Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE

Martina Válková

Útoky založené na hardwarových chybách

Katedra algebry

Vedoucí diplomové práce: Mgr. Štěpán Holub, Ph.D.
Studijní program: Matematika,
matematické metody informační bezpečnosti

2010

Prohlašuji, že jsem svou diplomovou práci napsala samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne 6. 12. 2010 Martina Válková

Obsah

Název práce: Útoky založené na hardwarových chybách
Autor: Martina Válková
Katedra (ústav): Katedra algebry
Vedoucí bakalářské práce: Mgr. Štěpán Holub, Ph.D.
e-mail vedoucího: Stepan.Holub@mff.cuni.cz

Abstrakt: V předložené práci studujeme využití hardwarových výpočetních chyb pro realizaci kryptoanalytických útoků. Zaměřujeme se konkrétně na návrhy útoků prezentovaných v článku Biham E., Carmeli Y., Shamir A.: Bug Attacks [1] a zkoumáme jejich praktické použití vůči schématům RSA a Pohlig-Hellman, včetně možnosti rozšíření a adaptace útoků na různé výpočetní okolnosti, přičemž upozorňujeme na vyšší zranitelnost schémat při realizaci modulárního mocnění použitím algoritmu Right-to-Left. Přinášíme výsledky praktického testování útoků prováděných vůči softwarové simulaci bugového procesoru, které potvrzují skutečnou bezpečnostní hrozbu uvažované situace. Čistě matematická část práce je věnována výpočetnímu předpokladu jednoho z útoků, problematice hledání odmocnin v $\mathbb{Z}_p$.

Klíčová slova: hardwarová chyba, útok, RSA, odmocniny modulo p

Title: Attacks based on hardware bugs
Author: Martina Válková
Department: Department of Algebra
Supervisor: Mgr. Štěpán Holub, Ph.D.
Supervisor's e-mail address: Stepan.Holub@mff.cuni.cz

Abstract: The study concerns hardware bugs producing computational errors and cryptanalytic attacks which utilize them. Particularly, the research is focused on attacks presented in the article by Biham E., Carmeli Y., Shamir A.: Bug Attacks [1] and their practical application in the case of schemes RSA and Pohlig-Hellman and various computational circumstances, which points out bigger vulnerability of schemes in the case of using the Right-to-Left modular exponentiation algorithm. The attacks have been tested against the software simulation of a faulty processor, which confirmed that they pose a real security threat in point of that situation. The mathematical part of this work concerns the problem of the finding any roots in $\mathbb{Z}_p$.

Keywords: hardware bug, attack, RSA, roots modulo p

Úvod

Práce se zaměřuje na kryptoanalytické útoky, které využívají výpočetních chyb dešifrovacího zařízení. Vycházíme z předpokladu, že dané dešifrovací zařízení obsahuje hardwarovou chybu (*bug*) v implementaci výpočetních instrukcí, kterou lze vhodným výběrem šifrových textů následně využít pro odhalení utajovaného (dešifrovacího) klíče. Současná asymetrická schémata užívaná v kryptografii jsou postavena především na operaci mocnění dlouhých čísel, tato operace je přitom typicky realizována prostřednictvím součinu. Její časté používání ve výpočtech se proto stává vhodným terčem útoků. Ze stejného důvodu se i zde v předpokladu zaměříme na hardwarové chyby projevující se při operaci násobení (*multiplication bug*). Tato myšlenka tzv. *bug-attacks,* včetně jejich základních principů, vychází z článku [1].

Tyto bug-útoky jsou do určité míry příbuzné s *fault-attacks*. Ty se soustřeďují na záměrné vyvolávání poruchy zařízení v určitém okamžiku výpočtu s cílem způsobit chybnost jeho výsledků, čehož bývá dosahováno provozováním činnosti zařízení v určitém (neobvyklém) prostředí, jako při vysokých teplotách, vlivem mikrovln apod. Takto vyvolané poruchy jsou však na rozdíl od účinků bugu jen přechodné, s náhodnými hodnotami výsledků. Vyžadují fyzické držení výpočetního zařízení útočníkem a pro úspěšnost útoků potřebují přesné načasování. Kromě toho jsou takové útoky dobře proveditelné u *smart*-karet, ale už hůře například u počítačů. Výhodou bug-útoků je to, že nevyžadují fyzický přístup k zařízení během probíhajícího výpočtu a jsou prováděny bez nutnosti manipulace s operačním prostředím. Chyby způsobované bugem jsou deterministické a nastávají vždy, když je prováděn dílčí výpočet. Útočník už při bug-útocích neovlivňuje čas nebo povahu chyby jako takové, volí už jen hodnoty vstupů probíhajících výpočtů.

Smyslem práce bylo teoretické i praktické ověření realizovatelnosti a úspěšnosti aplikace uvažovaných útoků, včetně jejich rozšíření a adaptace na různá výpočetně-implementační specifika.

V konkrétních útocích se zaměřujeme především na asymetrický šifrovací systém RSA, který patří nepochybně mezi jedno z nejrozšířenějších schémat používaných v současné kryptografii. Pro porovnání je uvažováno i principielně podobné, méně známé schéma Pohlig-Hellman, které funguje v okruhu s odlišnými vlastnostmi, teoreticky využitelnými pro snížení složitosti útoků. Předpokládanou metodou usnadňující generování šifrových textů vhodných pro útok je v jeho případě výpočet odmocnin modulo p prvočíslo. Této problematice se věnuje čistě matematická část práce - kapitola 2. Jí předcházející kapitola jen stručně shrnuje znalost v textu používaných definic a fakt z teorie čísel v okruzích a cyklických grupách a z teorie pravděpodobnosti.

Popisu základních uvažovaných šifrovacích schémat a výchozích výpočetních předpokladů (jako je konkretizace druhu chyby v hardwarovém zařízení či varianta a způsob implementovaných výpočetních algoritmů) je věnována kapitola 3. Na základě těchto specifikací jsou pak samotné myšlenky útoků, převzatých z článku [1], uvedeny v kapitole 5 a v následujících kapitolách dále rozpracovávány.

Narozdíl od předpokladu článku [1] se tato práce zaměřuje i na reálný výskyt náhodných chyb během výpočtů na chybovém zařízení, jejichž existence jednoznačně ovlivňuje praktickou úspěšnost při provádění útoků. Jejich pravděpodobnosti vzniku se podrobně věnuje kapitola 4, v kapitole 6 je zkoumán vliv těchto chyb na uvažované útoky a jsou zde popsány metody, kterými lze jejich negativní vliv na úspěšnost útoku do určité míry redukovat. Pro možnost přímého použití převzatých útoků bylo nezbytné provést úpravy některých problematických částí jejich principů, na jejichž rozbor se zaměřuje kapitola 7. V ní je současně ukázána možnost modifikace útoků pro libovolný, nejen (v článku [1]) specificky předpokládaný, způsob implementace uvažovaných algoritmů modulárního mocnění (Left-to-Right, Right-to-Left). Při analýze útoků uvidíme, že vyhodnocování dešifrovaných textů na základě práce s přesnou chybovou hodnotou vhodně poskytuje útoku výpočetní spolehlivost, proto si v této kapitole rovněž ukážeme, že takový postup není nutné omezovat jen na některý z uvedených algoritmů mocnění.

Z praktického hlediska použitelnosti útoků je také nezbytné uvažovat různé optimalizace a přídavná schémata implementovaná současně s kryptografickými schématy a posoudit možnost provedení útoku i v takovém případě. Tomuto aspektu, se zaměřením na schéma RSA, se věnuje kapitola 8, kde je posuzována jak samotná možnost realizace útoků při daných optimalizacích (schématech), tak možnost potřebné modifikace původních

návrhů útoků.

Nedílnou součástí práce bylo vedle teoretické části i praktické odzkoušení veškerých útoků popsaných v textu. Informacemi o provedeném testování se zabývá kapitola 9, ve které je čtenář seznámen jak s důležitými výsledky jednotlivých útoků, tak i se způsobem jejich provedení a dalšími důležitými informacemi týkajícími se tohoto hlediska, jako je například volba modelu testovacího zařízení nebo výběr testovaných parametrů.

Celkovým shrnutím se zabývá závěrečná kapitola, v níž jsou mimo jiné dána bezpečnostní doporučení, která je vhodné zavést jako určitou prevenci proti realizaci útoků využívajících hardwarových výpočetních chyb, včetně zaměření na volbu parametrů procesoru dešifrovacího zařízení.

Kapitola 1

Matematický základ

1.1 Základy pravděpodobnosti

Předpokládáme, že čtenář ovládá základy teorie pravděpodobnosti, a proto zde formální definice *pravděpodobnosti* a dalších pojmů (jako *náhodný jev*, *náhodná veličina*) uvádět nebudeme. Pro naše potřeby víceméně postačí i jen intuitivní znalost těchto pojmů. Uveďme zde jen používané značení a připomeňme základní fakta, která při výpočtech pravděpodobnosti budeme využívat.

Pro operace s jevy budeme používat zápisy: doplněk $\bar{A}$ (nenastane jev A), průnik $A \cap B$ (jevy A a B nastávají současně) a sjednocení $A \cup B$ (nastane alespoň jeden z jevů A, B). Počet jevů budeme předpokládat pouze konečný a kromě používaných základních zákonů jako je komutativita, asociativita a distributivita jevů explicitně připomeňme tzv. *de Morganovy zákony*: $\overline{\bigcap_{i=1}^{n} A_i} = \bigcup_{i=1}^{n} \bar{A}_i$, $\overline{\bigcup_{i=1}^{n} A_i} = \bigcap_{i=1}^{n} \bar{A}_i$.

Pravděpodobnost náhodného jevu A značíme jako $P(A)$. Základními vlastnostmi pravděpodobnosti jsou: $1 \geq P(A) \geq 0$, $P(\bar{A}) = 1 - P(A)$.

Pro sjednocení dvou jevů platí, že $P(A \cup B) = P(A) + P(B) - P(A \cap B)$. Tento vzorec lze indukcí rozšířit pro libovolný konečný počet jevů:

$$P(\bigcup_{i=1}^{n} A_i) =$$

$$\sum_{i=1}^{n} P(A_i) - \sum_{i<j} P(A_i \cap A_j) + \sum_{i<j<k} P(A_i \cap A_j \cap A_k) - \ldots + (-1)^{n+1} P(\bigcap_{i=1}^{n} A_i)$$

Pozorování 1.1.1. *Rovnost* $P(\bigcup_{i=1}^{n} A_i) = \sum_{i=1}^{n} P(A_i)$ *platí, jen pokud jsou jevy po dvou disjunktní (neslučitelné)*, tj. $A_i \cap A_j = \theta$, $i \neq j$. Jinak $P(\bigcup_{i=1}^{n} A_i) < \sum_{i=1}^{n} P(A_i)$, dle výše uvedeného vzorce.

Náhodné jevy A, B se nazývají *nezávislé*, když $P(A \cap B) = P(A) \cdot P(B)$. Náhodné jevy $A_1, A_2, \ldots$ se nazývají *nezávislé*, když pro libovolnou neprázdnou podmnožinu jevů platí $P(\bigcap_{j=1}^{k} A_{i_j}) = \prod_{j=1}^{k} P(A_{i_j})$. Lze dokázat, že doplňky nezávislých jevů jsou nezávislé jevy a platí tvrzení:

Tvrzení 1.1.2. *Jsou-li* $A_1, \ldots, A_n$ *nezávislé náhodné jevy, pak*

$$P(\bigcup_{i=1}^{n} A_i) = 1 - \prod_{i=1}^{n} P(\bar{A}_i).$$

Podmíněná pravděpodobnost (pravděpodobnost jevu A za podmínky, že nastal jev B) je dána vztahem: $P(A|B) = P(A \cap B)/P(B)$, kde $P(B) > 0$.

Pro *systém dvou jevů* $A, \bar{A}$ (tj. existují jen dva možné jevy) platí rovnosti: $P(A|B) + P(\bar{A}|B) = 1$, $P(B) = P(B|A)P(A) + P(B|\bar{A})P(\bar{A})$.

1.2 Elementární teorie čísel

Pojem dělitelnost budeme používat v této práci pouze v kontextu dělitelnosti v okruhu celých čísel $\mathbb{Z}$. Jako $a|b$ značíme, že a je dělitelem čísla b, a jako a div b celočíselný podíl při dělení a hodnotou b. *Kongruence* prvků a a b *modulo* n (značíme $a \equiv b \pmod{n}$) znamená, že a a b dávají stejný zbytek po celočíselném dělení hodnotou n, tj. pokud $n|a-b$. Zápis $\gcd(a, b)$ označuje největšího společného dělitele prvků a a b, který existuje pro každou dvojici $a, b \in \mathbb{Z}$ a při dodržování konvence $\gcd(a, b) \geq 0$ je jednoznačně určen. Pro jeho výpočet bývá používán klasický *Eukleidův algoritmus*, přičemž stačí uvažovat výpočet pro $a, b \geq 0$. Pokud $\gcd(a, b) = 1$, říkáme, že prvky a a b jsou *nesoudělné*. V rámci okruhu $\mathbb{Z}$ navíc platí tzv. *Bezoutova rovnost* (jejíž platnost bývá dokazována právě z průběhu Eukleidova algoritmu), která říká, že pro každou dvojici prvků $a, b \in \mathbb{Z}$ existují prvky $u, v \in \mathbb{Z}$ takové, že $\gcd(a, b) = ua + vb$. Pro přímý výpočet těchto hodnot splňujících Bezoutovu rovnost se používá obsáhlejší varianta algoritmu, tzv. *rozšířený Eukleidův algoritmus*.

Algoritmus 1. *Rozšířený Eukleidův algoritmus* [1]

VSTUP: a, b nezáporná celá čísla, $a > b$
VÝSTUP: $\gcd(a, b)$ a celá čísla u, v splňující $ua + vb = \gcd(a, b)$

1. $(a_1, u_1, v_1) \leftarrow (a, 1, 0)$
   $(a_2, u_2, v_2) \leftarrow (b, 0, 1)$
2. $i \leftarrow 2$
3. While $a_i \neq 0$ do: $\quad /\gcd(a_{i-1}, a_i) = \gcd(a_i, a_{i-1} \bmod a_i)/$
   $i \leftarrow i + 1$
   $a_i \leftarrow a_{i-2} \pmod{a_{i-1}}$
   $u_i \leftarrow u_{i-2} - (a_{i-2} \operatorname{div} a_{i-1})u_{i-1} \quad /a_i = u_i a + v_i b/$
   $v_i \leftarrow v_{i-2} - (a_{i-2} \operatorname{div} a_{i-1})v_{i-1}$
4. Return$(a_{i-1}, u_{i-1}, v_{i-1}) \quad /\gcd(a_{i-1}, 0) = a_{i-1}/$

1.2.1 Komutativní okruhy

Kryptografická schémata, kterým se budeme v práci věnovat, fungují v konečných komutativních okruzích (s jednotkou) $(\mathbb{Z}_n, +, -, \cdot, 0, 1)$, kde $\mathbb{Z}_n = \{0, \ldots, n-1\}$ a operace chápeme modulo n. Připomeňme si některá základní fakta, která v těchto strukturách platí.

Invertibilním prvkem v okruhu R je takový prvek a, pro který existuje prvek b splňující rovnost $a \cdot b = 1$. Tuto (jednoznačně určenou) hodnotu *inverzního prvku* obecně značíme jako a^{-1}. Je-li R komutativní okruh s jednotkou, pak $(R^*, \cdot, ^{-1}, 1)$ je abelovská grupa, kde R^* představuje množinu všech invertibilních prvků okruhu R. Jedná se o tzv. *multiplikativní grupu invertibilních prvků* okruhu R.

Eulerova funkce je definována jako počet čísel z množiny $\{1, \ldots, n-1\}$ nesoudělných s číslem n. Pokud je $n = p_1^{e_1} p_2^{e_2} \ldots p_k^{e_k}$ prvočíselný rozklad čísla n, pak lze hodnotu Eulerovy funkce pro n vypočítat jako

$$\varphi(n) = p_1^{e_1-1} p_2^{e_2-1} \ldots p_k^{e_k-1} \cdot (p_1 - 1)(p_2 - 1) \ldots (p_k - 1).$$

Speciálně, pokud je p prvočíslo, pak $\varphi(p) = p - 1$.

[1] Z důvodu větší přehlednosti algoritmu jsou proměnné indexovány podle jednotlivých kroků cyklu, poznamenejme však, že při implementaci postačí pro výpočet hodnot s indexem i uchovávat v paměti pouze výsledky s indexy $i-1$ a $i-2$.

Tvrzení 1.2.1. *Buď $n \geq 2$ a $n > a > 0$. Pak je ekvivalentní:*
$\gcd(a, n) = 1$, *a je invertibilní prvek v okruhu $\mathbb{Z}_n$.*

Tedy $\mathbb{Z}_n^* = \{0 < k < n : \gcd(n, k) = 1\}$ a speciálně $\mathbb{Z}_p^* = \mathbb{Z}_p - \{0\}$ pro p prvočíslo. Navíc lze díky platnosti tohoto tvrzení k výpočtu inverzních prvků vhodně využít rozšířený Eukleidův algoritmus, a to dle následujícího pozorování.

Pozorování 1.2.2. *Mějme prvek $a \in \mathbb{Z}_n^*$. Pak* $\gcd(a, n) = 1$ *a dle Bezoutovy rovnosti existují $u, v \in \mathbb{Z}$ taková, že platí $ua + vn = 1$. Odtud dostáváme hodnotu inverzního prvku $a^{-1} \equiv u \pmod{n}$.*

Eulerova věta říká, že pokud $a \in \mathbb{Z}_n^*$, pak $a^{\varphi(n)} = 1$. Jako důsledek pak platí, že pokud $r \equiv s \pmod{\varphi(n)}$, pak $a^r = a^s$, tj. exponent lze redukovat modulo $\varphi(n)$. Speciální případ Eulerovy věty pro prvočíslo p ($\varphi(p) = p - 1$) se nazývá *Malá Fermatova věta*.

Klasická *Čínská věta o zbytcích* (často označovaná anglickou zkratkou CRT) říká, že pro $n = n_1 \cdot \ldots \cdot n_k$, kde $n_1, \ldots, n_k$ jsou po dvou nesoudělná přirozená čísla, a pro libovolná celá čísla $u_1, \ldots, u_k$ existuje právě jedno $x \in \{0, \ldots, n - 1\}$, které řeší soustavu kongruencí

$$x \equiv u_1 \pmod{n_1}, \ \ldots, \ x \equiv u_k \pmod{n_k}.$$

V následující kapitole se budeme věnovat problematice existence a hledání řešení rovnic typu $x^r = a$. Pro okruh $\mathbb{Z}_n$, kde n je složené číslo s prvočíselným rozkladem $n = p_1^{e_1} p_2^{e_2} \ldots p_k^{e_k}$, lze díky platnosti CRT redukovat výpočty na okruhy $\mathbb{Z}_{p_i^{e_i}}$ určené činiteli rozkladu čísla n, a poté jen zkombinovat dílčí výsledky do konečného řešení. A protože v této práci budeme vzhledem k výběru kryptografických schémat explicitně pracovat jen v okruzích $\mathbb{Z}_p$ pro p liché prvočíslo a $\mathbb{Z}_n$, kde $n = pq$ je součinem dvou lichých prvočísel, omezme se na řešení těchto rovnic pouze v $\mathbb{Z}_p$. Navíc, díky tomu, že v $\mathbb{Z}_p$ pro každou dvojici prvků $a, b \neq 0$ platí, že $a \cdot b \neq 0$, můžeme vypustit triviální případ $a = 0$, a tedy pro $a \neq 0$ řešit rovnici ekvivalentně v mutliplikativní grupě $\mathbb{Z}_p^*$, o které navíc platí, že je *cyklická*.

Jen poznamenejme, že $\mathbb{Z}_n^*$ obecně cyklickou grupou být nemusí. Platí dokonce, že $\mathbb{Z}_n^*$ je cyklická právě tehdy, když $n = 2, 4, p^e, 2p^e$, kde $e \geq 1$ a p je liché prvočíslo. Platnost teorie o odmocninách tak lze rozšířit na tyto cyklické grupy, avšak nikoliv obecně pro $\mathbb{Z}_n^*$, kde n je libovolné složené číslo.

Některé výchozí definice a základní fakta, která platí v cyklických grupách, zmiňme v následující části.

1.2.2 Cyklické grupy

Pojmem *grupa* mějme na mysli abelovskou (komutativní) grupu. Grupa H je *podgrupou* grupy G právě tehdy, když $H \subseteq G$ a je uzavřená na všechny operace (tj. výsledek operace je opět prvkem H). *Řád grupy* je definován jako počet prvků grupy, *řád prvku* jako řád podgrupy, kterou generuje. To, že a generuje grupu H, označujeme jako $\langle a \rangle = H$. Pokud je grupa generována jedním prvkem, tzv. *generátorem*, nazýváme takovou grupu *cyklickou*.

Tvrzení 1.2.3. *Buď $n \geq 2$ a $n > a > 0$. Pak A je netriviální podgrupa grupy $(\mathbb{Z}_n, +, -, 0)$ právě tehdy, když $A = \{0, d, 2d, \ldots, (k-1)d\}$ pro nějaké $d|n$, kde $1 \leq d < n$ a $n = dk$.*

Důkaz. Buď A netriviální podgrupa $\mathbb{Z}_n$. Zvolme nejmenší $a \in A$, $a > 0$, a buď m nejmenší násobek (tj. součet k sčítanců prvku a) takový, že platí $n \leq m = k \cdot a$. Pokud $m \neq n$, pak $0 < m - n < a$. Protože $m - n \in A$, dostáváme spor s volbou a. Proto platí, že $m = n$, tedy $a|n$, $n = ak$. Pak a generuje k-prvkovou grupu $\langle a \rangle = \{0, a, \ldots, (k-1)a\}$, $\langle a \rangle \subseteq A$. Předpokládejme, že existuje $b \in A$, které neleží v $\langle a \rangle$. V takovém případě $a \not| b$, a tedy existuje i takové, že $ai < b < a(i+1)$. Pak $b - ai \in A$ a přitom $0 < b - ai < a$, což je opět spor. Odtud $\langle a \rangle = A$. Opačná implikace je zřejmá. □

Lze ukázat, že každá konečná cyklická grupa řádu n je izomorfní (cyklické) grupě $(\mathbb{Z}_n, +, -, 0)$. Díky tomu dostáváme následující tvrzení.

Důsledek 1.2.4. *Je-li G cyklická grupa řádu n, pak obsahuje podgrupu řádu d právě tehdy, když $d|n$. A pokud $d|n$, pak existuje právě jedna podgrupa řádu d (tj. o d prvcích) a je cyklická.*

Tím jsme zároveň dokázali speciální případ (pro cyklické grupy) obecně platné *Lagrangeovy věty,* která říká, že je-li H podgrupa konečné grupy G, pak řád grupy H dělí řád G, specielně tedy řád každého prvku dělí velikost celé grupy. O generátorech grup platí následující.

Tvrzení 1.2.5. *Buď $n \geq 2$ a $n > a > 0$. Pak je ekvivalentní:*
$\gcd(a, n) = 1$, *a je generátor cyklické grupy $\mathbb{Z}_n$.*

Jako důsledek tohoto tvrzení platí, že cyklická grupa řádu n má právě $\varphi(n)$ generátorů a tedy rovněž pro každé $d|n$ obsahuje $\varphi(d)$ prvků řádu d. Méně známým důsledkem daného tvrzení je rovnost:

Důsledek 1.2.6. $n = \sum_{d|n} \varphi(d)$

Důkaz. Každý prvek grupy, která je řádu n, generuje některou z jejích podgrup. Z toho vyplývá, že počet n prvků grupy odpovídá součtu množství generátorů přes všechny podgrupy. Každá podgrupa řádu d má podle důsledku předchozího tvrzení právě $\varphi(d)$ generátorů, přičemž $d|n$ podle Důsledku 1.2.4. Tedy $n = \sum_{d|n} \varphi(d)$. □

Uvažujme nadále cyklickou grupu v multiplikativní notaci $(G, \cdot, ^{-1}, 1)$. Řád grupy pak můžeme ekvivalentně definovat jako nejmenší kladné r takové, že $a^r = 1$ pro libovolný prvek grupy, a řád prvku a jako nejmenší kladné n takové, že $a^n = 1$. Generátor α cyklické grupy řádu n pak někdy nazýváme *n-tou primitivní odmocninou z* 1, neboť $\alpha^n = 1$ a $\alpha^m \neq 1$ pro všechna $0 < m < n$.

Definice 1. Buď G konečná cyklická grupa řádu n. Mějme α generátor grupy G a libovolný prvek $a \in G$. Jako *diskrétní logaritmus* prvku a *při základě* α nazýváme jednoznačně určené celé číslo x, $0 \leq x \leq n-1$, takové, že $a = \alpha^x$, tj. $x = \log_\alpha a$.

Úloha nalezení takového x při daných G, α, a je známa jako (*zobecněný*) *problém diskrétního logaritmu*, o kterém se předpokládá, že je *těžký*, tj. není znám způsob, jak jej algoritmicky řešit v tzv. polynomiálním čase alespoň pro nezanedbatelnou část všech možných vstupů, jsou-li parametry problému pečlivě vybírány s ohledem na znalost jeho snadno-řešitelných instancí. Existuje více výpočetních problémů, které jsou obecně považovány za *těžké*.[2] Důkazy pro to známy nejsou, ale neví se o efektivních algoritmech, které by v obecném případě dokázaly dané *těžké* úlohy řešit v reálném čase. Snaha o jejich vyřešení v konkrétních případech pak často bývá realizována výpočtem *hrubou silou*, tj. snahou odzkoušení všech možností.

Lemma 1.2.7. *Je-li a prvek řádu n, pak a^s je řádu $n/\gcd(s,n)$.*

Důkaz. Mějme prvek a řádu n, tj. $a^n = 1$. Řád prvku a^s odpovídá nejmenšímu x takovému, že $(a^s)^x = 1$, a podle řádu prvku a musí platit, že n dělí sx. A protože x je nejmenší možné, odpovídá sx nejmenšímu společnému násobku čísel s, n. Rozmysleme si, že je-li c nejmenším společným násobkem prvků e a f, platí, že $c \cdot \gcd(e,f) = e \cdot f$. Odtud $sx \cdot \gcd(s,n) = s \cdot n$, tedy $x = \frac{n}{\gcd(s,n)}$. □

[2] Pro podrobnější studium doporučujeme literaturu [5].

Kapitola 2

Výpočet odmocnin v $\mathbb{Z}_p^*$

V této kapitole se budeme věnovat řešitelnosti a hledání řešení rovnic typu $x^r = a$ v $\mathbb{Z}_p^*$, kde označením p mějme jednotně v celé kapitole na mysli liché prvočíslo. Uvažujme přitom jen kladné $r < p-1$, neboť v opačném případě lze exponent vždy redukovat modulo $\varphi(p) = p-1$ dle Malé Fermatovy věty se zachováním shodného řešení rovnice. Připomeňme, že multiplikativní grupa $\mathbb{Z}_p^*$ je cyklická vzhledem k násobení modulo p a je řádu $p-1$.

Kapitola je rozdělena do čtyř hlavních částí. V první části jsou předložena a dokázána základní tvrzení o existenci a vlastnostech r-tých residuí v $\mathbb{Z}_p^*$ a popsána idea rozložení problému výpočtu jejich r-té odmocniny při libovolném r na dílčí výpočty prvočíselných odmocnin. Dále zde stručně diskutujeme problém aplikace známých algoritmů pro výpočet odmocnin z hlediska použitelnosti při velkých exponentech a jako nejvhodnější je zvolen a popsán tzv. AMM algoritmus, u něhož uvádíme i jeho odvození.

Druhá část rozebírá problematiku výpočtu q^u-tých odmocnin metodou rekurzivně počítaných prvočíselných q-tých odmocnin, pro jejíž diskusi je zde definován pojem *rekurzivní posloupnosti q-tých odmocnin* a řešena její existence v $\mathbb{Z}_p^*$ na základě modelu datové struktury *stromu*. Pro obtížnější případ $q^u \nmid p-1$ uvádíme alternativní deterministický způsob přímého výpočtu.

V následující části pak dokazujeme korektnost rozložení problému hledání r-tých odmocnin na výpočty odmocnin prvočíselných, resp. dokazujeme řešitelnost dílčích rovnic. Nakonec se ještě krátce zmíníme o problému faktorizace, který s předpokládanými postupy výpočtů odmocnin souvisí.

Speciálnímu případu druhých odmocnin se stručně věnuje závěr kapitoly. Na základě odvozených vlastností kvadratických residuí jsou podrobněji ana-

lyzovány možnosti výpočtů druhých (a 2^u-tých) odmocnin a uvádíme alternativní postup pro třídu prvočísel $p \equiv 3 \pmod 4$ (a $p \equiv 5 \pmod 8$).

2.1 Residua a jejich odmocniny

Definice 2. Prvek $a \in \mathbb{Z}_p^*$ nazýváme *r-té residuum* (modulo p), pokud má rovnice $x^r = a$ řešení. V opačném případě a nazýváme *r-té neresiduum*. Speciálně pro $r = 2$ hovoříme o tzv. *kvadratických* (ne)residuích.

Řešení rovnice $x^r = a$ nazýváme *r-tou odmocninou* prvku a a označujeme jej obecně jako $a^{1/r}$.

Tvrzení 2.1.1. *Buď G cyklická grupa řádu n a nechť $\gcd(r, n) = 1$. Mějme libovolné $a \in G$. Pak má rovnice $x^r = a$ v dané grupě právě jedno řešení, a to prvek a^e, kde $re \equiv 1 \pmod n$.*

Konkrétně, pokud $\gcd(r, p-1) = 1$, pak každý prvek $a \in \mathbb{Z}_p^*$ je r-tým residuem a má právě jednu r-tou odmocninu.

Důkaz. Prvek r je invertibilní díky předpokladu nesoudělnosti s řádem grupy, tj. $re \equiv 1 \pmod n$ pro určité e. Mějme prvek b takový, že splňuje rovnost $a = b^r$. Úpravou této rovnosti dostáváme $a^e = (b^r)^e = b$, existuje tedy právě jedno řešení rovnice, a to hodnoty a^e, $(a^e)^r = a$. □

Tvrzení 2.1.2. *Mějme cyklickou grupu G řádu n a kladné celé číslo $r \neq 1$ takové, že $r|n$. Je-li rovnice $x^r = a$ pro dané $a \in G$ řešitelná, pak existuje celkem r různých řešení. Navíc, je-li α primitivní r-tá odmocnina z 1 a b nějaké řešení dané rovnice, pak všechna řešení lze vyjádřit jako $\{b, b\alpha, \ldots, b\alpha^{r-1}\}$, speciálně pro $r = 2$ jsou řešení $\{b, -b\}$.*

Konkrétně, pokud $r|p-1$, pak každé r-té residuum $a \in \mathbb{Z}_p^*$ má r různých r-tých odmocnin.

Důkaz. Mějme nějaké řešení b rovnice $x^r = a$ a prvek α, který je r-tou primitivní odmocninu z 1. Poznamenejme, že existence takového prvku α v grupě vyplývá z předpokladu, že $r|n$. Je zřejmé, že $(b\alpha^i)^r = b^r(\alpha^r)^i = a$ pro všechna $1 \leq i \leq r$, přičemž $b\alpha^i \in G$. Dokažme, že řešení jsou vzájemně různá.

Pro spor naopak předpokládejme, že existují $1 \leq i < j \leq r$ taková, že $b\alpha^j = b\alpha^i$, tj. z vlastnosti krácení v grupě, že $\alpha^j = \alpha^i$. Pak platí $\alpha^j\alpha^{r-i} = \alpha^i\alpha^{r-i} = \alpha^r = 1$. Dostáváme $1 = \alpha^r\alpha^{j-i} = \alpha^{j-i}$, $0 < j - i < r$, což je spor. Tedy $b\alpha^i \neq b\alpha^j$ pro všechna $i \neq j$. Každé řešení rovnice $x^r = a$ je kořenem

polynomu $x^r - a$, přičemž polynom stupně r má nejvýše r kořenů, tj. nalezli jsme všechna řešení dané rovnice. Pro $r = 2$ je $\alpha = -1$. □

Důsledkem tvrzení tak je, že pokud $r|p-1$ a a je r-té residuum, náhodně zvolený prvek ze $\mathbb{Z}_p^*$ bude r-tou odmocninou a s pravděpodobností $\frac{r}{p-1}$.

Dokažme nyní formuli, podle níž lze o libovolném prvku ze $\mathbb{Z}_p^*$ rozhodnout, zda je r-tým residuem (pro libovolné $1 < r < p-1$).

Tvrzení 2.1.3. *Prvek $a \in \mathbb{Z}_p^*$ je r-té residuum právě tehdy, když platí $a^{\frac{p-1}{\delta}} = 1$, kde $\delta = \gcd(r, p-1)$. To je navíc ekvivalentní s tím, že máme-li α generátor $\mathbb{Z}_p^*$, pak $a = \alpha^i$ pro nějaké i takové, že $\delta|i$.*

Důkaz. Pokud $\delta = \gcd(r, p-1) = 1$, pak každé $a \in \mathbb{Z}_p^*$ je r-té residuum podle Tvrzení 2.1.1, přičemž rovnost $a^{p-1} = 1$ platí vždy. Předpokládejme proto dále $\delta \neq 1$.

Nechť a je r-té residuum, tj. existuje b takové, že $a = b^r$. Protože $\delta|r$, dostáváme $a^{\frac{p-1}{\delta}} = (b^r)^{\frac{p-1}{\delta}} = b^{\frac{r}{\delta}(p-1)} = (b^{p-1})^{\frac{r}{\delta}} = 1$.

Naopak předpokládejme, že platí $a^{\frac{p-1}{\delta}} = 1$. Buď α generátor $\mathbb{Z}_p^*$, pak lze prvek a vyjádřit jako $a = \alpha^i$ pro nějaké i. Řád prvku a je pak podle Lemmatu 1.2.7 hodnoty $\frac{p-1}{\gcd(i,p-1)}$ a zároveň podle předpokládané rovnosti dělí $\frac{p-1}{\delta}$. Odtud dostáváme, že δ nutně dělí $\gcd(i, p-1)$, a tedy $\delta|i$. Připomeňme, že $\delta = \gcd(r, p-1)$, a proto z platnosti Bezoutovy rovnosti dostáváme, že existuje e takové, že $re \equiv \delta \pmod{p-1}$. Pak $(\alpha^{e\frac{i}{\delta}})^r = \alpha^{\delta\frac{i}{\delta}} = \alpha^i = a$. □

Pozorování 2.1.4. *Nechť $a \in \mathbb{Z}_p^*$ je r-té residuum a d nějaký dělitel čísla r, pak a je rovněž d-té residuum.*

Nakonec odvoďme, kolik r-tých residuí při daném r v $\mathbb{Z}_p^*$ vlastně existuje.

Tvrzení 2.1.5. *Počet r-tých residuí v $\mathbb{Z}_p^*$ je $\frac{p-1}{\delta}$, kde $\delta = \gcd(r, p-1)$.*

Důkaz. Buď $d = \frac{p-1}{\delta}$, kde $\delta = \gcd(r, p-1)$. Podle Tvrzení 2.1.3 jsou r-tá residua právě všechny prvky splňující rovnost $x^d = 1$. Tu splňují všechny prvky d-tého řádu a řádů d' dělících d. Poznamenejme, že $d|p-1$ a předpoklad prvků takových řádů je tedy v souladu s Lagrangeovou větou korektní. Přičemž prvků daného řádu d' je podle důsledku Tvrzení 1.2.5 právě $\varphi(d')$ a podle Důsledku 1.2.6 téhož tvrzení platí, že $\sum_{d'|d} \varphi(d') = d$. □

Důsledek 2.1.6. *Náhodně zvolený prvek $a \in \mathbb{Z}_p^*$ je r-tým residuem s pravděpodobností $\frac{1}{\delta}$.*

V případě, že $\gcd(r, p-1) = 1$, lze hodnotu r-té odmocniny prvku vypočítat dle Tvrzení 2.1.1. Je však otázkou, jak vypočítat r-té odmocniny prvku pro libovolné r.

Předpokládejme, že umíme vypočítat libovolnou prvočíselnou q-tou odmocninu prvku takovou, že $q|p-1$, a nechť $\delta = \gcd(r, p-1)$ má prvočíselný rozklad $\delta = q_1^{v_1} \dots q_k^{v_k}$, kde $v_i \geq 1$ pro všechna $1 \leq i \leq k$. Pak číslo r lze vyjádřit jako $r = q_1^{u_1} \dots q_k^{u_k} t$, kde t je taková hodnota, že $\gcd(q_i, t) = 1$ pro všechna $1 \leq i \leq k$, a platí, že $u_i \geq v_i$ a $\gcd(t, p-1) = 1$.

Výpočet r-té odmocniny lze pak rozložit na řešení soustavy rovnic s exponenty určenými dílčími činiteli uvedeného rozkladu čísla r, tj. potřebujeme postupně nalézt řešení rovnic $x_0^t = a$, $x_1^{q_1^{u_1}} = x_0$, $\dots$, $x_k^{q_k^{u_k}} = x_{k-1}$. Pak $x_k^r = a$. První rovnici lze díky nesoudělnosti t s $p-1$ řešit metodou mocnění dle Tvrzení 2.1.1. Ostatní lze zřejmě řešit jako vždy u_i-krát rekurzivně opakovaný výpočet q_i-tých odmocnin, neboť $x^{q^u} = (x^{q^{u-1}})^q = a$ (podrobněji se možností tohoto postupu budeme zabývat v části 2.2). Z uvedeného principu vidíme, že pro nalezení libovolné r-té odmocniny je postačující mít speciální algoritmus na výpočet prvočíselných q-tých odmocnin pro případy $q|p-1$.

Takový algoritmus odvodíme v následující části, v dalších bude diskutován podrobněji způsob výpočtu q^u-tých odmocnin v závislosti na hodnotě u a poté dokázána řešitelnost dílčích kongruencí ve výše popsaném postupu.

Protože řešení rovnice $x^r = a$ je kořenem polynomu $x^r - a$ a každý prvek $a \in \mathbb{Z}_p$ je kořenem polynomu $x^p - x$, jinou metodou testování, zda má daná rovnice řešení, může být zkoumání (ne)soudělnosti polynomu $x^r - a$ s polynomem $x^p - x$ v $\mathbb{Z}_p[x]$. Polynom $x^r - a$ nesoudělný s $x^p - x$ nemá žádný kořen ze $\mathbb{Z}_p$, naopak případné existující kořeny polynomu $x^r - a$, tedy r-té odmocniny prvku a, lze zjistit faktorizací polynomu $\gcd(x^p - x, x^r - a)$ na lineární činitele. Tento postup je zvolen v případě verze algoritmu Tonelli-Shanks [4]. Metoda hledání odmocnin na základě práce s polynomy je však při velkých hodnotách exponentů p, r v obecném případě nevyužitelná, a to z důvodu příliš velké výpočetní složitosti.

2.1.1 AMM algoritmus

Pro výpočet q-tých odmocnin v $\mathbb{Z}_p^*$, kde $q|p-1$, zde uvedeme algoritmus navržený v článku [2] autory Adleman-Manders-Miller (dále v textu budeme algoritmus označovat jako AMM alg.). Přesnou podobu algoritmu popišme pro $q > 2$. Pro jednodušší případ $q = 2$ zmiňme jen nutné úpravy, které je

třeba v rámci algoritmu provést, jeho konkrétní podobu lze nalézt v uvedeném článku.

AMM algoritmus vychází z následujících dvou pomocných tvrzení. V jejich popisu pracujme s lichými prvočísly p, q takovými, že $q|p-1$. Hodnota $v \geq 1$ nechť je největší celé číslo takové, že $q^v|p-1$ (v odpovídá definici tzv. *valuace*). Pak

$$p - 1 = q^v(qm + m') \qquad (*)$$

pro nějaká celá čísla m, m', $q > m' \geq 1$. Navíc zřejmě $\gcd(q, qm + m') = 1$.

Lemma 2.1.7. *Buď $p - 1 = q^v(qm + m')$ s vlastnostmi dle $(*)$ a mějme $h = q^{-1} \pmod{qm + m'}$. Pokud $a^{qm+m'} = 1$, pak a^h je q-tou odmocninou a.*

Důkaz. Prvek a leží v podgrupě řádu $qm+m'$. Protože $\gcd(q, qm+m') = 1$, plyne zbytek z Tvrzení 2.1.1. □

Lemma 2.1.8. *Buď $p - 1 = q^v(qm + m')$ s vlastnostmi dle $(*)$ a nechť g je q-té neresiduum. Mějme $a \in \mathbb{Z}_p^*$ takové, že platí $a^{q^j(qm+m')} = 1$ pro nějaké $1 \leq j < v$ a j buď nejmenší takové možné. Pak pro nějaké celé číslo λ, $0 < \lambda < q$, platí*

$$(ag^{q^{v-j}\lambda})^{q^{j-1}(qm+m')} = 1.$$

Důkaz. Pro zjednodušení zápisu položme $\bar{a} = a^{q^{j-1}(qm+m')}$. Hledáme řešení λ rovnosti $a^{q^{j-1}(qm+m')} \cdot g^{q^{v-1}\lambda(qm+m')} = 1$, tj. vyjádření inverzního prvku $(\bar{a})^{-1} \pmod{p}$ jako specifickou mocninu prvku $\bar{g} = g^{q^{v-1}(qm+m')} = g^{(p-1)/q}$. Dokažme existenci takového řešení.

Prvek g je dle předpokladu q-té neresiduum, přičemž $\gcd(q, p-1) = q$, tudíž podle Tvrzení 2.1.3 platí, že $g^{\frac{p-1}{q}} \neq 1$. Protože $(g^{\frac{p-1}{q}})^q = g^{p-1} = 1$, řád prvku $\bar{g} \neq 1$ dělí prvočíslo q, tudíž $\bar{g}$ je řádu q. Dosazením lze snadno ověřit, že $(\bar{a})^{-1} = \bar{a}^{(q-1)}$ a $\bar{a}^{(q-1)q} = 1$, tudíž analogicky jako v případě prvku $\bar{g}$ lze odvodit, že prvek $(\bar{a})^{-1} \neq 1$ je rovněž řádu q. Oba prvky $\bar{g}$, $(\bar{a})^{-1}$ jsou tak generátorem q-prvkové podgrupy, připomeňme, že jedinečné v $\mathbb{Z}_p^*$ (viz. Důsledek 1.2.4), existuje tedy nějaké kladné $\lambda < q$ takové, že $(\bar{a})^{-1} = \bar{g}^{\lambda}$. □

Lemma 2.1.8 tak ukazuje možnost, jak pomocí libovolného q-tého neresidua odvodit z prvku a prvek ležící v podgrupě, jejíž řád je alespoň o q-násobek nižší než je řád grupy, z níž prvek a pochází. Při opakované aplikaci daného Lemmatu se tedy budeme pohybovat v řetězci podgrup $G_{1(qm+m')} \subset G_{q(qm+m')} \subset \ldots \subset G_{q^v(qm+m')}$.

Algoritmus na základě těchto pozorování funguje tak, že na počátku je nalezen nejmenší exponent j_1, pro který $a^{q^{j_1}(qm+m')} = 1$. Pokud $j_1 = 0$, algoritmus nalezne hodnotu q-té odmocniny aplikací Lemmatu 2.1.7, v opačném případě je za použití Lemmatu 2.1.8 vypočtena nová hodnota a_2, pro kterou $a_2^{q^{j_2}(qm+m')} = 1$, kde j_2 je opět nejmenší takový exponent a přitom platí, že $j_2 < j_1$. Algoritmus takto pokračuje, až v nějakém kroce pro nějaké a_i platí $j_i = 0$ a může být použito Lemma 2.1.7 k výpočtu q-té odmocniny b_i prvku a_i. Z ní je nakonec odvozena hodnota q-té odmocniny prvku a. Konkrétní podoba algoritmu pak vypadá následovně.

Algoritmus 2. *AMM algoritmus*

VSTUP: $a \in \mathbb{Z}_p^*$, p, q lichá prvočísla, $q|p-1$ $\quad /a_0 = a/$
VÝSTUP: nějaké řešení $x^q = a$ (existuje-li)

1. If $a^{\frac{p-1}{q}} \neq 1$ then Return(a není q-té residuum) /ověření q-residuity/
2. Zvolme $g \in \mathbb{Z}_p^*$ takové, že $g^{\frac{p-1}{q}} \neq 1$ $\quad /g$ je q-té neresiduum/
   $\bar{g} \leftarrow g^{\frac{p-1}{q}}$
3. Vypočtěme v, m, m' taková, že $p - 1 = q^v(qm + m')$ $\quad /q^{v+1} \not| p-1/$
4. $s \leftarrow 1$
5. Nalezněme nejmenší j takové, že $a^{q^j(qm+m')} = 1$
6. While $j \neq 0$ do:
   (i) $\bar{a} \leftarrow a^{q^{j-1}(qm+m')}$
   Nalezněme řešení λ rovnice $\bar{a} \cdot \bar{g}^{\lambda} = 1$
   (ii) $a \leftarrow a \cdot g^{q^{v-j}\cdot\lambda}$ $\quad /a \in G_{q^{j-1}(qm+m')}$ dle Lemmatu 2.1.8/
   $s \leftarrow s \cdot g^{q^{v-j-1}\cdot\lambda}$ $\quad /a = a_0 \cdot s^q/$
   (iii) Nalezněme nejmenší j takové, že $a^{q^j(qm+m')} = 1$
7. $/j = 0 : a \in G_{1(qm+m')}/$
   $h \leftarrow q^{-1} \pmod{qm + m'}$
   $b \leftarrow a^h$ $\quad /b^q = a = a_0 \cdot s^q/$
8. Vypočtěme $s^{-1} \pmod p$ $\quad$ /použitím rozš. Eukleidova alg./
9. Return($b \cdot s^{-1}$) $\quad /(bs^{-1})^q = a/$

Pro hledání neresiduí není znám deterministický algoritmus. Nicméně, není-li prvočíslo q příliš velké, lze nalezení q-tého neresidua g v 2. kroce

provádět opakováním náhodné volby - dle Důsledku 2.1.6 je pravděpodobnost, že náhodně zvolené $g \in \mathbb{Z}_p^*$ bude q-té neresiduum, rovna hodnotě $1/q$.

Složitost nalezení potřebné hodnoty λ v kroce 6(i) odpovídá složitosti řešení úlohy diskrétního logaritmu, která je v obecném případě považována za *těžkou*. Nicméně v algoritmu je tato úloha řešena v rámci q-prvkové grupy, tudíž při nepříliš velkém q připadá v úvahu výpočet *hrubou silou*.

Tyto dva body výpočtu určují, že algoritmus funguje s vhodnou složitostí za předpokladu, že q není příliš velké.

Poznamenejme, že algoritmus vrací jen jedno řešení rovnice $x^q = a$. Podle Tvrzení 2.1.2 lze však z výstupní hodnoty b dopočítat všechna řešení jako $\{b, b\alpha, \ldots, b\alpha^{q-1}\}$, kde jako α, hodnotu q-té primitivní odmocniny z 1, lze použitím q-tého neresidua g, voleného v 2. kroce algoritmu, volit $\alpha = g^{\frac{p-1}{q}}$ (viz. důkaz Lemmatu 2.1.8).

Poznámka 1. Pro potřeby výpočtu druhé odmocniny prvku, tedy $q = 2$, stačí v algoritmu (resp. příslušných Lemmatech) jednoduše nahradit: $q \to 2$, $m' \to 1$, $h \to m{+}1$ a $\lambda \to 1$. Volba hodnoty λ plyne z vlastnosti hodnoty kvadratické neresiduity $g^{\frac{p-1}{2}} = -1$, čímž se celkově snižuje výpočetní složitost algoritmu vůči složitosti při prvočíselném lichém q. Obě možná řešení podle Tvrzení 2.1.2 získáme jako $b, -b$.

2.2 Výpočet q^u-tých odmocnin

Stále uvažujme rozklad $p - 1 = q^v(qm + m')$ dle $(*)$ z předchozí části, tj. q je libovolné prvočíslo takové, že $q|p-1$, a v největší celé číslo takové, že $q^v|p-1$. Podívejme se blíže na způsob, jakým vypočítat q^u-tou odmocninu prvku a při daném u.

Nabízí se přirozená myšlenka, že základní AMM algoritmus pro prvočíselné odmocniny aplikujeme u-krát za sebou. Je však zřejmé, že v takovém případě všechny postupně zvolené hodnoty odmocnin musejí být q-tými residui, aby bylo možné výpočet q-tých odmocnin skutečně u-krát rekurzivně opakovat. Zaveďme si pro tento jev definici.

Definice 3. Mějme libovolné celé číslo $1 < n < p - 1$. Pak posloupnost $a_0 = a \to a_1 = (a_0)^{1/n} \to a_2 = (a_1)^{1/n} \to \ldots \to a_k = (a_{k-1})^{1/n}$ prvků ze $\mathbb{Z}_p^*$, kde všechna a_i pro $0 \leq i < k$ jsou nutně n-tými residui, nazvěme *rekurzivní posloupností n-tých residuí* stupně k. Triviální případ a definujme jako posloupnost stupně 0.

Je-li a q^u-té residuum, pak je zřejmé, že taková rekurzivní posloupnost q-tých residuí stupně u existuje a složitost nalezení nějaké q^u-té odmocniny metodou rekurzivních výpočtů tedy bude odpovídat složitosti nalezení takové posloupnosti.

Na pomoc si vezměme strukturu z teorie grafů, tzv. *strom*. Budeme se snažit používat jen víceméně intuitivní pojmy, formální definice týkající se této struktury zde proto zavádět nebudeme, čtenáře případně odkazujeme na nějakou odbornou literaturu teorie grafů.

Strom je specifický graf s propojenými prvky, *uzly*. Uvažujme graf orientovaný a uspořádaný tak, že vrcholem grafu je jeden uzel a další prvky leží na jednotlivých nižších hladinách. Uzly stromu nechť odpovídají prvkům $\mathbb{Z}_p^*$ a je-li uzel propojen s nějakými uzly na hladině o 1 stupeň níže, znamená to, že tyto podřazené prvky, tzv. *synové*, jsou jeho q-tou odmocninou a daný prvek je tedy q-tým residuem. Není-li uzel stromu spojen s žádným uzlem na nižší hladině, je q-tým neresiduem. Protože uvažujeme případ, kdy $q|p-1$, má-li uzel stromu nějaké syny, má jich podle Tvrzení 2.1.2 právě q, tj. jedná se o model tzv. *úplného q-árního* stromu.

Představme si ve vrcholu stromu prvek a, který je q^u-tým residuem. Hledáme-li jeho q^u-té odmocniny, pak stačí uvažovat strom hloubky u, což je strom, který má vrchol a dalších u úrovní. Vrchol de facto odpovídá nulté úrovni. Prvky z i-té úrovně stromu splňují rovnost $x^{q^i} = a$ a je jednoduché si rozmyslet, že pokud je q-ární strom *plný*, tj. obsahuje při své hloubce maximální počet uzlů, pak v každé úrovni i leží q^i uzlů.

Pokud platí, že $q^u|p-1$, tj. $u \leq v$, potom v každé i-té úrovni, $i \leq u$, leží dle Tvrzení 2.1.2 vždy q^i různých prvků, tj. strom musí být plný a všechny jeho cesty jsou délky u, tj. vedou až na nejnižší u-tou úroveň. Tyto cesty odpovídají rekurzivním q-árním posloupnostem stupně u. Výpočet nějaké q^u-té odmocniny prvku rekurzivním výpočtem q-tých odmocnin si lze tedy v tomto případě představit jako pouze sestupný pohyb definovaným stromem - v každém uzlu je ověřena q-residuita prvku, která splněna je, poté je vypočtena nějaká hodnota q-té odmocniny daného prvku a proveden sestup na další úroveň, do příslušného uzlu.

V obecném případě však tvrzení, že strom je plný, neplatí a použití rekurzivních výpočtů q-tých odmocnin by tak k úspěšnému nalezení nějaké q^u-té odmocniny obecně vyžadovalo prohledávání výše definovaného stromu, hledání cesty délky u. Pro případ $q^u \not| p-1$, tj. $u > v$, je proto vhodnější použít odlišný postup výpočtu, a to dle následujícího tvrzení. Uvědomme

si přitom, že je-li b řešením $x^{q^u} = a$, pak danou rovnost splňují i všechna $\{b, b\beta, \ldots, b\beta^{q^v-1}\}$, kde β je q^v-tá primitivní odmocnina z 1.

Tvrzení 2.2.1. *Mějme $p-1 = q^v(qm+m')$ s vlastnostmi dle $(*)$. Nechť $a \in \mathbb{Z}_p^*$ je q^u-té residuum, kde $u > v$, a mějme inverzní prvek $h = (q^u)^{-1} \pmod{qm+m'}$. Pak a^h je q^u-tá odmocnina prvku a.*

Důkaz. Uvědomme si, že se jedná jen o jinak formulované rozšíření Lemmatu 2.1.7. Prvek a je dle předpokladu q^u-tým residuem, tudíž dle Tvrzení 2.1.3 splňuje $a^{\frac{p-1}{\delta}} = 1$, kde $\delta = \gcd(q^u, p-1) = q^v$, což tedy neznamená nic jiného, než $a^{qm+m'} = 1$. Platí, že $\gcd(q^u, qm+m') = \gcd(q, qm+m') = 1$ a zbytek plyne z Tvrzení 2.1.1. □

2.3 Výpočet r-tých odmocnin

Připomeňme si myšlenku výpočtu r-té odmocniny prvku a při libovolném r s využitím výpočtů prvočíselných q-tých odmocnin. Má-li $\delta = \gcd(r, p-1)$ prvočíselný rozklad $\delta = q_1^{v_1} \ldots q_k^{v_k}$, kde $v_i \geq 1$ pro všechna $1 \leq i \leq k$, pak

$$r = q_1^{u_1} \ldots q_k^{u_k} t,$$

přičemž $\gcd(t, p-1) = 1$ a $u_i \geq v_i$ pro všechna $1 \leq i \leq k$. Zapíšeme-li rozklad r bez ohledu na pořadí činitelů obecně jako $r = r_0 \ldots r_k$, pak potřebujeme postupně odvodit řešení rovnic $x_0^{r_0} = a$, $x_1^{r_1} = x_0$, $\ldots$, $x_k^{r_k} = x_{k-1}$. Tento rozklad r uvažujme v celém tomto oddíle.

Lemma 2.3.1. *Nechť $r = r_0 \ldots r_k$ je výše definovaný rozklad a r_i buď nějaký činitel rozkladu takový, že $r_i | p-1$. Je-li $a \in \mathbb{Z}_p^*$ r-té residuum, pak libovolné řešení rovnice $x^{r_i} = a$ je (r/r_i)-tým residuem.*

Důkaz. Je-li a dle předpokladu r-té residuum, pak podle Pozorování 2.1.4 je rovněž r_i-tým residuem, tudíž existuje řešení b rovnice $x^{r_i} = a$. Jestliže $\gcd(r, p-1) = \delta$, pak při uvažovaném specifickém rozkladu platí, že $r_i | \delta$ a že $\gcd(r/r_i, p-1) = \delta/r_i$. Platí, že $b^{\frac{p-1}{\delta/r_i}} = (b^{r_i})^{\frac{p-1}{\delta}} = a^{\frac{p-1}{\delta}} = 1$, přičemž poslední rovnost vyplývá z r-residuity prvku a. Dle Tvrzení 2.1.3 tak dostáváme, že b je (r/r_i)-tým residuem. □

Lemma 2.3.2. *Nechť $r = r_0 \ldots r_k$ je výše definovaný rozklad a r_i buď nějaký činitel rozkladu takový, že $r_i \nmid p-1$. Je-li $a \in \mathbb{Z}_p^*$ r-té residuum, pak a^f je (r/r_i)-tým residuem pro libovolné $1 \leq f < p-1$.*

Důkaz. Buď $\delta = \gcd(r, p-1)$. Z r-residuity prvku a plyne rovnost $a^{\frac{p-1}{\delta}} = 1$. Předpokládejme nejdříve, že $\gcd(r_i, p-1) = 1$. Potom $\gcd(r/r_i, p-1) = \delta$ a $(a^f)^{\frac{p-1}{\delta}} = (a^{\frac{p-1}{\delta}})^f = 1$. Z Tvrz. 2.1.3 tak plyne, že a^f je (r/r_i)-té residuum.

Předpokládejme $\gcd(r_i, p-1) \neq 1$, označme příslušnou hodnotu jako $\bar{r}_i$. Při uvažovaném specifickém rozkladu platí, že $\gcd(r/r_i, p-1) = \delta/\bar{r}_i$, přičemž $(a^f)^{\frac{p-1}{\delta/\bar{r}_i}} = (a^f)^{\frac{p-1}{\delta}\bar{r}_i} = 1$, tedy opět se jedná o (r/r_i)-té residuum. □

Zvolme způsob hledání řešení dílčích rovnic $x^{r_i} = x_{r_{i-1}}$ dle vlastností r_i:

1. Pokud $\gcd(r_i, p-1) = 1$, použijme Tvrzení 2.1.1.
2. Pokud r_i dělí $p-1$, tj. je tvaru $q_i^{u_i}$, kde $u_i = v_i$, řešení nalezneme rekurzivním, v_i-krát opakovaným výpočtem nějaké q_i-té odmocniny (použijme například AMM algoritmus).
3. Pokud r_i nedělí $p-1$, tj. je tvaru $q_i^{u_i}$, kde $u_i > v_i$, použijme Tvrzení 2.2.1.

Dokažme, že při uvedeném způsobu řešení jsou (nezávisle na pořadí zpracovávaných činitelů r_i) všechny jednotlivé rovnice řešitelné a bez nutnosti „návratů“ tak výpočet vede k nalezení r-té odmocniny z a, respektive, že řešení každé dílčí rovnice má potřebné residuální vlastnosti k tomu, aby následující libovolně zvolená rovnice byla řešitelná.

Důkaz. Hledáme řešení rovnice $x^r = a$, kde a je r-té residuum a $r = r_0 \dots r_k$ buď uvažovaný rozklad v libovolném pořadí jeho činitelů.

Vezměme si první činitel, r_0. Je-li rovnice $x^{r_0} = a$ řešitelná, musí platit, že a je r_0-té residuum. Splnění této podmínky plyne z r-residuity prvku a a Pozorování 2.1.4. Dle vlastnosti r_0 zvolíme způsob řešení rovnice $x^{r_0} = a$ a nalezneme nějaké řešení x_0. Přičemž, pokud $r_0 | p-1$, pak x_0 je (r/r_0)-tým residuem dle Lemmatu 2.3.1, v opačném případě platí, že $x_0 = a^f$ pro určité f a x_0 je (r/r_0)-tým residuem dle Lemmatu 2.3.2, tudíž rovnice $x^{r/r_0} = x_0$, kterou zbývá dopočítat, je řešitelná.

Indukcí tak lze analogicky ukázat pro všechna zbylá r_i, $i = 1, \dots, k$, že při vstupu x_{i-1} (které je $(r/r_0 \dots r_{i-1})$-tým residuem) nalezneme řešení x_i rovnice $x^{r_i} = x_{i-1}$, které je pak $(r/r_0 \dots r_i)$-tým residuem. Nakonec získané x_k splňuje $x_k^r = a$ a ukázali jsme, že všechny potřebné dílčí kongruence jsou řešitelné a nalezení konečného výsledku lze provádět bez jakéhokoliv návratu ve výpočtu. Úspěšnost způsobu řešení dle bodu 2. (pro $r_i | p-1$) byla řešena v předchozí části 2.2. □

2.3.1 Problém faktorizace

Ačkoliv jsme u navrženého výpočtu libovolných r-tých odmocnin dokázali řešitelnost dílčích rovnic s prvočíselnými exponenty, je zřejmé, že důležitým aspektem pro možnost realizace takovéhoto postupu není jen zmíněná velikost dílčích prvočíselných činitelů, která pro úspěšné používání AMM algoritmu nesmí být příliš velká, ale rozhodující je velkou částí primárně složitost rozkladu $\gcd(r, p-1)$ na prvočíselné činitele, tj. složitost *faktorizace*. Faktorizace je však obecně považována za *těžký problém*. Nejedná-li se o čísla určitých specifických vlastností, není pro velká čísla obecně znám algoritmus takový, který by dokázal při současných výpočetních možnostech faktorizovat libovolně velké číslo v reálném čase.

S problémem faktorizace souvisí rovněž problém výpočtů odmocnin v $\mathbb{Z}_n$. Jak jsme již uvedli v části 1.2.1, výpočet odmocnin v $\mathbb{Z}_n$ lze při známé faktorizaci čísla n snadno provést použitím CRT. Pokud však faktorizaci čísla n neznáme, nelze takový postup zvolit a jiný obecně znám není. Pokud je problém faktorizace těžký, pak je (pro téměř všechna residua) *těžký* i výpočet odmocnin modulo složené číslo.

2.4 Výpočet druhých odmocnin

Krátce se ještě na závěr zastavme u vlastností kvadratických residuí a alternativního způsobu výpočtu druhých odmocnin pro některé třídy prvočísel.

Podle Tvrzení 2.1.3 platí, že pokud je α generátor $\mathbb{Z}_p^*$, pak $a \in \mathbb{Z}_p^*$ je kvadratickým residuem právě tehdy, když $a = \alpha^i$, kde i je sudé, resp. pokud $a^{\frac{p-1}{2}} = 1$. Navíc platí:

Tvrzení 2.4.1. *Prvek $a \in \mathbb{Z}_p^*$ je kvadratickým neresiduem právě tehdy, když $a^{\frac{p-1}{2}} = -1$.*

Důkaz. Platí, že $a^{p-1} = 1$ pro všechny prvky $\mathbb{Z}_p^*$ a druhá odmocnina z 1 je ± 1. Přičemž $a^{\frac{p-1}{2}} = 1$ právě tehdy, když a je kvadratické residuum. □

Pro zjednodušení zápisů zaveďme značení množiny všech kvadratických residuí jako Q, množinu všech kvadratických neresiduí označujme jako $\bar{Q}$.

Dosazením do výrazu $a^{\frac{p-1}{2}}$ lze jednoduše dokázat, že $-1 \in Q$ pro $p \equiv 1 \pmod 4$ a naopak $-1 \in \bar{Q}$ pro $p \equiv 3 \pmod 4$. Toho využijeme v důkazu následujícího tvrzení.

Tvrzení 2.4.2. *Mějme $a \in \mathbb{Z}_p^*$ kvadratické residuum. Platí, že je-li $p \equiv 1 \pmod 4$, pak druhé odmocniny prvku a jsou buď obě kvadratickými residui, nebo obě kvadratickými neresidui. Je-li $p \equiv 3 \pmod 4$, pak jedna druhá odmocnina prvku a je kvadratickým residuem a druhá kvadratickým neresiduem.*

Důkaz. Podle Tvrzení 2.1.2 jsou druhé odmocniny kvadratického residua a prvky $\pm b$, přičemž $b \neq -b$. Buď g generátor $\mathbb{Z}_p^*$. Pak lze prvky vyjádřit jako $b = g^k$, $-b = g^l$ pro nějaká $k \neq l$ a dostáváme rovnost $b \cdot (-b) = -b^2 = g^{k+l}$. Z ní plyne, že $-b^2 \in Q$ právě když $k+l \pmod{p-1}$ je sudé, přičemž zároveň je zřejmé, že $-b^2 \in Q$ právě když $-1 \in Q$.

Pokud $p \equiv 1 \pmod 4$, pak $-1 \in Q$, tudíž $-b^2$ je kvadratické residuum a $k + l$ tedy musí být sudé. Odtud vyplývá, že obě druhé odmocniny b, $-b$, jsou kvadratickými residui, nebo jsou obě kvadratickými neresidui.

Analogicky pro $p \equiv 3 \pmod 4$, kde $-1 \in \bar{Q}$, dostáváme, že $-b^2$ je kvadratické neresiduum, $k + l$ musí být liché a odtud plyne zbytek. □

Důsledek 2.4.3. *V případě $p \equiv 3 \pmod 4$ při zvolení libovolného kvadratického residua a existuje rekurzivní posloupnost* kvadratických residuí libovolného stupně.

Dokázali jsme tak, že v případě $p \equiv 3 \pmod 4$ je možné 2^u-té odmocniny prvku (pro zcela libovolné u) počítat z libovolného kvadratického residua a je možné zvolit přímou metodu rekurzivních výpočtů druhých odmocnin, neboť vždy právě jedna hodnota je kvadratickým residuem a ta jednoznačně určuje pokračování rekurzivní posloupnosti kvadratických residuí. Pro prvočísla splňující $p \equiv 3 \pmod 4$ je však charakteristická ještě jedna vlastnost. Tou je existence alternativního způsobu výpočtu druhých odmocnin, metodou mocnění dle následujícího pozorování.

Pozorování 2.4.4. *Je-li $p \equiv 3 \pmod 4$, pak druhou odmocninu pro libovolné kvadratické residuum $a \in \mathbb{Z}_p^*$ lze vypočítat jako $a^{\frac{p+1}{4}}$ (resp. $-a^{\frac{p+1}{4}}$).*

Důkaz. Prvek a je dle předpokladu kvadratické residuum, proto existuje b takové, že $b^2 = a$. Díky řádu grupy platí, že $a^{p-1} = 1$, tudíž $a^{p+1} = a^{p-1} \cdot a^2 = a^2 = b^4$. A protože $4|p+1$, dostáváme z této rovnosti vyjádření $b = a^{\frac{p+1}{4}}$. □

Tento způsob výpočtu lze rozšířit ještě na prvočísla $p \equiv 5 \pmod 8$, kdy lze druhou odmocninu prvku a vyjádřit jako $a^{\frac{p+3}{8}}$, nicméně nelze tuto metodu použít obecně při libovolném prvočísle - konkrétně pro $p \equiv 1 \pmod 8$.

Pro takový případ je třeba použít nějaký speciální algoritmus pro výpočet druhých odmocnin, například variantu AMM algoritmu pro $q = 2$ (viz. část 2.1.1, Poznámka 1), další použitelné algoritmy lze nalézt například v literatuře [3] či [5]. Výpočet 2^u-tých odmocnin je pro třídu prvočísel $p \equiv 1 \pmod 4$ vhodné realizovat shodně jako v obecném případě q^u-tých odmocnin (viz. část 2.2).

Kapitola 3

Výchozí schémata a výpočetní algoritmy

Tato kapitola se věnuje popisu základních algoritmů a stanovení důležitých předpokladů týkajících se fungování výpočetního zařízení. Tato východiska jsou zásadní pro konkrétní návrh útoků.

V první části specifikujeme vlastnosti, které budeme u atakovaného chybového dešifrovacího zařízení očekávat. V rámci popisu obecných předpokladů týkajících se realizace násobení zvolíme konkrétní model výpočetní chyby, bugu, a určíme značení, které budeme pro odlišení chybných výpočtů dále v textu používat. Dále zde nalezneme popis algoritmů modulárního mocnění, o kterých předpokládáme, že mohou být v daném chybovém zařízení implementovány. Jedná se o tzv. algoritmy *repeated square-and-multiply*, konkrétně jejich dvě nejzákladnější používané varianty - algoritmy *Left-to-Right* a *Right-to-Left*. Následující oddíl se pak stručně zabývá současnými hranicemi výpočetních možností a posuzuje realizovatelnost hledání hodnot se specificky požadovanými vlastnostmi na základě očekávané složitosti. Tuto představu o současné výpočetní síle využijeme pro hodnocení složitosti útoků při specifikovaných parametrech procesoru.

V poslední části nalezneme popisy kryptografických schémat, na něž budeme útoky aplikovat. Prvním je asymetrické schéma RSA, které patří nepochybně mezi jedno z nejrozšířenějších schémat používaných v současné kryptografii. Druhým uvažovaným algoritmem je méně známé schéma Pohlig-Hellman, které svou formou velmi připomíná předchozí šifru, ale díky fungování v okruhu s odlišnými vlastnostmi se nejedná o asymetrické schéma. Zato umožňuje provedení některých útoků s menší složitostí.

3.1 Předpoklady

Počítačová aritmetika

Pro adaptaci útoku je důležitá znalost výpočetních specifik (mikro)procesoru dešifrovacího zařízení. Předpokládejme, že pracuje klasicky s *binární* (dvojkovou) soustavou a nezápornými celými čísly, resp. používá binární reprezentaci bezznaménkových celých čísel (*unsigned integers*). V celém textu budeme používat zápis, kdy řadíme bity zleva doprava od nejvyšších bitů k nejnižším (odpovídá architektuře *big-endian*). Binární reprezentaci prvku $d = \sum_{i=0}^{\log n} d_i \cdot 2^i$, $d_i \in \{0, 1\}$, kde $d < n$, tedy značíme jako $d = d_{\log n} \ldots d_1 d_0$. Bity indexujeme od nuly z důvodu vhodné korespondence bitu d_i a mocniny 2^i v zápisu hodnoty prvku, zjednodušeným výrazem $\log n$ máme na mysli $\lfloor \log_2 n \rfloor$ (celou dolní část hodnoty). Bitová délka $|n| = \log n + 1$ odpovídá maximální potřebné délce pro zápis čísel v rozsahu 0 až $n - 1$.

Významným parametrem (mikro)procesoru je tzv. (*bitová*) *délka slova*, která udává maximální počet bitů, které je možné zpracovat během jedné instrukce.[1] V textu ji budeme označovat parametrem w. Všechny aritmetické operace na velkých číslech, pro jejichž binární reprezentaci nepostačuje jen délka jednoho procesorového slova (tj. hodnoty vyšší než $2^w - 1$, neboli $x = \sum_{i=0}^{s-1} x_i \cdot (2^w)^i$, $x_i \in \{0, \ldots, 2^w - 1\}$, kde $s > 1$), musejí být rozděleny do několika kroků a naprogramovány s využitím aritmetických operací na jednotlivých slovech, z nichž se hodnoty operandů skládají (pro dělení na slova používejme zápis $x = (x_{s-1} \ldots x_1 x_0)_{2^w}$). Uvažujeme-li w-bitovou délku procesorových slov, pak se $|n|$-bitová hodnota skládá z $s = \lceil |n|/w \rceil$ slov.

Specifikace hardwarové chyby

Pokud jde o specifikaci výpočetního bugu, který budeme v chybovém zařízení předpokládat, budeme uvažovat modelovou situaci, že procesor provádí chybně součin nějaké dvojice procesorových slov (a, b), kterou budeme nazývat *chybovou dvojicí*. V celém textu budeme pro jednoduchost předpokládat existenci právě jedné takové dvojice (a, b).

V rámci víceslovní aritmetiky bývá konkrétně násobení velkých čísel nejjednodušeji realizováno tzv. metodou *školního násobení*, kdy součin dlouhých čísel x a y, každé reprezentované s slovy délky w, tj. $x = (x_{s-1} \ldots x_1 x_0)_{2^w}$, $y = (y_{s-1} \ldots y_1 y_0)_{2^w}$, bude prováděno tak, že každé z s slov hodnoty x musí být násobeno s každým slovem hodnoty y a příslušná slova dílčích součinů

[1] Typická délka se obecně pohybuje od 4 do 128 bitů, dnes nejčastěji 8, 16, 32 či 64.

pak budou sečtena potřebným způsobem tak, aby dala konečný součin $x \cdot y$. Pokud x obsahuje na nějaké pozici slovo a a y obsahuje slovo b, tj. $x_i = a$ a $y_j = b$ pro nějaká $0 \leq i, j < s$ (někdy budeme stručně zapisovat jako $a \in x$, $y \in b$), je při výpočtu součinu $x \cdot y$ procesorem vypočten chybný dílčí součin $x_i \cdot y_j$, a tak je chybný i součin výsledný (nepředpokládáme, že by se jednotlivě vzniklé chyby navzájem vyrušily).

Dvojice (a, b), (b, a) obecně považujeme za dvě různé, proto případná chybnost výsledku součinu dvou hodnot závisí na způsobu implemetace, zda je v procesoru prováděna dílčí operace $a \cdot b$, či $b \cdot a$. Předpokládané implementační pořadí bude v textu odpovídat pořadí zápisu výrazů (s prioritou závorek), a tak při zápisu $x \cdot y$ předpokládáme provádění dílčích součinů na slovech skutečně v pořadí $x_i \cdot y_j$. Je zřejmé, že tato informace musí být útočníkovi pro realizaci útoku známa, neboť mění nutnost rozmístění chybových slov mezi jednotlivé hodnoty.

Chybu vzniklou při výpočtu nazvěme chybou *očekávanou*, volíme-li záměrně vstupní hodnoty operandů tak, aby v určitém kroce vyhodnocování počítaného výrazu došlo na bugovém zařízení k násobení chybových slov, a tím k vzniku chybného výstupu. Je však zřejmé, že v případě součinu více jak dvou hodnot může dojít k chybě na jiném než očekávaném místě, pokud vzájemně násobené hodnoty mezivýsledků obsahují chybová slova. Ačkoliv je v obou případech výsledek nesprávný, původ chyby (a zřejmě i hodnota výstupu) se vzájemně liší. Takto vzniklou chybu nazvěme *náhodnou*.

Při zápisu operací na chybovém procesoru budeme typicky používat označení $\langle\ \rangle$. Protože se však z hlediska praktického používání předpokládá, že k chybným výpočtům i na chybovém procesoru dochází zřídka, pro většinu (náhodných) vstupů je očekáván korektní výsledek. V takovém případě budeme i výpočty na chybovém procesoru značit běžným způsobem a označení typu $\langle x \cdot y \rangle$, $x^{\langle r \rangle}$ bude použito jen pro zdůraznění vzniku součinové chyby při vyhodnocování příslušného výrazu (součinu, mocniny). Do jisté míry tak sice splývá označení pro správnost výsledku a prostředek realizace, smysl však bude dostatečně vyplývat z kontextu (případně jej explicitně upřesníme).

3.1.1 Algoritmy pro modulární mocnění

V práci předpokládáme dva různé algoritmy pro modulární mocnění, které mohou být implementované ve výpočetním zařízení. Jedná se o algoritmy *Left-to-Right* a *Right-to-Left* (dále jen LTOR, RTOL). Jejich názvy vystihují, v jakém pořadí jsou zpracovávány jednotlivé bity exponentu.

LTOR algoritmus uveďme ve dvou možných implementačních variantách, pro jejich odlišení je podle způsobu realizace označujme dále v textu jako *If*-verze a *If-else* verze. Pro LTOR i RTOL algoritmy předpokládáme:

VSTUP:	n kladné celé číslo
	$C, d \in \mathbb{Z}_n$, kde $d = d_{\log n} \dots d_1 d_0$ (v binární reprezentaci)
VÝSTUP:	$C^d \pmod n$

Algoritmus 3. *LTOR (Left-to-Right) algoritmus*

$z \leftarrow 1$	$z \leftarrow 1$
For $k = \log n$ down to 0 do:	For $k = \log n$ down to 0 do:
$z \leftarrow z^2 \pmod n$	If $d_k = 1$ then $z \leftarrow \mathbf{z^2C} \pmod n$
If $d_k = 1$ then $z \leftarrow zC \pmod n$	else $z \leftarrow z^2 \pmod n$
Return(z)	Return(z)

Algoritmus 4. *RTOL (Right-to-Left) algoritmus*

$z \leftarrow 1$, $y \leftarrow C$
For $k = 0$ to $\log n$ do:
If $d_k = 1$ then $z \leftarrow zy \pmod n$
$y \leftarrow y^2 \pmod n$
Return(z)

$$/C^d = C^{\sum_{i=0}^{\log n} d_i 2^i} = \prod_{i=0}^{\log n} (C^{2^i})^{d_i}/$$

If-else podoba LTOR algoritmu, bez ohledu na vhodnost implementace, je obecnější (*If*-verze je jejím speciálním případem) a lze ji provést celkem šesti různými způsoby, dle pořadí, v jakém může být vyhodnocován (tučně zvýrazněný) výraz z^2C, počítaný při jedničkové hodnotě bitu. Vzhledem k nesymetričnosti chybových dvojic bude v navrhovaných útocích záviset na této implementační podobě, lze si ale rozmyslet (zřejmě snáze na konkrétních příkladech útoků), že implementace jsou po dvojicích vzájemně symetrické, v tom smyslu, že stačí uvažovat jen tři možné varianty $z(zC)$, $(zC)z$, $(z^2)C$ a pro zbývající varianty lze nutné modifikace veškerých útoků popsaných dále v textu vyjádřit jen změnou v požadavcích na volby šifrových textů, záměnou slova a za b a naopak. Stejně tak v případě RTOL algoritmu stačí uvažovat provádění součinu zy jen v tomto pořadí, uvidíme ale, že uváděný návrh RTOL-útoku je dokonce na pořadí tohoto součinu nezávislý.

Výrazem *i-tý cyklus* mějme vždy na mysli jednu iteraci *For*-cyklu pro $k = i$. Bez ohledu na implementační podobu algoritmu platí následující.

Pozorování 3.1.1. *V i-tém cyklu LTOR algoritmu je vstupní hodnotou proměnné z hodnota odpovídající* $C^{d'} \pmod n$, *kde* $d' = \sum_{k=i+1}^{\log n} 2^{k-(i+1)} d_k$, *v i-tém cyklu RTOL je vstupní hodnotou proměnné y hodnota* $C^{2^i} \pmod n$, *proměnné z hodnota odpovídající* $C^{\bar{d}} \pmod n$, *kde* $\bar{d} = \sum_{k=0}^{i-1} 2^k d_k$.

3.1.2 Posuzování výpočetní složitosti

Má-li pravděpodobnost určitého jevu A hodnotu $P(A) = 2^{-v}$, znamená to, že stačí daný náhodný nezávislý pokus opakovat řádově 2^v-krát, abychom obdrželi jeden takový výsledek, který daný jev splňuje. Na tento odhad složitosti se budeme v textu odvolávat jako na složitost určenou pravděpodobností $P(A)$.

Z hlediska útoku *hrubou silou* (což je útok typicky používaný u symetrických šifer, spočívající v odzkoušení všech možných kombinací bitů klíče) je v dnešní době podle současných výpočetních možností považováno za výpočetně snadné testování méně než 2^{64} možností (a tedy z hlediska bezpečnosti nedostačující), za dostatečně bezpečné se dnes považuje množství kolem 2^{80} až 2^{100} a za principielně nezlomitelný (odzkoušením metodou hrubé síly) systém při nutnosti provedení více jak 2^{128} různých testů.

Podle těchto kryptograficky uznávaných hranic budeme v tomto textu posuzovat současné možnosti výpočetní síly - tedy, co v uvažovaných útocích můžeme považovat za *výpočetně realizovatelné* a co již za *výpočetně nemožné*.

3.2 Kryptografická schémata

V útocích se zaměříme na schéma RSA a Pohlig-Hellman, v této části uvedeme jejich stručný popis. Používáme přitom obvyklé značení M pro zprávu (*plaintext*) a C pro zašifrovanou zprávu (šifrový text - *ciphertext*).

3.2.1 RSA

Schéma s veřejným klíčem (*asymetrické* schéma) je systém založený na principu, že každá entita vlastní veřejný šifrovací klíč e a jemu odpovídající tajný soukromý dešifrovací klíč d. Díky tomu kdokoliv může šifrovat zprávu

určenou dané entitě, avšak jen entita samotná, tj. vlastník soukromého klíče, je schopen zprávu dešifrovat, neboť bezpečnost takového systému je obecně založena na předpokladu, že výpočet hodnoty d z veřejně známé hodnoty e je *těžký*, tj. v reálném čase neproveditelný. RSA (Rivest-Shamir-Adleman) je takovým kryptosystémem, v současné době zřejmě jedním z nejrozšířenějších. Jeho bezpečnost se opírá o *těžký* problém faktorizace. V případě nalezení obecně vhodného algoritmu pro provádění efektivní faktorizace v reálném čase by byl systém RSA prolomen.

Generování klíčů a princip fungování RSA:

1. Vygenerujme dvě náhodná (a vzájemně různá) velká prvočísla p a q, obě zhruba stejné délky.
2. Vypočtěme veřejný *modul* $n = pq$ a Eulerovo číslo $\varphi(n) = (p-1)(q-1)$.
3. Vyberme náhodné celé číslo e, $1 < e < \varphi(n)$ tak, aby $\gcd(e, \varphi(n)) = 1$.
   e - *šifrovací exponent*
4. Nalezněme jednoznačně určené celé číslo d, $1 < d < \varphi(n)$, takové, že $ed \equiv 1 \mod \varphi(n)$ (použitím rozšířeného Eukleidova algoritmu).
   d - *dešifrovací exponent*

Dostáváme tak: **veřejný klíč** (n, e), **privátní klíč** d.

Předpokládá se, že šifrovaná zpráva M je nejvýše nějaké maximální (bitové) délky. Zprávy delší než stanovené maximum musejí být rozděleny do bloků a takto vzniklé části pak mohou být šifrovány buď nezávisle na sobě, či pro zajištění ochrany proti manipulaci s jednotlivými bloky za použití nějakého dalšího bezpečnostního schématu. V případě RSA je třeba zprávu reprezentovat jako celé číslo M z intervalu $[0, n-1]$.

Šifrování: $C = M^e \pmod{n}$
Dešifrování: $M = C^d \pmod{n}$

RSA-operace tedy provádíme v okruhu $\mathbb{Z}_n$. Z bezpečnostního hlediska je doporučováno dodržovat některé základní vlastnosti volených parametrů:

Doporučená velikost modulu a výběr prvočísel:

Vzhledem k progresi faktorizačních algoritmů je doporučeno používat modul n délky alespoň 1024 bitů. Prvočísla p, q je přitom třeba volit tak, aby faktorizace čísla $n = pq$ byla předpokládaně *těžká*, proto:

- Prvočísla by měla být zhruba stejné bitové délky a dostatečně velká.
- Rozdíl $p - q$ by neměl být příliš malý.
- Někdy bývá navíc požadováno, aby prvočísla byla tzv. *silná*, v praxi se však ukazuje, že silná prvočísla neposkytují výrazně vyšší ochranu než prvočísla náhodně volená.

Volba veřejného exponentu:

Pro zrychlení procesu šifrování je z důvodu používání *repeated square-and-multiply* algoritmů (pro modulární mocnění) výhodné volit exponent e malý nebo s malým počtem jedničkových bitů v jeho binární reprezentaci, což eliminuje počet nutných operací (viz. popisy algoritmů v části 3.1.1). Exponent $e = 3$ je však náchylný k některým útokům. V praxi se proto často používá exponent (šifrovací klíč) $e = 2^{16} + 1 = 65537$, který má rovněž jen dva jedničkové bity ve své binární reprezentaci.

Poznámka. Uvědomme si, že právě díky způsobu volby klíčů platí jednoznačnost určení M, resp. C. Tj. pokud $(M')^e = C$, pak $M' = C^d = M$, analogicky $C' = C$ pro $(C')^d = M$ (plyne z Tvrzení 2.1.1) a nejnižší bit klíče e i d je vždy 1. Tyto vlastnosti platí shodně i pro níže uvedené schéma Pohlig-Hellman.

3.2.2 Pohlig-Hellman

Šifrovací schéma Pohlig-Hellman vypadá na první pohled velmi podobně jako systém RSA, nelze však v jeho případě hovořit o asymetrickém šifrování. Používá:

- veřejný prvočíselný modul p
- tajné exponenty pro šifrování e a dešifrování d,
  splňující: $\gcd(e, p-1) = 1$, $e \cdot d \equiv 1 \pmod{p-1}$
- **šifrování:** $C = M^e \pmod{p}$, **dešifrování:** $M = C^d \pmod{p}$

Funguje tedy v okruhu $\mathbb{Z}_p$. Podstatný rozdíl spočívá právě v použití veřejného prvočíselného modulu, což způsobuje, že oba exponenty musejí být tajné, tj. musejí zůstat sdíleným tajemstvím mezi komunikujícími stranami jako je tomu v případě symetrických šifer. Známe-li totiž jeden z exponentů, jsme schopni jednoduše dopočítat druhý, a to použitím rozšířeného Eukleidova algoritmu, díky nesoudělnosti exponentů se známou hodnotou $p - 1$.

Kapitola 4

Pravděpodobnostní analýza

Některé složitosti nalezení šifrových textů vhodných k útoku (zejména pro RSA) budou určeny pravděpodobnostmi výskytu jednoho či dvou daných chybových slov v náhodně volené hodnotě, tj. pravděpodobnostmi $P(a)$, $P(a \cap b)$. Na základě těchto pravděpodobností pak lze složitost a realizovatelnost generování textů metodou náhodných voleb posuzovat dle části 3.1.2. Článek [1] uvádí hodnoty $P(a) = 2^{-w} \lceil |n|/w \rceil$, $P(a \cap b) = (2^{-w} \lceil |n|/w \rceil)^2$, které lze však chápat jen jako hrubé aproximace. Článek uvažuje jen délky procesorových slov $w = 32, 64$, pro které je odchylka od skutečné hodnoty víceméně zanedbatelná, pravděpodobnosti obou jevů jsou však reálně nižší. V první části kapitoly odvodíme jejich přesné hodnoty.

Tyto pravděpodobnosti však v útocích neurčují jen složitost nalezení vhodných textů, avšak zároveň ovlivňují i to, s jakou pravděpodobností může dojít při výpočtech k náhodné součinové chybě - $P(a)P(b)$ pro chybnost výpočtu xy, $P(a \cap b)$ pro x^2. Článek tuto možnost z důvodu předpokládaných (pro parametry $w = 32, 64$) malých hodnot $P(a)$, $P(a \cap b)$ zanedbává. Uvažování vzniku náhodných chyb přitom dává představu o praktické realizovatelnosti útoku i o pravděpodobnosti odhalení bugu, a tedy vůbec reálnosti používání chybového zařízení v praxi. Odhadu *chybovosti* (pravděpodobnosti vzniku součinové chyby) uvažovaného bugového zařízení se budeme věnovat v druhé části kapitoly. Na základě rozboru možných implementací odhadneme pravděpodobnosti vzniku chyb během jednoho cyklu LTOR, resp. RTOL algoritmu a odvodíme vzorec pro odhad celkové pravděpodobnosti vzniku náhodných chyb v rámci zpracování celé délky exponentu, tj. chybovosti odpovídající jedné operaci dešifrování. Výsledky použijeme při posuzování vlivu náhodných chyb na úspěšnost útoků v kapitole 6.

4.1 Výskyt chybových slov

Tvrzení 4.1.1. *Uvažujme w-bitovou délku procesorových slov a náhodnou hodnotu x o délce s slov, $s \geq 1$. Označme jako $P(a)$ pravděpodobnost, že x obsahuje slovo a, tj. že alespoň jedno ze slov hodnoty x nabývá w-bitové hodnoty a. Pak*

$$P(a) = 1 - \left(\frac{2^w - 1}{2^w}\right)^s.$$

Důkaz. Pravděpodobnost, že se na i-té pozici bude vyskytovat právě slovo a, označme jako $P(a^i)$. Je zřejmé, že pro w-bitové slovo platí $P(a^i) = \frac{1}{2^w}$. Nechť a označuje shodně jev, že se slovo a vyskytuje na alespoň jedné pozici. Ačkoliv $P(a) = P(\bigcup_{i=0}^{s-1} a^i)$, neplatí $P(a) = \sum_{i=0}^{s-1} P(a^i)$, neboť tato rovnost nastává jen pro jevy po dvou neslučitelné, což jevy a^i, a^j, $i \neq j$, zjevně nesplňují, a dle Poznámky 1.1.1 tak hodnota $P(a)$ musí být menší než $\sum_{i=0}^{s-1} P(a^i) = s \cdot \frac{1}{2^w}$. Pro výpočet lze použít obecně platný vzorec pravděpodobnosti sjednocení jevů, jednoduším způsobem je však výpočet přes doplňkový jev. Je-li $\bar{a}$ jev, že se slovo a nevyskytuje na žádné pozici, pak z předpokladu nezávislosti jednotlivých pozic $P(\bar{a}) = P(\bigcap_{i=0}^{s-1} \bar{a}^i) = \left(\frac{2^w-1}{2^w}\right)^s$ a z platnosti $P(a) = 1 - P(\bar{a})$ plyne daná rovnost. □

Tvrzení 4.1.2. *Uvažujme w-bitovou délku procesorových slov a náhodnou hodnotu x o délce s slov, $s \geq 2$. Označme jako $P(a \cap b)$ pravděpodobnost, že x obsahuje slova a i b. Platí, že*

$$P(a \cap b) = 1 - \frac{2(2^w - 1)^s - (2^w - 2)^s}{2^{ws}}.$$

Důkaz. Vztah lze odvodit ze vzorce pro pravděpodobnost sjednocení a de Morganových vzorců jako $P(a \cap b) = P(a) + P(b) - P(\overline{\bar{a} \cap \bar{b}}) = P(a) + P(b) - 1 + P(\bar{a} \cap \bar{b})$, přičemž $P(\bar{a} \cap \bar{b})$ odpovídá pravděpodobnosti, že slovo a ani b se nevyskytuje na žádné pozici, tedy $P(\bar{a} \cap \bar{b}) = (\frac{2^w-2}{2^w})^s$, a je zřejmé, že $P(a) = P(b) = 1 - \left(\frac{2^w-1}{2^w}\right)^s$. Dostáváme tak dokazovanou rovnost. □

4.2 Míra chybovosti zařízení

Je zřejmé, že míra náhodné chybovosti bugového zařízení nebude při výpočtu (tj. v průběhu operace dešifrování) ovlivněna jen samotnou pravděpodobností

výskytu chybových slov v operandech (která je obecně důsledkem předpokládané délky operandů, délky procesorových slov a počtu chybových dvojic), ale závisí rovněž na používaném algoritmu modulárního mocnění a teoreticky je ovlivněna i konkrétním způsobem jeho implementace. Těmto aspektům se budeme věnovat v následující části. Zde jen uveďme, že známe-li pravděpodobnost $P(e)$ chybovosti zařízení v rámci jednoho cyklu algoritmu, můžeme celkovou chybovost (pro jeden výpočet modulární mocniny v závislosti na délce exponentu) odhadnout dle následujícího pozorování.

Pozorování 4.2.1. *Uvažujme náhodný exponent délky* $\log n + 1$ *a buď* $P(e)$ *pravděpodobnost součinové chyby v rámci jednoho cyklu použitého algoritmu pro modulární mocnění. Jednotlivé cykly pro jednoduchost považujme za nezávislé. Pravděpodobnost vzniku náhodné chyby v rámci celého průběhu algoritmu, tj. při zpracování exponentu v celé jeho délce*[1]*, je zhruba*

$$P = 1 - (1 - P(e))^{\log n}.$$

Důkaz. Nechť e_j označuje jev, kdy v j-tém cyklu algoritmu dojde k výpočetní chybě. Z předpokladu nezávislosti cyklů s použitím Tvrzení 1.1.2 dostáváme $P = P(\bigcup_{i=1}^{\log n} e_i) = 1 - \prod_{i=1}^{\log n} P(\overline{e}_i) = 1 - P(\overline{e})^{\log n} = 1 - (1 - P(e))^{\log n}$. □

Nyní se podrobněji podíváme na hodnotu pravděpodobnosti $P(e)$.

4.2.1 Vliv implementace

Připomeňme si hlavní ideu bugu, který uvažujeme v chybovém zařízení. Předpokládáme, že existuje jedna chybová dvojice slov (a, b) a chybný výpočet vzniká, násobíme-li spolu dvě hodnoty x a y, v pořadí $x \cdot y$, a přitom $a \in x$, $b \in y$. Základem odhadu pravděpodobnosti vzniku náhodné chyby je proto rozbor jednotlivě prováděných kroků (součinů) v rámci jednoho cyklu algoritmu při uvažovaném pořadí vyhodnocování (tj. při uvažované implementaci algoritmu) a zkoumání možných výskytů chybových slov v jednotlivých hodnotách, které by způsobovaly chybný součin. U LTOR alg. rozlišujeme různé varianty implementace podle způsobu vyhodnocování výrazu z^2C a rozmysleme si, že i zde můžeme použít rozdělení na vzájemně ekvivalentní implementace $\{(z^2)C, C(z^2)\}$, $\{z(zC), (Cz)z\}$, $\{z(Cz), (zC)z\}$ (v souladu s dvojicemi, na které je používán útok se symetricky volenými texty), kdy v rámci dvojic pravděpodobnost vzniku náhodných chyb zůstává stejná. Jde totiž

[1] Eliminujeme počáteční cyklus, ve kterém typicky k vzniku chyby nedochází.

o to, že chyby sice vznikají při obsahu odlišných slov v proměnných, jde však pouze o záměnu slov a za b a naopak a protože $P(a) = P(b)$, jsou výsledné pravděpodobnosti shodné. Stačí tak opět uvažovat tři základní varianty, analogicky u RTOL alg. jen jednu z možných dvou.

Pravděpodobnosti odvozujme obecně pro libovolný j-tý cyklus algoritmu (zpracovávající bit d_j) a hodnoty proměnných v rámci cyklu přitom považujme za náhodné a nezávislé, stejně tak hodnotu jejich součinu a rovněž samotnou hodnotu bitu d_j. Reálně tyto odhady zřejmě neodpovídají triviálním případům jako je první prováděná iterace ($z = 1$) nebo speciální případ druhé iterace v LTOR ($z = C$).

Analýza LTOR algoritmu

Pozorování 4.2.2. *Uvažujme výše uvedené předpoklady. Pak pravděpodobnost $P(e)$, že při náhodném vstupu C v libovolném cyklu LTOR algoritmu dojde k součinové chybě, je při daných implementacích dána hodnotami:*

$z(zC): \quad P(e) = 1/2 \cdot [P(a)^2(2 - P(a)) + P(a \cap b)]$
$(zC)z: \quad P(e) = 1/2 \cdot [P(a)^2(2 - P(a \cap b)) + P(a \cap b)]$
$(z^2)C: \quad P(e) = 1/2 \cdot [P(a \cap b)(2 - P(a)^2) + P(a)^2]$

Důkaz. V rámci důkazu zapisujme jev, že hodnota x obsahuje slovo a, jako a_x. Pravděpodobnost, že nastane nějaká součinová chyba, označme jako $P(e)$. Uvažujme první implementační případ a j-tý cyklus algoritmu. Pokud bit $d_j = 1$, pak je počítán výraz $z(zC)$, a tedy

$$\begin{aligned} P(e|d_j = 1) &= P((a_z \cap b_C) \cup (a_z \cap b_{zC})) \\ &= P(a_z \cap b_C) + P(a_z \cap b_{zC}) - P(a_z \cap b_C \cap b_{zC}) \\ &= P(a)\{P(b)\} + P(a)P(b) - P(a)\{P(b)\}P(b), \end{aligned}$$

přitom $P(a) = P(b)$. Pokud $d_j = 0$, počítán je pouze výraz $z \cdot z$, a tak $P(e|d_j = 0) = P(a_z \cap b_z) = P(a \cap b)$. Předpokládáme-li, že bit d_j nabývá jedničkové či nulové hodnoty s pravděpodobností $1/2$, pak celková pravděpodobnost vzniku chyby v rámci cyklu je (dle věty o úplné pravd.)
$P(e) = \frac{1}{2}P(e|d_j = 1) + \frac{1}{2}P(e|d_j = 0) = \frac{1}{2}(P(a)^2(2 - P(a)) + P(a \cap b))$.

Analogicky lze odvodit pravděpodobnost pro druhý případ

$$\begin{aligned} P(e|d_j = 1) &= P((a_z \cap b_C) \cup (a_{zC} \cap b_z)) \\ &= P(a)\{P(b)\} + P(a)P(b) - P(a \cap b)\{P(b)\}P(a), \end{aligned}$$

rovnost $P(e|d_j = 0) = P(a \cap b)$ platí vždy. Pro poslední možnost dostáváme:

$$\begin{aligned} P(e|d_j = 1) &= P((a_z \cap b_z) \cup (a_{z^2} \cap b_C)) \\ &= P(a \cap b) + P(a)\{P(b)\} - P(a \cap b)P(a)\{P(b)\}. \end{aligned}$$

□

LTOR algoritmus používá dvě proměnné, resp. proměnnou y a vstupní hodnotu C, která zůstává v rámci celého průběhu LTOR algoritmu neměnná. Z toho důvodu je pro přesnější odhad chybovosti vhodné specifikovat vlastnosti vstupu C. Pokud $b \in C$, pak $P(b_C) = 1$, pokud $b \notin C$, $P(b_C) = 0$. Při konkrétních vlastnostech C tak stačí pouze nahradit příslušné obecné pravděpodobnosti vztahující se k hodnotě C odpovídající hodnotou 0, nebo 1, tyto pravděpodobnosti jsou v důkazu zvýrazněny složenými závorkami. Protože však obecně považujeme pravděpodobnost $P(b)$ za malou, k výrazné změně hodnoty pravděpodobnosti (a to ke zvýšení) dochází pouze při $b \in C$. Odhady této odlišné pravděpodobnosti si pro porovnání uvedeme při specifikovaných parametrech v LTOR-tabulce hodnot v závěru kapitoly.

Analýza RTOL algoritmu

Bez důkazu (jde o analogii důkazu Pozorování 4.2.2) uveďme:

Pozorování 4.2.3. *Uvažujme výše uvedené předpoklady. Při náhodně zvoleném vstupu dojde k součinové chybě v rámci jednoho cyklu RTOL algoritmu s pravděpodobností zhruba* $P(e) = 1/2 \cdot [P(a)^2 + P(a \cap b)(2 - P(a))]$.

4.2.2 Výsledky

Konkrétní hodnoty odvozených pravděpodobností uvádíme v následujících tabulkách, pro specifikované bitové délky procesorových slov w a uvažované bitové délky $|n| = \log n + 1$ zpracovávaných operandů (velikost základu i exponentu bereme shodně). Voleny byly vzhledem k běžným parametrům dnešních procesorů hodnoty $w = 8, 16, 32, 64$ a podle rozsahu dlouhých čísel užívaných v dnešní asymetrické kryptografii délky $|n| = 512, 1024, 2048, 4096$. Při dosazení se ukazuje, že odhadované pravděpodobnosti vzniku chyb za těchto parametrů dosahují výraznějších odchylek jen při uvažování rozdílných algoritmů, v rámci různých implementací téhož algoritmu jsou tyto rozdíly zanedbatelné. Hodnoty chybovosti zařízení jsou tedy uvedeny souhrnně pro všechny možné implementační způsoby.

Z výsledků je například patrné, že nemá smysl za daných předpokladů uvažovat útoky vůči 8-bitovým procesorům, neboť i jen jedna chybová dvojice slov (tj. minimální počet) způsobuje prakticky stoprocentní chybovost ($P = 1$), a tudíž reálné používání takového chybového zařízení v praxi můžeme vyloučit.

Výskyt chybových slov

$P(a)$	$\|n\| = 512$	$\|n\| = 1024$	$\|n\| = 2048$	$\|n\| = 4096$
$w = 8$	$2^{-2.2}$	$2^{-1.3}$	$2^{-0.7}$	$2^{-0.2}$
$w = 16$	2^{-11}	2^{-10}	2^{-9}	2^{-8}
$w = 32$	2^{-28}	2^{-27}	2^{-26}	2^{-25}
$w = 64$	2^{-61}	2^{-60}	2^{-59}	2^{-58}
$w = 128$	2^{-126}	2^{-125}	2^{-124}	2^{-123}
$P(a \cap b)$	$\|n\| = 512$	$\|n\| = 1024$	$\|n\| = 2048$	$\|n\| = 4096$
$w = 8$	$2^{-4.4}$	$2^{-2.7}$	$2^{-1.3}$	$2^{-0.4}$
$w = 16$	2^{-22}	2^{-20}	2^{-18}	2^{-16}
$w = 32$	2^{-56}	2^{-54}	2^{-52}	2^{-50}
$w = 64$	2^{-122}	2^{-120}	2^{-118}	2^{-116}

Chybovost zařízení při **LTOR**[2]

$P(e)$	$\|n\| = 512$	$\|n\| = 1024$	$\|n\| = 2048$	$\|n\| = 4096$
$w = 8$	0.07 / 0.1	0.2 / 0.3	0.5 / 0.5	0.8 / 0.8
$w = 16$	2^{-21} / 2^{-12}	2^{-19} / 2^{-11}	2^{-17} / 2^{-10}	2^{-15} / 2^{-9}
$w = 32$	2^{-55} / 2^{-29}	2^{-53} / 2^{-28}	2^{-51} / 2^{-27}	2^{-49} / 2^{-26}
$w = 64$	2^{-121} / 2^{-62}	2^{-119} / 2^{-61}	2^{-117} / 2^{-60}	2^{-115} / 2^{-59}
P	$\|n\| = 512$	$\|n\| = 1024$	$\|n\| = 2048$	$\|n\| = 4096$
$w = 8$	1 / 1	1 / 1	1 / 1	1 / 1
$w = 16$	2^{-12} / 0.1	2^{-9} / 0.4	2^{-6} / 0.9	2^{-3} / 1
$w = 32$	2^{-46} / 2^{-20}	2^{-43} / 2^{-18}	2^{-40} / 2^{-16}	2^{-37} / 2^{-14}
$w = 64$	2^{-112} / 2^{-53}	2^{-109} / 2^{-51}	2^{-106} / 2^{-49}	2^{-103} / 2^{-47}

Chybovost zařízení při **RTOL**

$P(e)$	$\|n\| = 512$	$\|n\| = 1024$	$\|n\| = 2048$	$\|n\| = 4096$
$w = 8$	$0,07$	$0,2$	$0,5$	$0,8$
$w = 16$	2^{-21}	2^{-19}	2^{-17}	2^{-15}
$w = 32$	2^{-55}	2^{-53}	2^{-51}	2^{-49}
$w = 64$	2^{-121}	2^{-119}	2^{-117}	2^{-115}
P	$\|n\| = 512$	$\|n\| = 1024$	$\|n\| = 2048$	$\|n\| = 4096$
$w = 8$	1	1	1	1
$w = 16$	2^{-12}	2^{-9}	2^{-6}	2^{-3}
$w = 32$	2^{-46}	2^{-43}	2^{-40}	2^{-37}
$w = 64$	2^{-112}	2^{109}	2^{-106}	2^{-103}

[2]Hodnoty uvádíme pro možnosti šifr. textu C: [náhodné C]/[C obsahuje slovo b]

Kapitola 5

Základní útoky

Tato práce se zaměřuje na druh kryptoanalytického útoku, jehož cílem je odhalení hodnoty tajného dešifrovacího klíče, což následně útočníkovi umožňuje dešifrovat všechny zprávy určené vlastníkovi klíče. Typově se jedná o tzv. *chosen-ciphertext attack*, útok s výběrem šifrových textů, kdy útočník vybírá šifrový text dle svého výběru, a poté obdrží nějakým způsobem od oběti (vlastníka tajného dešifrovacího klíče) odpovídající dešifrovaný text. Předpokládáme tedy, že útočník má nějakým způsobem přístup k dešifrovacímu zařízení dané entity, resp. výstupům tohoto dešifrování, ne však k samotnému soukromému klíči. V našem konkrétním případě přitom rozlišujme dva typy tohoto útoku, dle závislosti volby šifrových textů na jeho průběhu:

1. *adaptivní*, kdy útočník může žádat dešifrování libovolných šifrových textů, které jsou (musejí být) vybírány až v průběhu útoku s ohledem na znalost dešifrovaných textů obdržených při předchozích dotazech.
2. *neadaptivní*, kdy útočník může žádat dešifrování libovolných šifrových textů, které jsou (mohou být) připraveny před zahájením útoku, resp. znalost dešifrovaných textů obdržených při předchozích dotazech nijak nemění předpokládaný průběh dotazů pokládaných v krocích následujících.

Jinou klasifikaci v této práci uváděných útoků bude možné provést na základě způsobu vyhodnocování dešifrovaných textů:

1. *chybové útoky* - posuzují jen chybnost, resp. korektnost dešifrovaných textů, nepracují s konkrétními chybovými odchylkami.
2. *útoky s chybovým faktorem* - pracují s přesnými chybovými odchylkami od bezchybně dešifrovaných textů.

V krátkém úvodu si nejdříve popíšeme základní myšlenku útoků a nezbytnou rozdílnost jejich provedení, která je dána vlastnostmi jednotlivých schémat. Poté uvedeme základní útoky na schémata Pohlig-Hellman a RSA, uvažujeme-li, že je modulární mocnění realizováno prostřednictvím algoritmu LTOR. V další části předložíme návrh útoku při použití RTOL algoritmu. Jedná se o útoky převzaté z článku [1] a jsou uvedeny v původní podobě. Všimněme si přitom, že navržené druhy útoků spývají s rozdělením dle použitého druhu algoritmu mocnění - chybový útok je použit pro LTOR algoritmus, útok s chybovým faktorem pro algoritmus RTOL. Porovnání vlastností těchto dvou typů útoků se bude věnovat kapitola 6, rozboru konkrétní realizace daných útoků pak kapitola 7. V té uvidíme, že se jedná spíše o principy útoků než o útoky přímo aplikovatelné ve stávající podobě a navrhneme nutné úpravy, které je pro funkčnost útoků zapotřebí provést.

Princip útoků a rozdílnost schémat

Díky podobným principům obou schémat mohou být útoky v jistém smyslu rovněž podobné. Avšak vzhledem k odlišnému charakteru dešifrovacího klíče e bude zapotřebí do jisté míry modifikovat vyhodnocování výstupních textů. Fungování schématu Pohlig-Hellman v okruhu $\mathbb{Z}_p$ zároveň v určitých případech poskytuje možnost úpravy útoků s nižší složitostí hledání vhodných šifrových textů.

Uvažované algoritmy LTOR a RTOL jsou založeny na technice *repeated square-and-multiply*, kdy je hodnota výstupní proměnné v každém cyklu algoritmu podle hodnoty zpracovávaného bitu exponentu buď násobena sama se sebou (mocněna), nebo násobena s hodnotou jiné pomocné proměnné. Navržené útoky využívají tohoto aspektu a jsou založeny na základní myšlence, že jsme-li schopni prostřednictvím výběru vstupu (zde vstupního šifrového textu) ovlivnit průběžnou hodnotu proměnných v daném okamžiku algoritmu tak, aby došlo k součinu chybové dvojice právě v jednom z obou možných případů, jsme schopni (existuje-li taková možnost) porovnáním výsledku s očekávaným korektním výsledkem rozhodnout, zda došlo k chybnému výpočtu, či nikoliv. Tím pádem jsme schopni i rozhodnout, jaká operace byla v daném okamžiku algoritmu prováděna a tedy určit, jaká je hodnota samotného bitu. Hodnoty proměnných v jednotlivých cyklech algoritmů jsou uvedeny v Pozorování 3.1.1.

Zvolme nějakou hodnotu šifrového textu C a získejme hodnotu textu dešifrovaného na chybovém zařízení, hodnotu $\hat{M}$. Pokud jde o systém RSA,

lze díky veřejnému exponentu e snadno ověřit korektnost hodnoty $\hat{M}$, protože jen proběhlo-li dešifrování bezchybně, tj. $\hat{M} = M$, platí, že $\hat{M}^e = C$. V případě schématu Pohlig-Hellman taková možnost neexistuje, protože exponent e je tajný, tudíž je třeba porovnat přímo hodnoty $\hat{M}$ a M, tj. porovnat výstup bugového zařízení s korektní hodnotou. Teoreticky by tak útočník potřeboval mít přístup nejen k bugovému zařízení, ale rovněž k zařízení, které by disponovalo shodným tajným klíčem a poskytovalo jen bezchybně dešifrované texty. Taková duplicita přitom v praxi obvykle k dispozici nebývá. Nicméně v článku [1] lze nalézt pravděpodobnostní návrh, jak pro porovnání získat, resp. dopočítat takovou bezchybnou hodnotu jen z výstupů bugového zařízení. Tímto aspektem se zde tudíž zabývat podrobněji nebudeme a v útocích budeme používat porovnávání hodnot $\hat{M}$ a M. V případě zájmu čtenáře odkazujeme na příslušný článek, v principu se však jedná o porovnávání hodnoty $\hat{M}^x$ s dešifrovanou hodnotou textu C^x pro nějaké x, u nějž už narozdíl od textu C očekáváme, že jeho dešifrování proběhne korektně.

Hodnotu šifrového textu přitom v souladu s myšlenkou útoků potřebujeme volit tak, aby během modulárního mocnění (tj. v průběhu dešifrování) hodnota proměnné (podle Pozorování 3.1.1 daná mocnina vstupu) splňovala určité vlastnosti, typicky jde o nějak specifikovaný obsah chybových slov. Zatímco tak okruh $\mathbb{Z}_p$ (schéma Pohlig-Hellman) obecně umožňuje zvolit si hodnotu s požadovanou (libovolně „komplikovanou“) vlastností a dopočítat potřebný vstup jako příslušnou odmocninu[1], u RSA díky neznámé faktorizaci (a její předpokládané výpočetní obtížnosti) modula n útočník tuto metodu použít nemůže a musí postupovat opačně - volit náhodnou hodnotu a testovat, zda proměnná bude mít v daných mocninách (tj. v potřebných fázích modulárního mocnění v rámci dešifrování) požadované vlastnosti. Pravděpodobnost takového úspěchu pak zjevně závisí na pravděpodobnosti daného jevu, s jakou nastává při náhodných hodnotách. Výpočetní realizovatelností takového postupu jsme se zabývali v oddíle 3.1.2, hodnoty odpovídajících pravděpodobností nalezneme v části 4.1, resp. tabulkách v části 4.2.2.

5.1 LTOR a chybový útok

V případě LTOR algoritmu článek [1] navrhuje adaptivní chybové útoky, tedy založené jen na posuzování (bez)chybnosti výstupní hodnoty dešifrování.

[1] Teoretické stránce výpočtu se věnuje kapitola 2

Označením *j-tá iterace* útoku mějme na mysli běh *For*-cyklu pro $i = j$. V každé iteraci útoku je určena jedna hodnota bitu exponentu, j-tá iterace zjišťuje hodnotu bitu d_j. Protože jsou bity exponentu odkrývány zleva doprava, tj. směrem od nejvyšších bitů, v j-té iteraci předpokládáme, že již známe hodnoty všech vyšších bitů, tedy bity $d_{\log n}(\text{resp. } d_{\log p}), \ldots, d_{j+1}$.

Poznamenejme, že z kontextu článku [1] vyplývá, že při návrhu útoků byla předpokládána *If-else* verze LTOR algoritmu, a to v implementační podobě, kdy z výrazu z^2C je nejdříve počítána hodnota zC (v tomto pořadí).

Algoritmus 5. *Chybový LTOR-útok na schéma Pohlig-Hellman*

0. Vyberme hodnotu $X \in \mathbb{Z}_p$ obsahující slova a a b.
1. For $i = \log p$ down to 1 do:
   (a) $d' \leftarrow \sum_{k=i+1}^{\log p} 2^{k-(i+1)} d_k$ /hodnota dosud známých bitů/
   (b) $C \leftarrow X^{1/d'} \pmod p$ /viz. pozn.[2], více v důkazu/
   (c) Získejme hodnotu textu dešifrovaného na chybovém procesoru $\hat{M} = C^{\langle d \rangle}$
   (d) Získejme správnou hodnotu $M = C^d$
   (e) If $\hat{M} = M$ then $d_i \leftarrow 1$ else $d_i \leftarrow 0$
2. $d_0 \leftarrow 1$ /platí vždy/

Důkaz správnosti algoritmu. Uvažujme j-tou iteraci algoritmu a odpovídající d'. Vstupní hodnotou proměnné z do j-tého cyklu LTOR algoritmu je dle Pozorování 3.1.1 hodnota $z = C^{d'} = X$, obsahující slova a a b. Je-li $d_j = 0$, je počítán výraz $z^2 = X \cdot X$, kdy dochází k součinu chybových slov a v důsledku toho $\hat{M} \neq M$. Je-li v případě $d_j = 1$ dle specifického předpokladu implementace z počítaného výrazu z^2C nejdříve vyhodnocen výraz $z \cdot C$, bude provedení tohoto chybného součinu zabráněno.[2] □

V případě RSA nelze z teoretického hlediska použít shodný postup (výpočet odmocnin) generování vhodných šifrových textů a inverzní způsob hledání textů - tj. nalezení takového C, že $C^{d'}$ obsahuje slova a a b - by měl složitost určenou pravděpodobností $P(a \cap b)$. Útok pro RSA je z toho důvodu založen na odlišné výpočetní chybě, aby hledání vstupních textů mělo nižší složitost, konkrétně složitost určenou pravděpodobností $P(a)$.

[2] Ačkoliv to článek neuvádí, je zřejmé, že pro úspěšnost zvažované metody nesmí šifrový text C obsahovat chybové slovo b. Tento případ explicitně uvádíme, neboť zřejmě stojí mimo rámec náhodných chyb.

Algoritmus 6. *Chybový LTOR-útok na schéma RSA*

1. For $i = \log n$ down to 1 do:
   (a) $d' \leftarrow \sum_{k=i+1}^{\log n} 2^{k-(i+1)} d_k$ /hodnota dosud známých bitů/
   (b) Nalezněme hodnotu $C \in \mathbb{Z}_n$ obsahující b, přitom takovou, že $C^{d'} \pmod n$ obsahuje a. /viz. pozn.[3], více v důkazu/
   (c) Získejme hodnotu textu dešifrovaného na chybovém procesoru $\hat{M} = C^{\langle d \rangle}$
   (d) $\hat{C} \leftarrow \hat{M}^e$
   (e) If $\hat{C} = C$ then $d_i \leftarrow 0$ else $d_i \leftarrow 1$
2. $d_0 \leftarrow 1$ /platí vždy/

Důkaz správnosti alg. Uvažujme j-tou iteraci útoku a odpovídající d'. Pokud $d_j = 1$, dle specific. předpokladu bude nejdřív vyhodnocován výraz $z \cdot C = C^{d'} \cdot C$, kdy dochází k chybnému součinu, a tak $\hat{M} \neq M$, tedy $\hat{M}^e \neq C$. V opačném případě je počítáno $z^2 = (C^{d'})^2$ vznik chyby neočekáváme.[3] □

5.2 RTOL a útok s chybovým faktorem

U RTOL algoritmu je využito pozorování, že proměnná y je v každém jeho cyklu nahrazována hodnotou y^2 a každá případně takto vzniklá výpočetní chyba se dostává i do všech následujících výpočtů druhé mocniny y v dalších cyklech. Do hodnoty výstupní proměnné z proniká tato chyba právě tehdy, když odpovídající bit, právě zpracovávaný příslušným cyklem, má hodnotu 1. Predikovatelnosti těchto výpočtů v rámci RTOL algritmu lze využít pro návrh neadaptivního útoku. A navíc to, že potenciální přinásobovaná chyba dosahuje v každém cyklu odlišné hodnoty, ve svém důsledku umožňuje vytvořit efektivní útok, kdy lze vypočítat hodnotu více bitů současně.

Navržený útok je společný pro oba systémy, modifikovány jsou v něm jen příslušné metody generování textů a porovnávání výstupních hodnot. Pracujeme s přesnou součinovou chybou dvojice slov a, b, resp. chybným součinem hodnot tyto slova obsahujících. Hodnota chybného součinu je porovnávána s korektní hodnotou jako *chybový faktor* (jde o poměr korektní vůči chybové

[3] Opět je nutný dodatečný předpoklad, že volíme text C tak, že $C^{d'}$ neobsahuje párové chybové slovo b. Tento případ rovněž zřejmě nespadá do kategorie náhodných chyb. Rozmysleme si, že stejně tak C nesmí zároveň obsahovat i slovo a - docházelo by k chybnému součinu vždy v druhém cyklu LTOR algoritmu.

hodnotě), který budeme značit β. Jeho přesný výpočet a použití si ukážeme až v popisu útoku. Bity klíče jsou v rámci útoku opět odkrývány směrem od nejvyšších, v jedné iteraci útoku je vypočítáváno r bitů najednou, přičemž r lze jako parametr útoku volit libovolně (s ohledem na časovou složitost výpočtu - jde o složitost hledání řešení diskrétního logaritmu v kroce 1(e)).

Jako j-tou iteraci označme pro jednoduchost běh *For*-cyklu algoritmu útoku pro $i = j$, nejnižší odkrývaný bit v ní odpovídá bitu d_j, přičemž známe hodnoty vyšších bitů $d_{\log n}(\text{resp. } d_{\log p}), \ldots, d_{j+r}$.

Algoritmus 7. *RTOL-útok s chybovým faktorem na schéma Pohlig-Hellman*

0. (a) Zvolme hodnotu $X \in \mathbb{Z}_p$ obsahující chybová slova a a b
   (b) $X^{\langle 2 \rangle}$ buď výsledek 2. mocniny hodnoty X na chybovém procesoru $\beta \leftarrow X^2 / X^{\langle 2 \rangle} \pmod p$
   (c) Zvolme parametr r (počet odkrývaných bitů v jedné iteraci útoku)
1. For $i = \log p - (\log p \pmod r)$ down to 0, step $-r$, do:
   (a) $C \leftarrow X^{1/2^{(i-1)}} \pmod p$
   (b) $d' \leftarrow \sum_{k=i+r}^{\log p} 2^{k-(i+r)} d_k$
   (c) Získejme hodnotu textu dešifrovaného na chybovém procesoru $\hat{M} = C^{\langle d \rangle}$
   (d) Získejme správnou hodnotu dešifrování $M = C^d$
   (e) Nalezněme r-bitovou hodnotu u, $0 \leq u < 2^r$, která splňuje $M/\hat{M} = \beta^{2^r d' + u} \pmod p$,
   (f) Buď $u_{r-1} u_{r-2} \ldots u_1 u_0$ binární reprezentace u $\qquad /u = \sum_{k=0}^{r-1} u_k 2^k /$
   For $l = 0$ to $(r-1)$ do: $d_{i+k} \leftarrow u_k$

Hlavní modifikací útoku pro schéma RSA je způsob generování vstupních šifrových textů. Ten se řeší v předvýpočtu, v krocích I. a II. Je hledána hodnota X obsahující chybová slova a a b tak, aby byl znám dostatečný počet předcházejících hodnot (2^i-tých odmocnin). Složitost tohoto inverzního způsobu hledání textů je určena zhruba pravděpodobností $P(a \cap b)$.

Algoritmus 8. *RTOL-útok s chybovým faktorem na schéma RSA*

I. Zvolme náhodné $C_0 \in \mathbb{Z}_n$, $t \leftarrow 0$
II. While $t \leq \log n$ or (C_t neobsahuje a i b) do:
   (a) $t \leftarrow t + 1$
   (b) $C_t \leftarrow C_{t-1}^2 \pmod n$

0. (a) $X \leftarrow C_t$ $\qquad$ /C_t obsahuje slova a, b/
   (b) ... Dál pokračuje algoritmus útoku stejně (se záměnou n za p),
1. přičemž volba C (krok 1(a)) odpovídá $C \leftarrow C_{t-i+1}$[4]a je vhodné pozměnit způsob vyhodnocování dešifrovaných textů jako:
   (d) $\hat{C} \leftarrow \hat{M}^e$
   (e) Nalezněme r-bitovou hodnotu u, $0 \leq u < 2^r$, která splňuje $C/\hat{C} = \left(\beta^{2^r d' + u}\right)^e$

   ...

Zjednodušenými zápisy typu $1/x$ máme formálně na mysli inverzní prvek $x^{-1} \pmod p$, resp. $\pmod n$. Uvědomme si, že ačkoliv v $\mathbb{Z}_n$ obecně nejsou všechny prvky invertibilní, v tomto případě je $n = pq$, tudíž nalezení neinvertibilního prvku by vedlo k faktorizaci daného n.

Důkaz správnosti algoritmu. Pokud předpokládáme j-tou iteraci útoku, do $(j-1)$-ního cyklu RTOL algoritmu vstupuje dle Pozorování 3.1.1 hodnota $y = C^{2^{j-1}} = X$, která obsahuje chybová slova a i b. Když je y na konci cyklu mocněno, vzniká chyba, kterou lze v porovnání s bezchybnou hodnotou druhé mocniny, X^2, vyjádřit jako *součinový chybový faktor* $\beta = X^2/X^{\langle 2 \rangle}$. Hodnota $y = X^{\langle 2 \rangle}$ vstupuje do j-té iterace. Pokud $d_j = 1$, pak je y přinásobeno k hodnotě výstupní proměnné z, a tím se stejný chybový faktor dostává do výsledného poměru vůči bezchybné hodnotě. Po následujícím výpočtu druhé mocniny bude proměnná y v dalším kroce alg. RTOL určovat chybový faktor β^2, který se opět promítne do hodnoty z, pokud $d_{j+1} = 1$, a tak dále pro všechny následující kroky. Na konci RTOL algoritmu je výsledný chybový faktor poměru dešifrované hodnoty $\hat{M}$ vůči korektní hodnotě M roven součinu jednotlivých chybových faktorů, které ovlivňují výsledek právě tehdy, když je příslušný bit exponentu jedničkový. Celkovou chybu tak lze vyjádřit rovnicí:

$$\frac{M}{\hat{M}} = \prod_{k=j}^{\log n} \left(\beta^{2^{k-j}}\right)^{d_k} = \beta^{\sum_{k=j}^{\log n} 2^{k-j} d_k},$$

přičemž z bitů d_j až $d_{\log n}$ neznáme pouze dolních r bitů.

Tedy $\sum_{k=j}^{\log n} 2^{k-j} d_k = 2^r \sum_{k=j+r}^{\log n} 2^{k-(j+r)} d_k + \sum_{k=j}^{j+r-1} 2^{k-j} d_k = 2^r \cdot d' + u$, kde d' je hodnota známých bitů a u je hodnota neznámých r bitů, $0 \leq u < 2^r$. $C/C' = (M/M')^e$. □

[4]Oprava článku [1], kde je uváděno C_{t-i}.

Kapitola 6

Přesnost útoků a eliminace vlivu náhodných chyb

6.1 Analýza úspěšnosti chybového útoku

Pozorování 6.1.1. *Uvažujme i-tou iteraci chybového LTOR-útoku (5.1), jemu odpovídající způsob volby šifrového textu a znalost vyšších bitů klíče d, tj. mějme správnou hodnotu d′.*

Pohlig-Hellman (Alg. 5):
Bezchybný výstup implikuje $d_i = 1$*, při chybném výstupu je* $d_i = 0$ *až na pravděpodobnost vzniku náhodné chyby.*

RSA (Alg. 6):
Bezchybný výstup implikuje $d_i = 0$*, naopak při chybném výstupu je* $d_i = 1$ *až na pravděpodobnost vzniku náhodné chyby.*

Důkaz. Uvažujme útok na schéma Pohlig-Hellman (při chybném výstupu dešifrování volíme $d_i \leftarrow 0$). Lze si rozmyslet, že pokud $d_i = 0$, při zvoleném šifrovém textu bude výstup LTOR algoritmu vždy chybný, ať už výpočetní chyba vznikne v předpokládaném či jiném místě algoritmu, tudíž bezchybný výstup nastane jen v situaci, že $d_i = 1$ a nedošlo k náhodné součinové chybě. Naopak chybný výstup může nastat i v případě $d_i = 1$, a to, dojde-li k náhodné chybě během výpočtu. RSA-útok probíhá analogicky. □

Důsledek 6.1.2. *P buď pravděpodobnost vzniku náhodné chyby v rámci průběhu LTOR/RTOL algoritmu (viz. sekce 4.2.1) a předpokládejme správ-*

ně určené hodnoty vyšších bitů klíče při i-té iteraci. Pak pravděpodobnost chybně určeného bitu d_i je následující.[1]

Pohlig-Hellman (při chybném výstupu volíme $d_i \leftarrow 0$):
$P(d_i \leftarrow 0 | d_i = 1) = P$, *naopak* $P(d_i \leftarrow 1 | d_i = 0) = 0$.

RSA (při chybném výstupu volíme $d_i \leftarrow 1$):
$P(d_i \leftarrow 1 | d_i = 0) = P$, *naopak* $P(d_i \leftarrow 0 | d_i = 1) = 0$.

Důsledkem tak pravděpodobnost, s jakou se můžeme spolehnout na správnost určení hodnoty bitu podle chybného výsledku dešifrování, je daná pravděpodobností P, s jakou dochází k náhodným chybám. Přičemž vždy jen jedna hodnota bitů (nulová, nebo jedničková, to v závislosti na útoku) může být při správně určených vyšších bitech klíče zvolena chybně.

Pro praktickou použitelnost útoku je nutné, aby pravděpodobnost nesprávně zvolené hodnoty bitu byla „dostatečně malá“. Případně lze tuto pravděpodobnost snížit k-násobným opakováním útoku pro každou iteraci (s různými šifrovými texty), přičemž je navíc i vhodné využít následujícího pozorování pro detekci špatně určených hodnot vyšších bitů. Přesnou podobu modifikovaného útoku pro schéma RSA uvádí Alg. 9, modifikace pro Pohlig-Hellman je analogická, jde jen o záměnu role nulových a jedničkových bitů.

Pozorování 6.1.3. *Předpokládáme-li v i-té iteraci chybového LTOR-útoku špatnou hodnotu vyšších bitů klíče, bude při volbě šifrového textu (způsobem odpovídajícím útoku) vstupní hodnotou proměnné z v i-tém cyklu LTOR algoritmu neočekávaná hodnota, kterou můžeme považovat za náhodnou, a tak s pravděpodobností $1 - P$ bude vrácen bezchybný výstup.*

Algoritmus 9. *Pravděpodobnostní úprava LTOR-útoku pro RSA*
P buď pravděpodobnost vzniku náhodné chyby v průběhu LTOR algoritmu, tj. odpovídající pravděpodobnosti pro jednu iteraci LTOR-útoku. Zvolme parametr k pro počet opakování iterací útoku tak, aby pravděpodobnost chybně určeného bitu P^k byla „dostatečně malá“[2]. Dále zvolme vhodný parametr l pro „málo pravděpodobnou“ délku sekvence po sobě jdoucích nul v binární reprezentaci náhodně zvoleného čísla. Útok pak provádějme takto:

a) Iteraci útoku Alg. 6 pro každý bit opakujme k-krát, vždy s jinou hodnotu šifrového textu C. Platí-li vždy $C \neq \hat{C}$ (resp. $M \neq \hat{M}$), volme $d_i \leftarrow 1$, jinak $d_i \leftarrow 0$.

[1] Zbývající pravděpodobosti lze odvodit z rovnosti $P(A|B) + P(\bar{A}|B) = 1$.
[2] Pro volbu parametru k se jako vhodný ukázal odhad $P^k < \frac{1}{(\log n)k}$.

b) Tvoří-li bit d_i s vyššími bity sekvenci po sobě jdoucích nulových bitů delší než l, otestujme hodnotu posledního jedničkového bitu, předpokládejme, že ležícího na nějaké pozici j. Dojdeme-li ke shodnému výsledku $d_j = 1$, pak pokračujme v útoku na pozici d_{i+1}, v opačném případě se vraťme na pozici j, položme $d_j \leftarrow 0$ a pokračujme v útoku na pozici d_{j+1}.

Důkaz správnosti alg. Úprava algortimu vychází z Pozorování 6.1.1 a jeho Důsledku. Je-li $d_i = 1$, k chybnému výstupu dochází vždy a bit je určen vždy správně. Je-li $d_i = 0$, pravděpodobnost, že náhodná chyba nastane při všech k iteracích a hodnota bitu bude určena chybně, je P^k (z předpokladu nezávislosti jednotlivě prováděných iterací). Pravděpodobnost, že alespoň v jedné iteraci nenastane náhodná chyba, je doplňkem předchozího jevu, tedy nulový bit bude určen správně s pravděpodobností $1 - P^k$, $P^k \leq P^{k-1}$.

Pokud jsme zvolili chybně nějakou hodnotu vyššího bitu, pak dle předchozího Pozorování 6.1.3 ke vzniku chyby dochází náhodně a hodnotu 0 volíme, pokud alespoň v jednom případě nenastane chybový výstup, tj. s pravděpodobností $1-P^k$. Rostoucí k tedy snižuje pravděpodobnost chybně určeného bitu (1 namísto 0) při správně určených vyšších bitech a zároveň zvyšuje pravděpodobnost zvolení nulového bitu v případě chybně určeného některého z vyšších bitů. „Příliš dlouhá“ sekvence nul tak může dobře signalizovat, že došlo k špatnému určení nějakého předchozího bitu, a při správné znalosti nejvyšších bitů mohl být chybně zvolen jen jedničkový bit. □

6.2 Analýza útoku s chybovým faktorem

Pozorování 6.2.1. *Předpokládejme znalost vyšších bitů klíče d, tj. mějme správnou hodnotu d'. Nemá-li rovnice $M/\hat{M} = \beta^{2^r d' + u}$ pro i-tou iteraci řešení u (hodnota bitů $d_{i+r-1} \ldots d_i$), došlo k náhodné chybě. Přitom existence řešení v případě vzniku náhodné chyby není příliš pravděpodobná.*

Důkaz. Pokud k náhodné chybě nedojde, rovnice má správné řešení (viz. rozbor algoritmu). Útok předpokládá v i-tém cyklu RTOL alg. hodnotu chybového faktoru β a v každém následujícím cyklu hodnotu chybového faktoru β^{2^j}, $0 < j \leq \log n - i$. Nastane-li náhodná chyba ať už při výpočtu y^2 nebo $z \cdot y$, je pravděpodobné, že oba takto neočekávané, náhodně určené chybové faktory budou odlišné od předpokládaných β^{2^j}, a proto není příliš pravděpodobné, že by rovnice při vzniku náhodné chyby měla očekávané řešení. □

Výskyt náhodných chyb tedy dokážeme zjistit neřešitelností dané rovnice, navíc vhodnou úpravou algoritmu dle následujícího popisu jsme schopni veškeré součinové chyby vznikající při výpočtu y^2 předvýpočtem přesně určit a za určitých podmínek útok i při tomto typu chyb úspěšně provést, čímž lze vhodně redukovat celkový počet náhodných chyb a jejich negativní vliv na úspěšnost útoku. Původní návrh RTOL-útoku je jeho speciálním případem.

Algoritmus 10. *Úprava algoritmu RTOL-útoku*
Dle $C \to C^{\langle 2 \rangle} \to C^{\langle 2^2 \rangle} \to \dots$ simulací na chybovém zařízení spočtěme jako $\beta_i = \frac{C^{2^i}}{C^{\langle 2^i \rangle}}$ jednotlivé chybové faktory $\beta_0, \beta_1, \dots, \beta_{\log n}$, přičemž vždy $\beta_0 = 1$. Uvažujeme-li i-tou iteraci útoku a $\beta_j = 1$ pro všechna $0 \leq j < i$, pak k určení bitů $d_{i+r-1}, \dots, d_i$ hledáme řešení rovnice $M/\hat{M} = \beta_{\log n}^{d_{\log n}} \dots \beta_{i+r}^{d_{i+r}} \cdot \beta_{i+r-1}^{d_{i+r-1}} \dots \beta_i^{d_i}$, přičemž bity $d_{i+r}, \dots, d_{\log n}$ známe.

Nicméně poznamenejme, že použití této varianty při praktickém testování útoku (při jedné chybové dvojici) nebylo nutné. Pokud v dané iteraci není možné najít řešení rovnice, často namísto opakování útoku s nově zvoleným chybovým faktorem stačí využít následujícího pozorování.

Pozorování 6.2.2. *Není-li v i-té iteraci rovnice řešitelná, vhodnou změnou parametru r (počet odkrývaných bitů v jedné iteraci útoku) lze obejít vznikající náhodnou chybu a nalézt tak hodnoty bitů vztahujících se k této iteraci.*

Důkaz. Není-li v i-té iteraci rovnice řešitelná, dochází podle Pozorování 6.2.1 během operace dešifrování ke vzniku náhodné výpočetní chyby. Je-li zvolen parametr r tak, že není prováděna iterace i s šifrovým textem $X^{1/2^{i-1}}$, je pro zjištění hodnoty bitu d_{i+k}, $0 \leq k < r$, použita nějaká j-tá iterace, $j \neq i$, s textem $X^{1/2^{j-1}} \neq X^{1/2^{i-1}}$, který zřejmě nezpůsobuje náhodnou chybu ve shodném místě algoritmu jako předchozí text. Obecně při jeho použití dojde k náhodné chybě a tím opět k neřešitelnosti rovnice s pravděpodobností P jako při ostatních náhodně volených textech. □

Výskyt náhodných chyb při tomto druhu útoku ovlivňuje pouze složitost nalezení vhodných šifrových textů, avšak narozdíl od chybových útoků, které jsou do jisté míry jen pravděpodobnostního charakteru, nezpůsobuje jeho nepřesnost. Práce s přesnými chybovými odchylkami dokáže určit původ vzniklých chyb, proto má tento druh útoku v porovnání s chybovým útokem mnohem lepší vlastnosti. To vedlo ke snaze aplikovat myšlenku chybového faktoru i v případě používání LTOR algoritmu.

Kapitola 7

Diskuse k útokům

Návrhy útoků z článku [1], uvedené v kapitole 5, jsou spíše ideové. Při realizaci těchto postupů se setkáme s určitými otázkami, které „obecnost“ popisu útoku může vyvolávat, nebo dokonce s některými překážkami, které je pro možnou realizaci nutné nějakým způsobem odstranit. Tato kapitola jednotlivé problémy diskutuje a současně případně předkládá návrh jejich řešení. Realizovatelnost útoků na základě složitosti nalezení vhodných šifrových textů v této kapitole ponecháváme stranou.

7.1 LTOR-útoky

Ačkoliv jsou oba navržené LTOR-útoky (Alg. 5,6) mírně odlišné, vyskytují se v nich shodné „inicializační“ problémy (bez újmy na obecnosti v popisu použijme modul n). Navíc připomeňme, že článek [1] při jejich návrzích předpokládá specifický způsob implementace, nepoužitelný pro obvyklejší *If*-verzi LTOR algoritmu. Možnou podobu útoku pro odlišnou implementaci naznačíme v samostatné části 7.1.1, část 7.1.2 je věnována možnosti použití chybových faktorů při LTOR.

Problém I. (*Délka a hodnota soukromého klíče*)
Algoritmus útoku je popsán pro obecný tvar klíče $d = d_{\log n} \dots d_1 d_0$, jehož délka představuje pouze horní odhad počtu bitů nutných pro jeho binární reprezentaci. Ve skutečnosti mohou být některé nejvyšší bity nulové, skutečný klíč je binární reprezentace $d = d_k \dots d_1 d_0$ pro nějaké $k \leq \log n$, $d_k = 1$. Jak zjistíme, pro jaké nejvyšší i máme iteraci útoku skutečně provádět, resp. jaká je skutečná platná délka klíče?

Rozmysleme si, že LTOR algoritmus je implementačně nezávislý na tom, zda je hodnota exponentu, zde klíče d, v binární reprezentaci uložena jen v platné délce, tj. s jedničkovým nejvyšším bitem, či s některými počátečními nejvyššími bity nulovými (v rámci algoritmu po tu dobu zůstává v proměnné z stále jen hodnota 1), a díky tomu lze útok provádět bez ohledu na to, k jakým indexům se příslušné hodnoty bitů skutečně vztahují, tj. první provedenou iterací útoku zjistíme hodnotu prvního nejvyššího platného bitu klíče atd. Platnou délku je možné jednoduše určit až po provedení maximálního možného počtu kroků (otestováním rovnosti $C^d = M$ na nějakém šifrovém textu C použitém v rámci útoku, pro nějž byl obdržen bezchybně dešifrovaný text M; přitom víme, že platná část klíče končí jedničkovým bitem) nebo ji lze zjistit již v průběhu útoku, a to následovně.

Bude-li útok obecně prováděn až do nejnižšího bitu, tj. nebude-li nejnižší bit nastavován explicitně, pak vrácení hodnoty dešifrovaného textu, který odpovídá v iteraci útoku specificky volenému X (resp. $C^{d'}$), indikuje, že v minulé iteraci útoku byl zjištěn poslední platný bit. Nedostaneme-li takový dešifrovaný text ani v poslední iteraci, je délka klíče zřejmě maximální.

Důkaz správnosti. Předpokládejme, že jsme odkryli již všechny bity exponentu d. Pak v následující iteraci útoku platí $d' = d$ a je volen šifrový text $C = X^{1/d'}$. Výstupem dešifrovacího zařízení bude (nedojde-li k náhodné chybě) hodnota $\left(X^{1/d'}\right)^d = X$ (resp. $C^{d'}$). □

Problém II. (*Inicializace útoku*)
Algoritmus útoku vychází obecně z principu, že známe nějaký úsek nejvyšších bitů a podle jejich hodnot volíme šifrový text s takovými vlastnostmi, abychom zjistili hodnotu následujícího nižšího bitu. *Při nejvyšších dvou iteracích (pro zjištění nejvyšších dvou bitů klíče) však útok správně nefunguje* a pro určení hodnoty druhého nejvyššího bitu je třeba zvolit jinou metodu.

Důkaz správnosti. V první iteraci útoku lze předpokládat přirozenou defaultní hodnotu $d' = 0$, nicméně volba požadovaného šifrového textu by nebyla zřejmá. Avšak dle předchozího víme, že první iterace algoritmu zjišťuje první nejvyšší bit a ten je jedničkový. Tudíž první iteraci lze jednoduše přeskočit, $d' = 1$.

V druhé iteraci obou útoků dochází k takové volbě vstupního šifrového textu, že obsahuje obě chybová slova a i b, což způsobuje chybný výstup nezávisle na hodnotě druhého nejvyššího bitu (tj. v obou případech). □

Jednou možností, jak zjistit hodnotu druhého nejvyššího bitu, je použití *speciálně vygenerovaného šifrového textu*, který je volen tak, aby chyby vznikaly typicky právě tehdy, když hodnota druhého nejvyššího bitu je 1 (nezávisle na hodnotách ostatních bitů).[1] Nicméně s ohledem na složitost jeho nalezení může být vhodnější zvolit *pravděpodobnostní postup paraleleního výpočtu*, kdy po nějakou dobu provádíme útok pro obě možné hodnoty tohoto bitu a jakmile obdržíme první chybně dešifrovaný text, jedná se pravděpodobně, dle Pozorování 6.1.3, o správnou volbu předpokladu.

Při realizaci je navíc nutné dbát na dodatečné podmínky kladené na šifrové texty - uvedené v algoritmech útoků jako poznámka pod čarou 2 a 3.

7.1.1 LTOR-útok pro odlišnou implementaci

Ačkoliv původní navržené chybové útoky pro LTOR algoritmus vychází z předpokladu specifické, dokonce poměrně neobvyklé implementace tohoto algoritmu a jsou tak nepoužitelné pro zbývající implementační případy, ukazuje se, že není velký problém princip LTOR-útoku na RSA (Alg. 6) modifikovat i pro „standardní“ *If*-verzi, a to i se zachováním stejné složitosti. Modifikovaná verze původnímu útoku dokonce velmi přesně odpovídá, jen namísto hodnoty $C^{d'}$ volíme text s ohledem na hodnotu $C^{2d'}$. Vzhledem k analogii jen stručně popišme princip útoku pro i-tou iteraci:

Nalezněme náhodný šifrový text C obsahující b (nikoliv a) takový, že $C^{2d'}$ obsahuje chybové slovo a (nikoliv b), přičemž $d' = \sum_{k=i+1}^{\log n} 2^{k-(i+1)} d_k$ je hodnota vyšších známých bitů. Při bezchybně dešifrovném textu zvolme $d_i \leftarrow 0$, jinak $d_i \leftarrow 1$.

Důkaz správnosti alg. Dle Poz. 3.1.1 do i-tého cyklu LTOR algoritmu vstupuje hodnota $z = C^{d'}$. Pokud $d_i = 1$, je počítán výraz $(z^2) \cdot C = \langle C^{2d'} \cdot C \rangle$, kde dle způsobu volby hodnot dojde k součinové chybě. Je-li $d_i = 0$, pak je v následujícím cyklu počítán výraz $z^2 \cdot z^2$, nebo $(z^2 \cdot z^2)C$ (dle hodnoty bitu d_{i-1}), ani v jednom z těchto výrazů, ve kterých se hodnota $z^2 = C^{2d'}$ jedině vyskytuje, k součinové chybě podle zvolených vlastností textu nedojde. □

[1] Jedná se o univerzálně fungující šifrový text C takový, že $b \in C$, $a, b \notin C^2$, $a, b \in C^3$, konkretizace implementace dovoluje některé výjimky vypustit. Při použití takového textu bezchybný výstup implikuje počáteční bity $10\ldots$, zatímco chybný výstup s určitou pravděpodobností (až na vznik náhodné chyby) odkazuje na případ $11\ldots$

7.1.2 LTOR a chybový faktor

Jak vyplývá z analýzy kapitoly 6, při práci s přesnými hodnotami chybných výstupů, tj. použitím principu chybových faktorů, lze poměrně snadno dle řešitelnosti určité rovnice rozhodnout, zda chybný výstup způsobila chyba očekávaná, která vypovídá o hodnotě konkrétního bitu, či zda jej způsobila chyba náhodná. Snahou proto bylo aplikovat myšlenku chybových faktorů i v útocích při použití LTOR algrotimu. Útok tím získává užitečnou vlastnost detekce náhodných chyb a jak uvidíme, lze tak dokonce přesně určit bitovou délku klíče.

Chybový faktor lze zavést přímo do návrhů chybových útoků, a to poměrně jednoduchým způsobem (i když, narozdíl od uvedeného RTOL algoritmu, už nelze snadno zajistit výpočet více než jen jednoho bitu klíče v jedné iteraci). Kroky jako je volba textu zůstanou stejné, jde jen o to na základě principu útoku dodatečně určit hodnotu chybového faktoru a stanovit rovnici, jejíž platnost určuje hodnoty bitů klíče.

Uveďme zde popis LTOR-útoku s chybovým faktorem pro běžněji používanou *If*-verzi implementace a pro schéma RSA, jeho modifikace pro systém Pohlig-Hellman je na základě dříve probíraných rozdílů již zřejmá. Princip vychází z chybového útoku části 7.1.1.

V první části útoku zjišťujeme platnou délku soukromého klíče d, druhá část popisuje výpočet jeho bitů, opět směrem od nejvyšších.

Algoritmus 11. *LTOR-útok s chybovým faktorem pro RSA*

I. (a) Zvolme $C \in \mathbb{Z}_n$ obsahující chybová slova a a b
   (b) $\beta \leftarrow \frac{C^2}{C^{\langle 2 \rangle}} \pmod n$ /chybový faktor/
   (c) Získejme hodnotu textu dešifrovaného na chybovém procesoru $\hat{M} = C^{\langle d \rangle}$
   (d) $\hat{C} \leftarrow \hat{M}^e$
   (e) Nalezněme[2] t' takové, že $C/\hat{C} = \left(\beta^{2^{t'}}\right)^e$
   (f) Return($t \leftarrow t' + 2$) /délka klíče/

Klíč je délky t, tvaru $d = d_{t-1} \ldots d_1 d_0$.

[2]Stačí zřejmě testovat několik horních hodnot blízkých (menších) délce $\log n + 1$, předpokládáme totiž bitovou délku klíče $|d| \approx \log n + 1$.

1. $d_{t-1} \leftarrow 1$
2. For $i = t-2$ down to 0 do:
   (a) $d' = \sum_{k=i+1}^{t-1} 2^{k-(i+1)} d_k$ /hodnota známých bitů/
   (b) Zvolme šifrový text $C \in \mathbb{Z}_n$ takový, že:
   $b \in C$ $(a \notin C)$, $a \in C^{2d'}$ $(b \notin C^{2d'})$
   (c) $\beta_i \leftarrow \frac{C^{2d'+1}}{\langle C^{2d'} \cdot C \rangle} \pmod n$ /chybový faktor/
   (d) Získejme hodnotu textu dešifrovaného na chybovém zařízení
   $\hat{M} = C^{\langle d \rangle}$
   (e) $\hat{C} \leftarrow \hat{M}^e$
   (f) If $C = \hat{C}$ then $d_i \leftarrow 0$
   If $C/\hat{C} = \left(\beta_i^{2^i}\right)^e$ then $d_i \leftarrow 1$

Důkaz správnosti algoritmu. Jako $\bar{d} = \sum_{k=0}^{i-1} 2^k d_k$ označme hodnotu bitů $d_{i-1} \dots d_0$. Předpokládejme klíč d délky t, pak dešifrování textu použitím LTOR alg. můžeme pro klíč $d = d_{t-1} \dots d_1 d_0$ při absenci náhodných chyb rozepsat jako:
$M = ((((C^2 \cdot C^{d_{t-2}})^2 \dots \cdot C^{d_{i+1}})^2 \cdot C^{d_i})^2 \cdot C^{d_{i-1}} \dots)^2 \cdot C = ((C^{d'})^2 \cdot C^{d_i})^{2^i} \cdot C^{\bar{d}}$.
Při volbě šifrového textu C tak, že $a, b \in C$, nastává chybný výpočet vždy, již v druhé iteraci, kdy je počítáno $C^{\langle 2 \rangle}$. Při bitové délce t tak při absenci náhodných chyb platí $\frac{M}{\hat{M}} = \frac{(C^2 \cdot C^{d_{t-2}})^{2^{t-2}} \cdot C^{\bar{d}}}{(\langle C^2 \rangle \cdot C^{d_{t-2}})^{2^{t-2}} \cdot C^{\bar{d}}} = \beta^{2^{t-2}}$, kde $\beta = \frac{C^2}{C^{\langle 2 \rangle}}$. Délka klíče je tedy o 2 bity větší, než hodnota řešení t' rovnice $M/\hat{M} = \beta^{2^{t'}}$. Přitom $C/\hat{C} = (M/\hat{M})^e$.

Uvažujme i-tou iteraci útoku, připomeňme, že vychází z principu chybové verze z předchozí části 7.1.1. Pokud $d_i = 1$, pak předpokládáme výpočetní chybu $z \leftarrow \langle C^{2d'} \cdot C \rangle$ v i-tém cyklu LTOR alg., dešifrování lze zapsat jako $\hat{M} = \langle C^{2d'+1} \rangle^{2^i} \cdot C^{\bar{d}}$ a platí $\frac{M}{\hat{M}} = \frac{(C^{2d'+1})^{2^i} \cdot C^{\bar{d}}}{\langle C^{2d'+1} \rangle^{2^i} \cdot C^{\bar{d}}} = \beta_i^{2^i}$. Pokud $d_i = 0$, dle principu útoku je očekáváno bezchybné dešifrování, tj. rovnost textů, neboli poměr 1. Jiný poměr $M/\hat{M}$ zřejmě signalizuje, že došlo k náhodné chybě. □

Varianta pro odlišnou implementaci LTOR

Chceme-li získat útok s chybovým faktorem vůči RSA při implementaci typu $z(zC)$ nebo $(zC)z$, stačí obdobným způsobem modifikovat chybový útok pro RSA (Alg. 6). Volíme faktor $\beta = (C^{d'} \cdot C)/\langle C^{d'} \cdot C \rangle$ a ověřujeme platnost rovnice $C/\hat{C} = (\beta^{2^i})^e$. Platí-li, pak $d_i \leftarrow 1$, při poměru 1 volíme

$d_i \leftarrow 0$. V případě tohoto útoku je již opět (ve shodě s původním chybovým návrhem) nutné věnovat pozornost počáteční inicializaci útoku, neboť algoritmus nelze aplikovat pro druhý nejvyšší bit - text by byl volen tak, že $a, b \in C$ a získáme poměr v obou případech hodnoty bitu stejný (viz. výpočet délky klíče v Alg. 11). Nalezení správné hodnoty bitu je zde však díky přesně očekávaným hodnotám poměru $C/\hat{C}$ jednoduché.

7.2 RTOL-útoky

Problém I. (*Inicializace a provádění iterací algoritmu útoku*)
Algoritmus v původní podobě není aplikovatelný pro nejnižší iteraci. Pro korektní fungování algoritmu je třeba dát pozor na počáteční inicializaci proměnné $d' \leftarrow 0$ a *iterace algoritmu provádět jako:*

> For $i = \log n - ((\log n - 1) \bmod r)$ down to 1, step: $-r$ ...
> Na závěr: $d_0 \leftarrow 1$

Důkaz. Podle původního návrhu v poslední iteraci pro $i = 0$ volíme text $C \leftarrow X^{1/2^{-1}} = X^2$, který už očekávaný chybový faktor $\beta = X^2/X^{\langle 2 \rangle}$ ve výsledném poměru způsobit nemůže a nijak se z hlediska vzniku náhodných chyb neliší od jiné libovolně zvolené hodnoty C. Sekvence prováděných iterací je proto vhodné „posunout o 1 bit doleva“ a provádět je tak pouze do $i = 1$, aby poslední vypočtené u odpovídalo $u = d_r d_{r-1} \dots d_1$ (funguje). □

Poznámka. (*Hledání a volba vstupních šifrových textů v útoku na RSA*)
Stačí použít cyklus While $(t < \log n - 1)$ or (C_t neobsahuje a i b) do: ..., přičemž volba šifrového textu $C \leftarrow C_{t-i+1}$ v i-té iteraci je správná.

Důkaz. Postupně generované texty ve *While*-cyklu (viz. Alg. 8) lze vyjádřit jako $C_i = C_0^{2^i}$, kde C_0 je první, náhodně volený text. *While*-cyklus končí ve chvíli, kdy C_t pro určité t obsahuje a a b, $X = C_t$. Při šifrovém textu C do i-tého cyklu RTOL algoritmu vstupuje dle Pozorování 3.1.1 hodnota $y = C^{2^i}$. Ve shodě s útokem na schéma Pohlig-Hellman potřebujeme, aby volba textu odpovídala tomu, že do i-tého cyklu vstupuje hodnota $y = X^{\langle 2 \rangle}$, což splňuje volba C_{t-i+1}, platí $C_{t-i+1}^{2^i} = (C_0^{2^{t-i+1}})^{2^i} = (C_0)^{2^{t+1}} = (C_0^{2^t})^2 = C_t^{\langle 2 \rangle}$.

Nově navržený tvar While-cyklu končí nejdříve, pokud $t = \log n - 1$ a $C_{\log n - 1}$ obsahuje chybovou dvojici. Pak C_0 je v útoku použitý text s nejnižším indexem a tedy vygenerovaná posloupnost textů je optimální délky. □

Jak jsme si ukázali v předchozí kapitole, použití chybových faktorů je sice výhodnější, ale jsou situace, jako například použití bezpečnostního schématu OAEP, o kterém se zmíníme v kaptitole 8, ve kterých přesné hodnoty chybových výstupů nelze získat. Vytvořit z útoku s chybovým faktorem pouze útok chybový by teoreticky nebyl problém. Avšak rozmysleme si, že u uvedeného typu RTOL-útoku, který byl navržen tak, že výpočetní chyba vznikala primárně vždy v pomocné proměnné y a nikoliv přímo ve výstupní proměnné z, by například použití tohoto modelu v chybové formě nefungovalo - k chybě, a tedy k chybnému dešifrování, dochází vždy, nezávisle na hodnotách bitů klíče.

Při RTOL algoritmu se ale nabízí ještě jiný druh útoku, než jaký jsme uváděli, dobře využitelný právě pro chybové útoky. Jedná se o útok založený na součinové chybě $\langle zy \rangle$, namísto dosud používané chyby $\langle y^2 \rangle$. S využitím Pozorování 3.1.1 jde o víceméně zřejmou analogii chybového LTOR-útoku, proto zde jeho konkrétní popis již uvádět nebudeme. Lze jej případně nalézt v článku [1] (jako útok na RSA-OAEP). Útok je adaptivní, v každé iteraci je třeba nalézt nový šifrový text, se složitostí $P(a)P(b)$, a tak i když při uvažovaných parametrech platí, že $P(a)P(b) \approx P(a \cap b)$, celková složitost oproti uváděnému RTOL-útoku z části 5.2, kde postačoval jeden text pro celý útok, je vyšší. Pokud tedy nejsme nuceni používat jen chybový útok, zdá se, že použití tohoto druhého útoku ani nebudeme zvažovat. Avšak zmiňme zde, že tento alternativní adaptivní RTOL-útok odkrývá hodnoty bitů klíče od nejnižších, což zejména pro RSA může mít podstatné důsledky, a to možnost efektivního dopočítání hodnoty klíče při znalosti pouhé části jeho binární reprezentace. Konkrétněji se k tomuto tématu vyjádříme v samotném závěru práce.

Kapitola 8

RSA a vliv optimalizací a bezpečnostních schémat

Kryptografická schémata nemusejí být vždy implementována v jednoduché formě tak, jak je samotný princip kryptosystému uváděn, ale mohou k nim být například přidávána další bezpečnostní schémata, která mohou nějak měnit průběh operací či hodnoty použitých operátorů. Zejména ve výpočetně náročné asymetrické kryptografii pak bývají také používány různé výpočetní optimalizace, které mohou způsobit odlišný průběh výpočtů v porovnání se základním principem. Předmětem této části je zaměřit se konkrétně na schéma RSA, uvážit časté optimalizace a schémata používaná v jeho případě a provést rozbor, zda může nějak jejich použití zamezit provedení v textu probíraných útoků, případně podobu útoků vhodně modifikovat.

RSA-šifrování je postaveno na operaci modulárního mocnění, realizovaného operacemi modulárního násobení velkých čísel. Tyto operace tak bývají považovány z výpočetního hlediska za časově rozhodující a výkonnostní implementace se proto zaměřují právě na jejich optimalizaci. Mezi časté optimalizace v rámci modulárního mocnění patří používání Montgomeryho násobení, které používá efektivní způsob modulární (Montgomeryho) redukce v průběhu výpočtu součinu, namísto aplikace klasické redukce, která bývá obvykle prováděna až po ukončení operace násobení a funguje na principu výpočtu zbytku po celočíselném dělení. Konkrétně v případě RSA je to pak také často aplikace CRT (Čínské věty o zbytcích), kdy je operace mocnění rozložena na dvě modulární mocnění při výrazně menších modulech, což výrazně redukuje celkový výpočetní čas. Při redukci výpočetní

složitosti lze zvažovat i použití pokročilejších technik jako Karatsuba-Ofman a FFT násobení, zde se však zaměříme jen na výše uvedené optimalizace.

Pokud jde o zvýšení bezpečnosti, RSA bývá používáno společně s OAEP schématem, na který se rovněž zaměříme, a to v poslední části kapitoly.

8.1 Použití CRT

Proces dešifrování $M = C^d \pmod{n}$, kde $n = pq$, bývá z důvodu výpočetní složitosti optimalizován aplikováním CRT. Spočítány jsou dílčí výsledky $M_p = C_p^{d_p} \pmod{p}$, $M_q = C_q^{d_q} \pmod{q}$, kde C_p, C_q jsou hodnoty šifrového textu C redukované modulo p, resp. q, a d_p, d_q hodnoty klíče d redukované modulo $p - 1$, resp. $q - 1$. Dešifrovaný text se pak vypočítá jako $M = M_p + (M_q - M_p) \cdot p_{inv} \cdot p \pmod{n}$, kde $p_{inv} = p^{-1} \pmod{q}$.

Použití této metody umožňuje provedení útoku, který vyžaduje v optimálním případě dešifrování jediného šifrového textu specifické hodnoty. Je při něm použit princip známý z útoků třídy *fault-attack*, založený na chybném výsledku jedné dílčí hodnoty dešifrování, který při následném výpočtu největšího společného dělitele určitých hodnot vede k faktorizaci veřejného modulu n. Z tohoto důvodu je použití CRT při implementaci RSA z hlediska bezpečnosti nevhodné. Konkrétní podobu útoku lze nalézt v článku [1].

8.2 Montgomeryho násobení

Montgomeryho násobení je metoda, která provádí modulární redukci součinu již v průběhu výpočtu a právě při modulárním mocnění se s jeho použitím dosahuje dobrých optimalizací časové složitosti. Bez podrobné analýzy si uveďme, jak takový algoritmus pro Montgomeryho násobení vypadá. Ohledně detailů a důkazu správnosti algoritmu čtenáře odkazujeme například na literaturu [5], ze které byla teorie čerpána.

Pro algoritmus je vybrána nějaká hodnota b, při jejímž základě jsou vyjadřovány hodnoty proměnných, a hodnota R, kterou lze obecně volit víceméně libovolně, podle hodnoty modula n jen musejí být splněny podmínky $n < R$ a $\gcd(n, R) = 1$. Pokud je však reprezentace modula n při základě b délky k, tj. $n = (n_{k-1} \dots n_1 n_0)_b$, typickou volbou bývá podle zvoleného základu b hodnota $R = b^k$. Podmínka $R > n$ je při takové volbě splněna,

avšak $\gcd(R, n) = 1$ platí pouze, pokud $\gcd(b, n) = 1$. Taková volba R tedy není možná pro libovolné n, uvažujeme-li však například právě systém RSA (či Pohlig-Hellman), n je liché a je-li b mocninou dvojky (což je přirozená volba při používané binární reprezentaci), je $R = b^k$ vyhovující. Volba $b = 2^v$ má navíc v počítačové aritmetice tu výhodu, že dělení v kroce 2(b) pak může být realizovanáno pouhým posunem (*shiftem*) do prava o v bitových pozic.

Algoritmus 12. *Montgomeryho násobení - Mont(x, y)*

VSTUP: $n = (n_{k-1} \dots n_1 n_0)_b$
$x = (x_{k-1} \dots x_1 x_0)_b$, $y = (y_{k-1} \dots y_1 y_0)_b$, kde $0 \leq x, y < n$
$n' = -n^{-1} \pmod b$
$R = b^k$, kde $\gcd(n, b) = 1$
VÝSTUP: $xyR^{-1} \pmod n$

1. $A \leftarrow 0$ /registr $A = (a_k a_{k-1} \dots a_1 a_0)_b$/
2. For $i = 0$ to $(k - 1)$ do:
   (a) $u_i \leftarrow (a_0 + \mathbf{x_i y_0}) n' \pmod b$
   (b) $A \leftarrow (A + \mathbf{x_i y} + u_i n)/b$ /celočíselné dělení beze zbytku/
3. If $A \geq n$ then $A \leftarrow A - n$
4. Return(A)

S Montgomeryho násobením lze kombinovat libovolný základní algoritmus pro modulární mocnění, zde se zaměříme opět na varianty LTOR a RTOL algoritmu. Modifikace pak vypadají tak, že vlastní tělo upravovaných algoritmů (tj. *For*-cyklus v uvedených popisech v části 3.1.1) zůstává identické, jen namísto běžného modulárního součinu nějakých hodnot x, y voláme funkci Montgomeryho násobení $Mont(x, y)$, která vrací hodnotu xyR^{-1} (poznamenejme, že hodnota R zůstává pro celý běh algoritmu modulárního mocnění stejná). Z toho vyplývá potřeba vhodně upravit počáteční hodnoty proměnných v rámci algoritmů tak, aby na konci byla získána požadovaná hodnota mocniny. Tyto inicializace si uveďme, přičemž příslušné proměnné odlišme od proměnných v základním algoritmu vlnkou, tj. pokud daný algoritmus v popisu v části 3.1.1 pracuje například s proměnnou z, zde bude tato proměnná označena jako $\tilde{z}$.

LTOR: $\tilde{z} \leftarrow R \pmod n$, $\tilde{C} \leftarrow Mont(C, R^2 \bmod n) = CR$
RTOL: $\tilde{z} \leftarrow R \pmod n$, $\tilde{y} \leftarrow Mont(C, R^2 \bmod n) = CR$

Po ukončení *For*-cyklu dostaneme hodnotu odpovídající $\tilde{z} = C^d R$, proto je nutné před výstupem provést ještě operaci $\tilde{z} \leftarrow Mont(\tilde{z}, 1) = C^d$. Celková podoba algoritmu je pak již zřejmá. Rozepíšeme-li si algoritmy (připomeňme si, že všechny operace provádíme v $\mathbb{Z}_n$), lze odvodit následující pozorování:

Pozorování 8.2.1. *Vstupními hodnotami v i-tém cyklu LTOR-Montgomery algoritmu jsou* $\tilde{C} = CR$, $\tilde{z} = C^{d'}R$, *kde* $d' = \sum_{k=i+1}^{t} 2^{k-(i+1)} d_k$.

Pozorování 8.2.2. *Vstupními hodnotami v i-tém cyklu RTOL-Montgomery algoritmu jsou* $\tilde{y} = C^{2^i}R$, $\tilde{z} = C^{\bar{d}}R$, $\bar{d} = \sum_{k=0}^{i-1} 2^k d_k$.

Realizovatelnost útoků

Montgomeryho násobení používá v rámci vlastního algoritmu běžnou operaci součinu na vstupních hodnotách x, y (viz. zvýrazněné části algoritmu). Předpokládáme-li přirozenou volbu základu b odpovídající velikosti procesorového slova, tj. 2^w, pak jsou v průběhu mocnění stále prováděny součiny na slovech uvažovaných velikostí a lze tak shodné principy útoků aplikovat i zde, jen je zapotřebí při modifikaci útoku brát v úvahu, jaké hodnoty do algoritmu vstupují. Porovnáme-li výsledky Pozorování 8.2.1 a 8.2.2 s Pozorováním 3.1.1, platným pro základní verze algoritmů LTOR a RTOL, zjistíme, že hodnoty proměnných v modifikované Montgomery-verzi jsou oproti základním algoritmům přenásobeny hodnotou R. Díky podmínce $\gcd(n, R) = 1$, s jakou musí být hodnota R volena, existuje inverzní prvek $R^{-1} \pmod{n}$ a při jeho znalosti je pak snadné dopočítat hodnoty potřebné pro útok. Pro ukázku uveďme adaptaci dvou základní útoků na RSA.

Pozorování 8.2.3. *(Chybový LTOR-Montgomery útok na RSA)*
(Modifikace Alg. 6) Zvolme náhodnou hodnotu $\tilde{C} \in \mathbb{Z}_n$ obsahující slovo b, z níž vypočteme odpovídající hodnotu šifrového textu $C = \tilde{C}R^{-1}$. Hledáme takovou hodnotu, že $C^{d'}R$ obsahuje chybové slovo a.

Pozorování 8.2.4. *(RTOL-Montgomery útok s chybovým faktorem na RSA)*
(Modifikace Alg. 8) Najděme takové $C \in \mathbb{Z}_n$, že $C_t = C^{2^t} = X$, pro které platí, že XR obsahuje chybová slova a a b. Chybový faktor pak volíme jako $\beta = \frac{X^2R}{Mont\langle XR, XR\rangle}$, neboť chyba je očekávána při výpočtu $Mont\,\langle XR, XR\rangle \neq Mont(XR, XR) = X^2R$. Pro výpočet bitů klíče pak použijme identickou rovnici jako v základním útoku.

Rozmysleme si, že odpovídají-li výsledky součinů svými vlastnostmi náhodným hodnotám, očekávaná složitost hledání textů se vůči původním

návrhům nemění. Pokud jde o porovnání pravděpodobnosti vzniku náhodných chyb, pokud by hodnota n nebo n' (hodnoty konstantní pro celý algoritmus) obsahovaly slovo b, náhodné součinové chyby by zde mohly navíc vzniknout v místech $(a_0 + x_i y_0) \cdot n'$ či $u_i \cdot n$. Avšak $(a_0 + x_i y_0)$ je nejvýše dvouslovní hodnotou a u_i dokonce jen jedním slovem, pravděpodobnost výskytu chybového součinu je proto v těchto případech zanedbatelná, a tak pravděpodobnost vzniku náhodných chyb je srovnatelná s chybovostí při použití základního školního násobení.

Je-li základ volen jako $b = 2^v$, pro možnost realizace útoku je nezbytné, aby hodnota uvažovaných chybových slov byla menší než daný základ, aby mohlo v průběhu algoritmu k násobení chybových slov docházet. Uvědomme si, že v případě $v = 1$ by dokonce byly prováděny jen samé bitové operace. Druhým podstatným předpokladem realizace útoku bylo, že hodnota R byla známa. Zabránění realizace útoku by tak zřejmě mohla být vhodně volená hodnota b nebo utajovaná, či spíše různě volená hodnota R.

8.3 Bezpečnostní schéma OAEP

Z důvodu odolnosti vůči adaptivním *chosen-ciphertext* útokům bývá ke schématu RSA připojováno kódovací schéma OAEP (nebo silnější metoda OAEP+), kdy je zpráva M před samotným šifrováním určitým způsobem překódována. Pokud kryptografické zařízení dešifruje text, jehož hodnota po dekódování nesplňuje specifikované vlastnosti, je zamítnuta a výstupem je namísto její hodnoty pouze univerzální chybové hlášení. Schéma funguje na principu jednosměrných funkcí a z předpokladu, na kterém je toto bezpečnostní schéma založeno, náhodně volený šifrový text zřejmě nebude po dešifrování odpovídat očekávané formě a bude zamítnut. Pokud tedy chceme provádět navrhované útoky vůči takovému schématu, není možné volit libovolné šifrové texty, ale je nutné postupovat přirozeně tak, že zvolíme náhodnou zprávu, na kterou použijeme schéma OAEP, a takto upravenou zprávu zašifrujeme. Tento postup je třeba opakovat tak dlouho, dokud náhodně nezískáme šifrový text C s požadovanými vlastnostmi. Struktura překódovaného textu je víceméně náhodná, výskyt chybových slov v hodnotě C lze tedy rovněž považovat za náhodný. Informace o tom, že korektně získaný šifrový text byl systémem zamítnut, signalizuje, že došlo k nějaké výpočetní chybě. Poznamenejme, že schéma OAEP a používané jednosměrné funkce typicky používají jen bitové posuny a logické funkce, případná součinová

chyba tudíž zřejmě skutečně pochází z vlastního procesu RSA-dešifrování.

Rozmysleme si, že pokud jde o složitost nalezení vstupního textu, v případě, že se volba šifrového textu v původním návrhu útoku týká přímo hodnoty textu C, složitost se při RSA-OAEP zvyšuje. Pokud je v původním útoku hledáno C jen s ohledem na nějaké vlastnosti jeho určité mocniny C^x, složitost nalezení textu pro útok na RSA-OAEP zůstává neměnná. Pokud hledaný text musí splňovat vlastnosti v hodnotách C i C^x, celková složitost je určena součinem pravděpodobností dílčích.

Schéma OAEP tak sice nedokáže zcela zamezit realizaci bug-útoků, ale díky tomu, že nelze získat přesné chybové hodnoty, nelze při použití OAEP provádět přesné útoky s chybovým faktorem, v úvahu připadají jen útoky chybové. O podobě chybového útoku pro RTOL algoritmus jsme se již zmínili v předchozí kapitole, v závěru části 7.2, jeho přesný popis lze nalézt v článku [1], odkud byla myšlenka útoků na schéma OAEP převzata. Pouze shrňme, že OAEP vyžaduje použití útoků, v nichž složitost hledání vhodných šifrových textů pro oba základní chybové útoky na RSA-OAEP (při LTOR i RTOL) je shodně určena pravděpodobností $P(a)P(b)$ pro každou iteraci útoku.

Kapitola 9

Simulace a testování

Tato kapitola pojednává o praktickém testování útoků popsaných v textu. Ačkoliv se jedná o útoky založené na typicky hardwarových chybách, testovány byly softwarově, tj. softwarovou simulací chybového dešifrovacího zařízení. Nicméně je zřejmé, že takový postup není z hlediska ověřování realizovatelnosti útoků nijak v rozporu s podstatou věci. Podívejme se, na základě čeho byly voleny vlastnosti modelu dešifrovacího zařízení, za jakých podmínek bylo testování útoků prováděno a s jakými výsledky.

Dešifrovací zařízení, w-bitový procesor, byl v souladu s předpokladem simulován kódem provádějícím LTOR, resp. RTOL algoritmus, v rámci něhož dochází k chybovému víceslovnímu násobení. Použitím takového modelu při testování je dosaženo reálné situace, kdy součinové chyby nenastávají jen v přesně očekávaných okamžicích výpočtu dešifrované hodnoty, ale dochází k nim rovněž (s příslušnou pravděpodobností) náhodně. Inspirací pro volbu dalších rysů modelového procesoru byl článek [7]. Výpočty ve schématech asymetrické kryptografie bývají v praxi zrychlovány použitím specializovaných kryptografických koprocesorů, nicméně výsledkem daného článku bylo na základě implementací a provedených analýz zjištění, že provozování kryptografie s veřejným klíčem je schůdné i na malých výpočetních zařízeních bez hardwarového zrychlení (konkrétně bylo testováno RSA-1024 a RSA-2048 na 8-bitových mikroprocesorech) a při provádění víceslovního násobení metodou klasického školního násobení. Na základě těchto výsledků byl proto při volbě modelu testovacího procesoru zvolen rovněž pouze jednoduchý typ dešifrovacího zařízení, který víceslovní součin realizuje jako školní násobení (konkrétně byla zvolena přirozená implementační metoda *Row-Wise*, nic-

méně použitý způsob implementace nijak neovlivňuje průběh součinových chyb ani hodnotu výstupu) a uvažovány byly i malé délky procesorových slov ($w = 8$). Otázkou bylo, jakou metodu zvolit v rámci jednoslovní aritmetiky, pro výpočty typu $w \times w$ bitů.

Článek [1], ze kterého byla převzata myšlenka bug-útoků, vychází z předpokladu modelové situace, že je používán procesor, který provádí chybný součin na jedné ze všech $2^w \times 2^w$ možných dvojic slov. K reálnosti takové situace uvádí, že „ je-li například použita 64×64-bitová násobička (*multiplier*), existuje 2^{128} různých dvojic vstupů, u nichž je výpočetně nemožné ověřit správnost všech výsledků a lze dokonce předpokládat, že součin většiny z těchto dvojic ani nikdy nebude na daném procesoru proveden.“ Přitom „díky zvětšující se délce procesorových slov a stále propracovanějším optimalizacím součinových jednotek v moderních procesorech je stále více pravděpodobnější, že mohou tyto procesory obsahovat nějakou neodhalenou chybu“ a ilustruje podobnou situaci na případu *Pentium division bugu* z 90. let 20. století, kdy byl náhodně objeven bug v dělicí jednotce Pentiumčipu. Nicméně konkrétní případ součinového bugu článek neuvádí, jedná se tedy spíše jen o hypotézu. Navíc poznamenejme, že při větším w zase už i provádění některých navržených útoků leží na hranici výpočetních možností.

Snahou proto bylo i zjistit, zda je z praktického hlediska skutečně možné, aby na součinovém hardwaru procesoru bylo možné realizovat „zřídka se vyskytující“ bug, a to navíc blížící se ideálu jedné chybové dvojice. Jako zdroj přehledu hardwarových realizací součinu byla použita literatura [8]. Zmíněný Pentium division bug byl způsoben tím, že v čipu implementovaný algoritmus *radix*-4 *SRT* používal pomocnou vyhledávací tabulku hodnot uloženou v paměti, z níž pět možných vstupů z tisícovky bylo vynecháno, a nesprávně tak byly příslušné hodnoty považovány za nulové. S určitým typem „tabulkové metody“ se můžeme setkat i při praktické realizaci výpočtů v případě násobení, a to, když při používání *higher-radix* metody, při které je z důvodu zrychlení procesu vyhodnocován najednou větší počet bitů, je zároveň použito tzv. *Boothovo překódování*, kterým se redukuje velikost přičítaných násobků během výpočtu. Pro určování těchto násobků se právě rovněž používá vyhledávací tabulka. Nicméně tato higher-radix metoda není používána v takovém stupni, aby tabulka mohla dosahovat obdobných rozměrů jako ve výše uvedeném případě, a v důsledku toho ani chyba způsobená jedním chybným záznamem nenastává při náhodných výpočtech s dostatečně malou, resp. použitelně malou, pravděpodobností.

Zdá se přirozené, že komplikovanější operace, v jejichž praktické imple-

mentaci jsou výrazné snahy optimalizace, poskytují pro „umístění“ bugu větší prostor, než jaký je u jednoduchých, resp. jednoduše prováděných operací. Jako „složitější“ varianta součinové jednotky byl testován hardwarový návrh dle článku [9], založený na stromové struktuře. Ani zde se však při hledání vhodného umístění bugu nepodařilo způsobit použitelně malou chybovost a i v nejúspěšnějším případě byla dosažená chybovost daného zařízení stále zhruba 40 000-krát vyšší, než při velikostně srovnatelném modelu násobičky, která způsobuje chybný součin právě jedné dvojice slov. Navíc obdržené výsledky vyvolaly otázku, zda jsou tyto složitější součinové jednotky pro realizaci bugu vůbec vhodné, zda nejsou už příliš komplexní. Obecně lepších výsledků bylo totiž naopak dosahováno při umisťování bugů do pomocných součtových jednotek (*adderů*).

Z důvodu, že se experimentálně nepodařilo nalézt vhodný součinový bug ani žádný článek o takovém případě, byl namísto specifikace součinového zařízení pro simulaci zvolen jen obecný kód, vracející výsledek násobení jednotlivých slov jako *„black box“*.

Při testování byly vzhledem k běžným parametrům dnešních procesorů uvažovány hodnoty $w = 16, 32, 64$ (délka procesorových slov) a podle rozsahu dlouhých čísel užívaných v dnešní asymetrické kryptografii bitové délky $|n| = 512, 1024, 2048, 4096$. Jak ukázala analýza v kapitole 4, při $w = 8$ dochází k nepoužitelně vysoké chybovosti zařízení, což prokázalo i praktické testování. Softwarová simulace byla realizována v programu Wolfram Mathematica 7.0 (Windows), s použitím vestavěných optimalizovaných funkcí pro řešení určitých výpočetních problémů jako: PowerModList[ ] (pro výpočet druhých odmocnin v $\mathbb{Z}_p$), FactorInteger[ ] (pro faktorizaci dlouhých čísel), GCD[ ] (pro výpočet největšího společného dělitele celých čísel). Pro výpočet libovolných odmocniny v $\mathbb{Z}_p$ byl implementován AMM algoritmus (kap. 2).

Všechny útoky uvedené v této práci byly na výše popsaném modelovém zařízení při nějakých parametrech w, n odzkoušeny, poznamenejme, že vždy při parametrech volených s ohledem na složitost generování šifrových textů v daném útoku. Vzhledem k tomu, že práce se podrobně zabývá modelovou situací jen jedné chybové dvojice, byly útoky rovněž testovány pouze za takových podmínek. Důležité výsledky realizací jednotlivých útoků jsou uvedeny dále. Jako ukázku si lze simulaci jednoho z testovaných útoků, RTOL-útok na RSA (Alg. 8), prohlédnout na přiloženém CD.[1]

[1]Program lze rovněž stáhnout na webových stránkách věnovaných konferenci Wolfram Technology Conference 2010 jako součást prezentace RNDr. Antonína Slavíka, Ph.D., na adrese <http://www.wolfram.com/events/techconf2010/speakers_p3.html>.

Výsledky testování

Všechny útoky převzaté z článku [1] (uvedené v kapitole 5) byly implementovány s použitím nutných úprav popsaných v kapitole 7. Z nich návrh útoku na schéma Pohlig-Hellman (Alg. 5) je vlastně jediným útokem, který se nepodařilo úspěšně realizovat v celém rozsahu, tj. odkrýt celou hodnotu soukromého klíče při žádných z uvažovaných parametrů. Návrh podle článku [1] vychází z předpokladu, že v $\mathbb{Z}_p$ můžeme počítat libovolné odmocniny. Jako vhodný pro velká čísla byl v této práci mezi zvažovanými algoritmy vybrán algoritmus AMM, nicméně i tato metoda není obecně výpočetně snadná. Algoritmus k výpočtu d'-té odmocniny potřebuje výpočet faktorizace hodnoty $\gcd(p-1, d')$, což je pro velká čísla obecně těžký problém (a číslo $p-1$ ve schématech předpokládáme velké a hodnota d' odpovídá hodnotě odkryté části klíče, která se tak v každé iteraci útoku významně zvyšuje). Při testování se dařilo útok realizovat pouze zhruba do odhalení $2^8 = 256$ bitů klíče, rozhodující byla právě složitost faktorizace.

V útoku s chybovým faktorem na schéma Pohlig-Hellman (Alg. 7) je předpokládán výpočet 2^i-tých odmocnin, kde hodnota i dosahuje až bitové délky klíče. Při uvažovaných délkách byla v tomto případě naopak úspěšně testována i pouhá rekurzivní metoda výpočtu druhých odmocnin.

Modifikovaný algoritmus pro chybový útok na RSA (Alg. 9) redukující vliv náhodných chyb byl využit při testování s parametry $w = 16$, $|n| = 1024$, který při volbě $k = l = 10$ vedl k úšpěšnému výpočtu tajného klíče, naopak pro nižší k útok nebyl bezchybný a detekce špatně určených bitů nefungovala dobře. Pro volbu parametru k se tak ukázal jako dobrý odhad $P^k < \frac{1}{(\log n)k}$. Pro $w = 32$ použití této modifikované verze už nebylo nutné (i podle vzorce platí $k = 1$) a stačil útok v základní podobě (Alg. 6). Naopak eliminace náhodných chyb (podle Alg. 10) vůbec nebyla použita v RTOL-útocích s chybovým faktorem (Alg. 7, 8), jako dostačující se u nich ukázalo používání změny volby parametru r (Pozorování 6.2.2).

Úspěšně, tj. s výpočtem hodnoty klíče v celém rozsahu při uvažovaných parametrech, byly otestovány i všechny vlastní návrhy jako modifikace LTOR útoku pro odlišnou implementaci (z části 7.1.1), LTOR-útok s chybovým faktorem (Alg. 11) a útok při použití Montgomeryho násobení pro variantu LTOR i RTOL (viz. Pozorování 8.2.3 a 8.2.4).

Kapitola 10

Závěr

V práci jsme uvažovali situaci, kdy je dešifrování prováděno na chybovém zařízení, a zabývali se návrhy kryptografických útoků využívajících tuto skutečnost, včetně adaptace těchto postupů na různé okolnosti. Testováním pak bylo ověřeno, že lze dané útoky obecně k úspěšnému prolomení uvažovaných schémat (RSA, Pohlig-Hellman) skutečně prakticky využít. Existující bug, způsobující chybnost výsledků, přitom v zařízení může být jak útočníkem záměrně vytvořený, tak pouze „nedopatřením“ vzniklý, nicméně útočníkem odhalený. Nejspíš o žádném zařízení, které neumožňuje otestování veškerých možných situací, které mohou během jeho fungování nastat, nelze s jistotou tvrdit, že nějaký bug využitelný pro provedení útoku neobsahuje. Závěrem je proto vhodné shrnout získané výsledky a vyvodit na jejich základě nějaká bezpečnostní doporučení eliminující potenciální nebezpečí, která je v praxi dobré zavést jako prevenci proti takovýmto útokům. V kapitole 8 jsme například viděli, že používání schématu OAEP redukuje aplikovatelné útoky jen na chybové, navíc dokonce obecně zvyšuje celkovou složitost útoků na RSA co do generování textů i počtu prováděných iterací, jeho používání tedy nepochybně zvyšuje bezpečnost systému.

Ovšem možnou cestou zabezpečení může být třeba i jen používání co nejjednodušších zařízení. Například používání 8-bitových procesorů lze zřejmě zvolit jako jednoduché opatření proti bugovým útokům (v souladu s výsledky článku [7], který potvrzuje možnost provozování asymetrické kryptografie na takových procesorech), neboť se dá předpokládat, že jakýkoliv bug v součinové jednotce takových parametrů by se při provozu velmi brzy projevil (viz. prakticky stoprocentní chybovost 8-bitového procesoru při chybně prováděném součinu byť jen jediné dvojice slov). Na procesorech s větší

délkou slov by bylo možné simulovat nízkobitové procesory implementací Montgomeryho násobení, a to prostřednictvím volby základu b. Je zřejmé, že výpočetní možnosti procesoru by pak nebyly plně využívány, ale bylo by možné tento princip v rámci prevence útoku použít i jinak, například tak, aby bylo dešifrování prováděno vždy s jinou virtuální délkou procesorového slova, čímž by byly hodnoty proměnných v průběhu algoritmu děleny do slov předem neznámé délky. V takovém případě je pak obtížnější volit texty tak, aby splňovaly vlastnosti nezbytné pro útok. Snižováním délek zároveň redukujeme rozsah chybových slov použitelných pro útok. Viděli jsme navíc, že i variabilní volba hodnoty R by mohla znesnadnit realizaci útoků.

Na druhou stranu i volby dostatečně velkých délek procesorových slov mohou bránit provedení většiny z uvedených útoků. Délky slov ovlivňují zejména útoky na schéma RSA, jejichž složitost je výrazně určována hledáním textů metodou náhodné volby, která se zvyšující se délkou slov roste. Způsoby volby textů a odpovídající pravděpodobnosti jejich nalezení shrňme pro přehled v následující tabulce.[1]

realizace RSA	bez OAEP		s OAEP	
varianta útoku	vlastnosti textu	složitost	vlastnosti textu	složitost
LTOR-RSA	$b \in C, a \in C^x$	$P(a)$	$b \in C, a \in C^x$	$P(a)P(b)$
RTOL-RSA	$a, b \in C^x$	$P(a \cap b)$	$a \in C^y, b \in C^x$	$P(a)P(b)$

Představu o výpočetní náročnosti útoků lze získat porovnáním příslušných pravděpodobností s konkrétními hodnotami uvedenými v tabulkách v části 4.2.2. Veškeré útoky se například při 128-bitových slovech stávají podle odhadů výpočetních možností (viz. část 3.1.2) neproveditelné. Proto by i volba velkých délek procesorových slov mohla sloužit jako bezpečnostní opatření, ovšem jedná se spíše o opatření pravděpodobnostní, protože při vyšším počtu chybových dvojic bude složitost nalezení textů klesat. Přitom se tím dostáváme do situace, kdy skutečně není výpočetně možné otestovat všechny stavy a ověřit tak správné fungování součinové jednotky.

Pokud jde o návrhy útoků pro schéma Pohlig-Hellman, předpokládá se, že složitost generování vstupních šifrových textů v jejich případě závisí „jen“ na složitosti algoritmu pro výpočet odmocnin v $\mathbb{Z}_p$. Jak ale ukázalo praktické

[1] Za útoky bez OAEP lze dosadit libovolný útok uváděný v této práci určený pro RSA bez přídavných schémat a optimalizací, nezávisle na implementačním pořadí či typu útoku. Útokem s OAEP máme vždy na mysli chybový útok. Chybový RTOL-útok pro RSA-OAEP v této práci nebyl explicitně uveden, detailní popis lze nalézt v článku [1].

testování LTOR-útoku, ani AMM algoritmus, který je poměrně vhodný pro výpočet libovolných odmocnin při velkých číslech, není pro rozsah odmocnin počítaných v rámci tohoto útoku dostačující a jiný, principielně odlišný algoritmus, který by nepoužíval faktorizaci ať už celých čísel, nebo polynomů, se v dostupné literatuře nalézt nepodařilo. Příslušný postup tak lze zřejmě použít pro odhalení jen určité části klíče. Zdůrazněme, že tento výpočetní problém se netýká RTOL-útoku, který vyžaduje hledání 2^i-tých odmocnin, které je i při velkých číslech dobře realizovatelné. Pokud však i u schématu Pohlig-Hellman uvážíme použití OAEP, nelze pak již v žádném z případů využít hledání textů prostřednictvím odmocňování a schéma tím ztrácí slabinu „usnadněného“ generování textů.

Schéma OAEP je tedy v mnohých ohledech jednoznačně vhodné používat a protože probírané útoky obecně vyvolávají chybné výstupy dešifrování, které zařízení při aplikaci OAEP detekuje, je jako další opatření vhodné při větším počtu zamítnutých textů pozastavit další dešifrování, z důvodu podezření na prováděný útok. Systém této detekce by bylo samozřejmě teoreticky možné zavést i bez použití OAEP, pokud by dešifrovací zařízení svůj výstup samo zpětně kontrolovalo vůči hodnotě šifrového textu.

Algoritmy LTOR a RTOL způsobují prakticky shodnou chybovost zařízení a rovněž jsme ukázali, že samotná volba algoritmu nijak neurčuje druh útoku, který je v jejím případě možné použít - vždy lze zkonstruovat útok chybový i s chybovým faktorem. Z těchto hledisek tedy například na výběru implementovaného algoritmu nezáleží. Avšak používání RTOL algoritmu s sebou přináší narozdíl od LTOR dvě velká bezpečnostní nebezpečí. Tím prvním je při možnosti použití útoku s chybovým faktorem (není-li například použito bezpečnostní schéma OAEP) značně redukovaný počet iterací, který je nezbytný k výpočtu celého klíče. Navíc se jedná o neadaptivní útok, kde volba textů není nijak ovlivněna hodnotou dešifrovacího klíče, proto si lze veškeré šifrové texty připravit v předvýpočtu, a k tomu lze při shodné chybové dvojici tyto texty použít pro odhalení libovolného dalšího klíče. Druhou bezpečnostní slabinou RTOL algoritmu je, že umožňuje provedení varianty útoku s odkrýváním bitů klíče od nejnižších. To má následující závažný důsledek. Konkrétně pro schéma RSA totiž existuje dle článku [6] způsob, jak za určitých podmínek ze znalosti části klíče dopočítat jeho zbývající bity. Teoreticky tak nemusí být nutné provádět útoky v celém rozsahu a právě při znalosti jen čtvrtiny nejnižších bitů článek uvádí metodu, jak dopočítat celý klíč. RTOL algoritmus tedy umožňuje, ať při chybovém útoku nebo

útoku s chybovým faktorem, zjištění hodnoty klíče i při výrazně omezeném počtu provedených iterací útoku. Naopak všechny ostatní útoky uvedené v této práci odkrývají hodnoty bitů klíče směrem od nejvyšších a tomu v daném článku odpovídá metoda, jejíž použití vyžaduje znalost takové části klíče, která není výrazně nižší než celá jeho délka. Aplikace metody, která by tedy přicházela v úvahu v ostatních případech, konkrétně při použití LTOR, nedává výrazné zlepšení oproti provedení úplného útoku. Implementace LTOR algoritmu je tudíž z bezpečnostního hlediska evidentně vhodnější.

Práce ověřila potenciální bezpečnostní hrozbu způsobovanou hardwarovými chybami a snadnou zranitelnost kryptografických schémat provozovaných na takových chybových zařízeních. Otázkou však zůstává, zda je uvažovaná situace vůbec reálná, neboť se pro ni nepodařilo najít konkrétní podklady, které by dokazovaly možnost vzniku součinových chyb v hardwarových jednotkách s odpovídajícími uvažovanými charakteristikami. Pokud je taková situace možná, hledání vhodného umístění bugu by zřejmě vyžadovalo odbornější technickou znalost nebo v případě hardwarových součinových jednotek stromové struktury možná alespoň podrobnější matematickou analýzu. Specifikace součinového bugu, a to jakékoliv povahy, by přitom nejen konkretizovala možnou podobu útoků, ale mohla by zároveň i pomoci ve stanovení bezpečnostních opatření nejen proti realizaci bugových útoků jako takových, ale rovněž proti umístění samotného bugu v zařízení. Uvažování existence součinového bugu však zůstává i nadále jen hypotézou, i na tu je ale při implementaci kryptografických schémat nepochybně vhodné brát zřetel. Proto doporučujeme přijetí alespoň základních bezpečnostních opatření jako je používání schématu OAEP při šifrování, realizace modulárního mocnění prostřednictvím algoritmů třídy LTOR a nepoužívání CRT pro výpočetní optimalizaci.

Literatura

[1] Biham E., Carmeli Y., Shamir A. (2008): Bug Attacks. *Advances in Cryptology - CRYPTO* 2008, *LNCS* 5157, Springer, 221-240.

[2] Adleman L., Manders K., Miller G. (1997): On taking roots in finite fields. *FOCS* 1997, 175-178.

[3] Bach E., Shallit J. (1996): Algorithmic Number Theory, Vol. 1: Efficient Algorithms (Foundations of Computing). MIT Press, London.

[4] Cohen H. (1996): A course in Computational Algebraic Number Theory. 3. corr. print., Springer, 37-38.

[5] Menezes A., van Oorschot P., Vanstone S. (1996): Handbook of Applied Cryptography. CRC Press, 87-132, 592-634.

[6] Boneh D., Durfee G., Frankel Y. (1998): Exposing an RSA Private Key Given a Small Fraction of its Bits. *Advances in Cryptology - ASIACRYPT* 1998, *LNCS* 1514, Springer, 25-34. Dostupný z www: <http://crypto.stanford.edu/∼dabo/papers/bits_of_d.ps>

[7] Gura N., Patel A., Wander A., Eberle H., Shantz S. Ch. (2004): Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs. *CHES* 2004, *LNCS* 3156, Springer, 119-132.

[8] Hennessy J., Patterson D. (1990): Computer architecture: a quantitative approach. 4th ed. (2007), Elsevier, San Francisco. Appendix I: Computer Arithmetic, by Goldberg D.

[9] Example 16x16 Multiplier [online]. *Advanced VLSI Design, ECEN* 6263, Oklahoma State University [cit. 2010-10-28]. Dostupný z www: <http://lgjohn.okstate.edu/5263/lectures/mult16.pdf>