(h_i, q) = GenRandom(); u' = u - k; dt_{q-1} = bit(d, u-1, \ldots u')$;
9. **for** $(i = 0; i < q - 1; i++)$ **do**
10. $w_i = (Random\ number, max(0, dt_i - 2^{h_i} + 1) \leq w_i \leq dt_i)$;
11. $dt_{i+1} = (dt_i - w_i) \times 2^{k-h_i} + bit(d, u' - 1, \ldots, u' - (k - h_i))$;
12. $u' = u' - (k - h_i)$;
13. $w_{q-1} = dt_{q-1}$;
14. /* proces modulárního umocňování */
15. $v = tab[w_0]; i = 1$;
16. **while** $(i < q)$ **do**
17. $v = v^{2^{k-h_i}} \ (mod\ n); v = v \times tab[w_i] \ (mod\ n); i = i + 1$;
18. **return** v;

Algorithm *Randomized Table Window Method*

1. /* Vygeneruj náhodné číslo. */
2. $r = (b - bit\ random\ number)$;
3. /* Předpočítání tabulkových dat. */
4. $t = a^r\ (mod\ n)$;
5. **for** $(i = 0; i < 2; i++)\ tab[i] = a^{i \times 2^b} \times t\ (mod\ n)$;
6. /* Fáze vytváření oken. */
7. $dw = d; q = 0$;
8. **while** $(dw \geq r \times 2^{k \times q})$ **do**
9. $\quad dw = dw - r \times 2^{k \times q}; q = q + 1$;
10. **for** $(i = 0; i < q; i++)$ **do** ;
11. $\quad w_i = bit(dw, b + (q - i) \times k - 1, \ldots, b + (q - i - 1) \times k)$;
12. $d_m = dw\ (mod\ 2^b)$;
13. /* proces modulárního umocňování */
14. **if** $(q == 0)$ **return** $a^{d_m}\ (mod\ n)$;
15. $v = w_0; i = 1$;
16. **while** $(i < q)$ **do**
17. $\quad v = v^{2^k}\ (mod\ n); v = v \times tab[w_i]\ (mod\ n); i = i + 1$;
18. /* normalizace */
19. $v = v \times a^{d_m}\ (mod\ n)$;
20. **return** v;