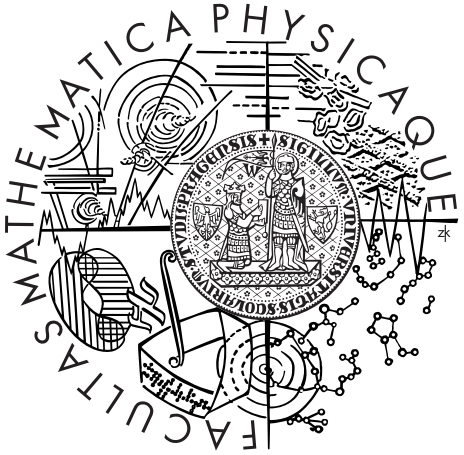
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Jakub Töpfer

Rotační kryptoanalýza ARX šifer

Katedra algebry

Vedoucí bakalářské práce: RNDr. Michal Hojsík, Ph.D.
Studijní program: Matematika
Studijní obor: obecná matematika

Praha 2012

Chtěl bych poděkovat svému vedoucímu RNDr. Michalu Hojsíkovi, Ph.D. za mnohé konzultace a za spoustu užitečných rad a připomínek k podobě práce.

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

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

V Praze dne 24. května 2012 Jakub Töpfer

Název práce: Rotační kryptoanalýza ARX šifer

Autor: Jakub Töpfer

Katedra: Katedra algebry

Vedoucí bakalářské práce: RNDr. Michal Hojsík, Ph.D., Katedra algebry

Abstrakt: Tato práce za zabývá funkcemi, které lze vyjádřit pomocí sčítání, XORu a rotace (ARX). Případně ještě povolíme přičtení či XOR konstanty (ARX+C). Nejprve zkoumáme tyto funkce z teoretického pohledu. Ukážeme, že pomocí operací ARX+C umíme zapsat každou funkci. Můžeme si dokonce dovolit vypustit operaci XOR. Naopak pomocí jiných kombinací zkoumaných operací všechny funkce nezískáme. Nabízíme též jednoduchý algoritmus určující, jestli lze funkci zapsat pomocí sčítání a XORu.
Následně prezentujeme metodu rotační kryptoanalýzy určenou právě pro ARX funkce. Ukážeme její podobu na zjednodušených variantách šifer Threefish, TEA a XTEA a diskutujeme, pro které šifry je metoda vhodná. Zabýváme se též úpravou metody v podobě rotačního rozpínavého útoku, jehož užití opět demonstrujeme na Threefish.

Klíčová slova: ARX šifry, úplnost ARX, AX funkce, rotační kryptoanalýza, rotační rozpínavý útok

Title: Rotational cryptanalysis of ARX ciphers

Author: Jakub Töpfer

Department: Department of Algebra

Supervisor: RNDr. Michal Hojsík, Ph.D., Department of Algebra

Abstract: We investigate functions which can be realized using only addition, XOR and rotation (ARX). Sometimes we admit operations with constants as well (ARX+C). We start our research from a theoretical point of view. We prove that every function can by written using only ARX+C operations and that we can even omit XOR. On the other hand, other combinations of these operations do not generate all functions. We also present an algorithm determining whether a function can be realised only by addition and XOR.
After that we present a rotational cryptanalysis, which is designed especially for ARX ciphers. We demonstrate this method on reduced variants of Threefish, TEA and XTEA ciphers and discuss for which ciphers it is suitable. The last part of this thesis deals with a modification of rotational cryptanalysis called rotational rebound attack and shows its application on Threefish.

Keywords: ARX cryptosystems, completeness of ARX, AX functions, rotational cryptanalysis, rotational rebound attack

Obsah

Úvod

Kryptografické primitivy založené pouze na modulárním sčítání, operaci XOR a bitové rotaci jsou v symetrické kryptografii poslední dobou velmi oblíbené. Uplatňují se při konstrukci blokových šifer, jako jsou například RC5 či Threefish, proudových šifer (např. Salsa 20) i hašovacích funkcí. Z nich můžeme jmenovat například funkce z rodiny MD (MD4, MD5) a jejich nástupce SHA-1 a SHA-2.

Několik takových kryptosystémů nalezneme i mezi finalisty soutěže na nový standard SHA-3 (např. Skein nebo Blake). Jejich velkou výhodou je především rychlost plynoucí z dobré hardwarové podpory všech požadovaných operací.

Obecně se předpokládá, že kombinací těchto operací (souhrnně je budeme označovat zkratkou ARX) lze dostat kvalitní kryptografické primitivy.

Prvním z cílů této práce je tyto operace důkladně prozkoumat a zjistit, jaké funkce lze s užitím jednotlivých operací získat. Ukážeme, že pokud k modulárnímu sčítání, XORu a rotaci přidáme ještě možnost přičtení konstanty (funkce využívající pouze tyto operace nazveme ARX+C), umíme vytvořit zcela libovolnou funkci. To samé platí dokonce i když vynecháme XOR. Naopak žádnou další z operací již vynechat nemůžeme.

Prozkoumáme také detailně funkce založené pouze na sčítání a XORu. Přesně popíšeme, jaké funkce dokážeme pomocí těchto operací sestrojit. Navíc nabídneme jednoduchý algoritmus na zjištění, zda funkci lze v této podobě zapsat.

Po teoretičtějším úvodu přijde řada na popis rotační kryptoanalýzy. Jedná se o kryptoanalytický útok aplikovatelný právě na primitivy využívající pouze ARX, případně ARX+C. Základem útoku je odhalení určité pravidelnosti v chování šifry, pokud ji zadáme nějaký vstup a pak ten samý vstup zrotovaný. Právě to je častým nedostatkem ARX šifer.

Velkou překážku pro použití rotační kryptoanalýzy představují konstanty. Ukážeme, že ale i s nimi se občas dokážeme vyrovnat. Obzvláště pokud nejsou zvoleny tak, aby se chovaly dostatečně náhodně. Tak tomu bude v případě prvního prezentovaného útoku na blokovou šifru Threefish, která je využívána v hašovací funkci Skein. Nabídneme též užití rotační kryptoanalýzy pro šifry TEA a XTEA a pokusíme se o shrnutí, pro které druhy šifer je rotační kryptoanalýza vhodná.

V poslední kapitole se potom budeme věnovat úpravě rotační kryptoanalýzy v podobě rotačního rozpínavého útoku. Pomocí něj dokážeme odhalit nepravidelnost v chování Threefish při mnohem větším počtu rund než v případě užití normální rotační kryptoanalýzy.

Většina práce je převážně rešeršního charakteru a vychází především z článků [1, 2, 3]. Původní jsou sekce 1.5, 3.1.4, 3.2 a elementární důkaz lemmatu 2.1. Některé další důkazy a zdůvodnění (např. důsledku 1.9) jsou pak rozepsány podrobněji, než tomu bylo v původních článcích.

1. Kryptosystémy ARX

První kapitola se bude zabývat obecně kryptosystémy využívajícími pouze modulární sčítání, XOR a bitovou rotaci, případně ještě přičítání konstanty. Budeme zkoumat, které funkce lze pomocí těchto operací sestrojit a které ne.

1.1 Základní pojmy

Předpokládáme, že čtenář zná základní pojmy z oblasti kryptografie. Zde tedy jen v krátkosti shrneme to nejdůležitější a zavedeme značení, které budeme využívat. Formální a podrobné vysvětlení všech potřebných pojmů lze nalézt například v knize Douglase Stinsona [4].

Je mnoho různých druhů šifer. My se budeme zabývat převážně šiframi blokovými. *Blokovou šifrou* pro nás bude nějaká funkce, které na vstupu zadáme otevřený text a šifrovací klíč. Ona nám pak vrátí zašifrovaný text. Navíc k této funkci existuje ještě funkce inverzní, která ze zašifrovaného textu a klíče vyrobí původní otevřený text.

Bloková šifra může v principu vypadat různě. Často ale k šifrování dochází několikanásobným opakováním velmi podobných operací. Jednomu opakování operací říkáme *runda*.

Proudové šifry se od blokových liší v tom, že nejdříve pomocí klíče vytvoří dlouhou posloupnost znaků nazývanou *proud klíče*. A až pomocí této posloupnosti šifrují otevřený text. Toto šifrování pak už obvykle není příliš složité.

V této práci se budeme zabývat téměř výhradně šiframi, které jsou založeny na třech jednoduchých operacích: sčítání, XORu a rotaci. V jejich značení se budeme řídit zavedenými zvyklostmi. Přesto ho zde ale pro jistotu uvádíme.

Značení. Mějme množinu celých čísel $\{0, 1, \ldots, q-1\} = \mathbb{Z}_q$, kde $q = 2^n$ je mocnina dvojky. Na této množině pro $a, b \in \mathbb{Z}_q$ uvažujeme následující operace

- *(Modulární) sčítání* modulo q budeme značit $a + b$. Tuto operaci budeme též občas označovat zkratkou ADD, ve schematech ji potom budeme značit pomocí symbolu $\boxplus$.
- *XOR* neboli sčítání po bitech modulo 2 označíme $a \oplus b$. Na operaci XOR se můžeme dívat jako na sčítání ve vektorovém prostoru $\mathbb{Z}_2^n$.
- *Rotaci* o r bitů vpravo budeme zapisovat obecně $\ggg_r$, konkrétní zrotovanou proměnnou pak $a \ggg_r$. Pokud bude hodnota r jasná z kontextu, budeme rotaci zapisovat zjednodušeně jako $\overrightarrow{a}$. Analogicky pro rotaci vlevo.

Prvkům $\mathbb{Z}_q$ budeme říkat *slova*. Jednotlivé cifry při zápisu slov v dvojkové soustavě budeme pak nazývat *bity*. Pojmy slovo a bit mají dobrý význam z hlediska počítačové reprezentace čísel.

V dalších částech této kapitoly se budeme zabývat funkcemi, které jsou realizovatelné s využitím pouze několika operací. V označení těchto funkcí se budou vyskytovat písmena A, R, X a C symbolizující, které operace jsou ve funkcích povoleny. A (neboli addition) značí modulární sčítání, R (rotation) rotaci, X operaci

XOR a C možnost nevyužívat ve funkci pouze zadané proměnné, ale i předem stanovené konstanty.

Definice. Řekneme, že funkce $f : \mathbb{Z}_2^n \to \mathbb{Z}_2$ je zapsána v *algebraické normální formě (ANF)*, pokud je tvaru

$$f = \bigoplus_{I \subseteq \{1,\ldots,n\}} a_I \prod_{i \in I} x^i,$$

kde $a_I \in \mathbb{Z}_2$.

Platí, že každou funkci $f : \mathbb{Z}_2^n \to \mathbb{Z}_2$ lze zapsat v algebraické normální formě. Tento zápis je navíc jednoznačný.

1.2 Úplnost ARX+C

Hned první prezentované tvrzení je svým způsobem překvapující. Ukážeme, že pomocí operací ADD, XOR, rotace a konstanty 1 dokážeme vytvořit libovolnou funkci nad $\mathbb{Z}_q$. Tuto pozoruhodnou větu dokázali ve svém článku [1] pánové Khovratovich a Nikolić.

Definice. Nechť $\mathcal{F}$ je množina funkcí. Řekneme, že množina $\mathcal{Q}$ *generuje* $\mathcal{F}$, pokud každou funkci z $\mathcal{F}$ můžeme vyjádřit pomocí složení funkcí z $\mathcal{Q}$.

Věta 1.1 (úplnost ARX+C)**.** Nechť $q = 2^n$, $k \in \mathbb{N}$ a nechť

$$\mathcal{F}_k = \{f : \mathbb{Z}_q^k \to \mathbb{Z}_q\}.$$

Pak množina funkcí $\{+, \oplus, \ggg_1, 1\}$ generuje $\mathcal{F}_k$.

Poznámka. Mimo výše uvedených funkcí v souladu s původním článkem předpokládáme, že máme ještě k dispozici funkci $\Xi : \mathbb{Z}_q^k \to \mathbb{Z}_2^{nk}$, která všechna vstupní slova spojí do jednoho řetězce, a funkci $\Pi : \mathbb{Z}_2^{nk} \to \mathbb{Z}_2^n$, která vezme ze slova délky nk pouze n posledních bitů.

Důkaz. Budeme postupovat konstruktivně. Ukážeme, jak vytvořit libovolnou funkci $f \in \mathcal{F}_k$. Všech k vstupních slov si zapíšeme pomocí Ξ za sebe a vytvoříme tak jedno vstupní slovo tvořené $d = nk$ bity. Nechť $X = (x_{d-1}, \ldots, x_0)$ a $Y = (y_{d-1}, \ldots, y_0)$ jsou dvě d-bitová slova. Postupně budeme konstruovat složitější a složitější funkce.

1. Vytvoříme projekce $P_i : \mathbb{Z}_2^d \to \mathbb{Z}_2^d$, $P_i(X) = 00\ldots0x_i$ pro libovolné $i \in \{0, \ldots, d-1\}$.
   Využijeme skutečnosti, že násobení 2 odpovídá shiftu o jedna vlevo. Nejdříve slovo zrotujeme o i vpravo pomocí i jednoduchých rotací. Nyní je bit x_i na poslední pozici ve slově. Následně slovo vynásobíme 2^{d-1} tak, že ho 2^{d-1}-krát sečteme samo se sebou. Tím dostaneme slovo $x_i00\ldots0$. Teď už stačí slovo zrotovat $d-1$-krát vpravo a jsme hotovi.

2. Vytvoříme funkce $M_i(X, Y) = 00\ldots0(x_iy_i)$. Tedy součin dvou funkcí, které mají nejvýše jeden nenulový bit.
   Zkonstruujeme funkce $P_i(X)$ a $P_i(Y)$ a ty následně sečteme. Hodnota x_iy_i odpovídá přenosu z poslední pozice při sčítání, neboli druhému bitu zprava výsledku. Stačí tedy vzít $P_1(P_i(X) + P_i(Y))$.

3. Vytvoříme funkce $B_C(X) = \begin{cases} 00\ldots01 & \text{pro } X = C \\ 00\ldots00 & \text{jinak.} \end{cases}$

   Nechť $C = (c_{d-1}, \ldots, c_0)$. Potom

   $$B_C(X) = \prod_{i=0}^{d-1} (x_i \oplus c_i \oplus 1).$$

   Funkce $x_i \oplus c_i \oplus 1$ mají nejvýše jeden nenulový bit, umíme je tedy násobit podle druhého bodu.

4. Zkonstruujeme funkce $V_{C_1C_2}(X) = \begin{cases} C_2 & \text{pro } X = C_1 \\ 0 & \text{jinak} \end{cases}$

   pro d-bitové konstanty C_1 a C_2.
   Platí $V_{C_1C_2}(X) = C_2 \cdot B_{C_1}(X)$. Násobení konstantou C_2 realizujeme pomocí sčítání.

5. Označme si F funkci f, jejíž výstup je zleva doplněn pomocí Ξ nulami na délku d. Platí

   $$F = \bigoplus_{X \in \mathbb{Z}_q^k} V_{X,F(X)}.$$

   Libovolnou funkci $f : \mathbb{Z}_q^k \to \mathbb{Z}_q$ vytvoříme jako $f = \Pi \circ F$. □

Tato věta mám mimo jiné říká, že nemůžeme najít nějakou obecnou vlastnost společnou pro ARX+C kryptosystémy. Libovolný kryptosystém totiž můžeme považovat za ARX+C. Ačkoli jeho zápis jen pomocí modulárního sčítání, XORu, rotace a přičítání konstanty může být často hodně složitý.

Z toho bychom mohli usuzovat, že kryptoanalytické útoky na ARX+C systémy prezentované dále jsou použitelné pro libovolnou šifru. To je svým způsobem pravda. Uvidíme ale, že útoky budou použitelné pouze pro systémy s relativně malým počtem modulárních sčítání a omezeními pro konstanty.

1.3 Úplnost AR+C

Jistě přirozenou otázkou je, jestli opravdu všechny operace z ARX+C potřebujeme. Jestli neexistuje nějaký menší generátor všech výše zkoumaných funkcí.

Zkusme se zamyslet, jak by mohl vypadat. Určitě potřebujeme mít v generátoru konstantu — jinak bychom neměli jak dosáhnout funkce vracející konstantní jedničku. A také potřebujeme modulární sčítání. Bez něho si nemáme jak poradit s násobením. Ze všech tří operací jen pomocí něho totiž můžeme dosáhnout součinu bitů (jakožto přenosu při sčítání).

Samotné modulární sčítání a konstanta nám také určitě nestačí. V již zmíněném článku [1] je ukázáno, že když k nim přidáme ještě rotaci, tak ale už dokážeme vytvořit libovolnou funkci. K tomu nám bude stačit vyjádřit operaci XOR pomocí zbývajících.

Věta 1.2 (úplnost AR+C)**.** Nechť opět $q = 2^n$, $k \in \mathbb{N}$ a

$$\mathcal{F}_k = \{f : \mathbb{Z}_q^k \to \mathbb{Z}_q\}.$$

Pak množina funkcí $\{+, \ggg_1, 1\}$ generuje $\mathcal{F}_k$.

Důkaz. Dle věty 1.1 stačí vyjádřit XOR pomocí ADD a rotace. Uvažujme opět dvě d-bitová slova $X = (x_{d-1}, \ldots, x_0)$ a $Y = (y_{d-1}, \ldots, y_0)$. Stejně jako v důkazu předchozí věty zkonstruujeme funkce $P_i(X) = 00\ldots0x_i$ pro libovolné $i \in \{0, \ldots, d-1\}$. Pomocí rotace vpravo vytvoříme z $P_i(x)$ funkce $Q_i(x) = x_i00\ldots0$. Pokud sečteme $Q_i(X)$ a $Q_i(Y)$, dostaneme na nejlevější pozici $x_i \oplus y_i$ a jinde nuly. Po zrotování na i-tou pozici tak dostáváme funkce $S_i(X, Y) = 00\ldots0(x_i \oplus y_i)00\ldots0$. Nyní si stačí uvědomit, že

$$X \oplus Y = S_{d-1}(X, Y) + S_{d-2}(X, Y) + \cdots + S_0(X, Y)$$

a jsme hotovi. □

1.4 AX funkce

Jiné podskupiny operací ARX+C už všechny funkce nad $\mathbb{Z}_q$ negenerují (to uvidíme dále). I tak může být ale užitečné se jimi zabývat a zkoušet je co nejlépe popsat. Obzvláště zajímavá je otázka, jak poznat, jestli lze danou funkci zapsat jenom pomocí zvolených operací.

Přesně na to se zaměřili Anton Klyachko a Ekaterina Menshova. Všechna tvrzení této části pochází z jejich článku [3]. Společně s nimi prozkoumáme, jaké funkce můžeme dostat pouze s využitím operací ADD a XOR.

Definice. Funkci $f : \mathbb{Z}_q^k \to \mathbb{Z}_q$ nazveme *AX* funkcí, pokud ji lze vyjádřit pomocí operací ADD a XOR. Formálně množina $F_{k,q}$ všech AX funkcí je nejmenší množina funkcí splňující následující dvě podmínky:

- $f(x_1, \ldots, x_k) = x_i \in F_{k,q}\ \forall i \in \{1, \ldots, k\}$
- $f, g \in F_{k,q} \Rightarrow f + g, f \oplus g \in F_{k,q}$

Ukážeme, že o funkci umíme velmi snadno rozhodnout, jestli je AX nebo ne. Algoritmus pro rozhodování bude založen na následující větě.

Ve zbytku této pasáže budeme používat trochu neobvyklé značení. To nám ale umožní se zbavit přílišného množství indexů a text tak bude přehlednější. Budeme pracovat s funkcemi k proměnných, které budeme značit $x, y, \ldots$ — tři tečky zde neobvykle znamenají jen konečné pokračování. Tyto proměnné budou z okruhu $\mathbb{Z}_q$, kde $q = 2^n$ je mocnina dvojky. Můžeme tedy jejich hodnotu zapsat ve dvojkové soustavě. Jednotlivé bity budeme označovat dolními indexy.

Definice. Buď i je formální parametr. Mějme polynom v proměnných

$$x_i, y_i, \ldots; x_{i-1}, y_{i-1}, \ldots; x_{i-2}, y_{i-2}, \ldots; \ldots.$$

Jeho *váha* je rovna maximální váze jeho monomu.
Váhu monomu určíme jako součet vah proměnných, které ho tvoří.
Váha proměnné s indexem $i-l$ (např. x_{i-l}) je rovna 2^{-l}. Závisí tedy na vzdálenosti indexu proměnné od nejvyššího použitého indexu.

Věta 1.3 (popis AX)**.** Funkce $f : \mathbb{Z}_q^k \to \mathbb{Z}_q$ je AX právě tehdy, když pro každé $i \in \{0, \ldots, n-1\}$ lze i-tý bit její hodnoty vyjádřit ve tvaru

$$(f(x, y, \ldots))_i = g(x_i, y_i, \ldots; x_{i-1}, y_{i-1}, \ldots; x_{i-2}, y_{i-2}, \ldots; \ldots), \tag{1.1}$$

kde bity se záporným indexem uvažujeme jako nulové a g je funkce nad $\mathbb{Z}_2$ v algebraické normální formě bez absolutního členu nezávislá na i, jejíž váha je nejvýše jedna.

Příklad. Využijme značení z předchozí věty. Pro $q = 8$ a $k = 1$ existují právě čtyři monomy v algebraické normální formě, jejichž váha je nejvýše jedna. Jsou to x_i, x_{i-1}, x_{i-2} a $x_{i-1}x_{i-2}$. Můžeme tedy utvořit 2^4 polynomů v ANF, jejichž váha je nejvýše jedna.

Například AX funkce odpovídající polynomu v ANF $x_i \oplus x_{i-1}x_{i-2}$ je tvaru $f(x) = f(x_2, x_1, x_0) = (x_2 \oplus x_1x_2, x_1 \oplus x_0x_{-1}, x_0 \oplus x_{-1}x_{-2}) = (x_2 \oplus x_1x_2, x_1, x_0)$, což odpovídá funkci s funkčními hodnotami $f(0) = 0$, $f(1) = 1$, $f(2) = 2$, $f(3) = 7$, $f(4) = 4$, $f(5) = 5$, $f(6) = 6$ a $f(7) = 3$.

Nezbývá než se pustit do důkazu. Ten ale nebude tak jednoduchý. A tak se na něj musíme připravit. Nejdříve si větu 1.3 popisující AX funkce přeformulujeme tak, aby se nám pak lépe dokazovala.

Značení. Označme si symbolem $\odot$ součin po bitech modulo 2 neboli konjunkci. Operaci $\odot$ můžeme chápat jako na součin po složkách ve vektorovém prostoru $\mathbb{Z}_2^n$.

Věta 1.4 (alternativní popis AX)**.** Funkce $f : \mathbb{Z}_q^k \to \mathbb{Z}_q$ je AX funkcí právě tehdy, když ji lze zapsat ve tvaru

$$f(x, y, \ldots) = \bigoplus_i \left((2^{k_{i,1}}x) \odot (2^{k_{i,2}}x) \odot \cdots \odot (2^{l_{i,1}}y) \odot (2^{l_{i,2}}y) \odot \ldots \right), \tag{1.2}$$

kde pro každé i platí nerovnost $2^{-k_{i,1}} + 2^{-k_{i,2}} + \cdots + 2^{-l_{i,1}} + 2^{-l_{i,2}} + \cdots \leq 1$.

Zdůvodnění ekvivalence vět 1.3 a 1.4. Funkci g z věty 1.3 můžeme zapsat jako XOR součinu bitů zadaných proměnných. Přitom do hodnoty i-tého bitu nemohou zasahovat bity na vyšších pozicích. To přesně odpovídá podmínce (1.2) z věty 1.4. Funkce f je tu zapsaná jako XOR součinu posunutých zadaných proměnných. Pokud se podíváme na i-tý bit funkce f, zjistíme, že se na něm mohou podílet (kvůli přenásobení vhodnou mocninou 2) bity na i-té pozici a na pozicích nižších.

Podmínka pro $k_{i,j}$ z věty 1.4 přesně odpovídá požadavku na váhu polynomu ve větě 1.3.

Nyní si již není těžké uvědomit, že podmínka (1.2) nemůže připouštět žádnou funkci, kterou (1.1) nepřipouští a naopak. □

Pro důkaz si musíme připravit několik pomocných lemmat.

Definice. Nechť $x, y \in \mathbb{Z}_q$. Potom prvek $[x, y] = x \oplus y \oplus (x + y)$ nazveme *komutátorem* prvků x a y. Komutátor je rozdíl mezi $\oplus$ a $+$ dvou prvků: jeho i-tý bit $[x, y]_i$ je roven přenosu na i-tou cifru při modulárním sčítání.

Lemma 1.5. Jednotlivé bity komutátoru splňují

$$[x, y]_i = x_{i-1}y_{i-1} \oplus [x, y]_{i-1}(x_{i-1} \oplus y_{i-1}) \tag{1.3}$$

Důkaz. Přenos $c_i = [x, y]_i$ pro i-tý bit je roven 1, pokud většina z bitů x_{i-1}, y_{i-1}, c_{i-1} (tedy alespoň dva) je rovna 1. V opačném případě je $c_i = 0$. Tento bit můžeme tedy zapsat pomocí vztahu

$$c_i = x_{i-1}y_{i-1} \oplus y_{i-1}c_{i-1} \oplus c_{i-1}x_{i-1} = x_{i-1}y_{i-1} \oplus c_{i-1}(x_{i-1} \oplus y_{i-1}),$$

což jsme chtěli dokázat. □

Poznámka. Vztah (1.3) můžeme přepsat do tvaru

$$[x, y] = 2(x \odot y \oplus [x, y] \odot (x \oplus y)).$$

Užitím distributivity násobení dvěma vzhledem k $\odot$ a $\oplus$ a distributivity $\odot$ vzhledem k $\oplus$ ho pak můžeme dále upravit na

$$(2x) \odot (2y) = [x, y] \oplus (2[x, y]) \odot (2x) \oplus (2[x, y]) \odot (2y). \tag{1.4}$$

Pojem komutátoru nám ale stačit nebude. Musíme si ho zobecnit.

Definice. *Násobným komutátorem složitosti n* rozumíme formální výraz v proměnných $x, y, \ldots$ definovaný induktivně podle následujících pravidel

- každá proměnná je násobný komutátor složitosti 1,
- výraz $[u, v]$ je násobný komutátor složitosti n, pokud u a v jsou násobné komutátory a součet jejich složitostí je n.

Definice. *Hloubku $d(w)$* násobného komutátoru w je definujeme

- $d(x) = 0$ pro x proměnnou,
- $d([u, v]) = \max\{d(u), d(v)\} + 1$.

Příklad. $[a, [[b, [c, d]], [e, f]]]$ je násobný komutátor složitosti 6 a hloubky 4.

Poznámka. Indukcí můžeme snadno (přímo z definice) dokázat, že pokud je jedna z proměnných obsažených v násobném komutátoru rovna 0, je roven 0 celý komutátor.

Lemma 1.6. Nechť w je násobný komutátor. Pokud $i < d(w)$, tak i-tý bit w je roven nule.

Důkaz. Využijeme indukci podle hloubky komutátoru. Pro hloubku 0, je tvrzení triviálně splněné. Předpokládejme, že $d(u)$ nejnižších bitů násobného komutátoru u a $d(v)$ nejnižších bitů násobného komutátoru v jsou nulové. Dle vztahu (1.3) pak $\max\{d(u), d(v)\}+1$ bitů násobného komutátoru $[u, v]$ je nulových. Tím máme indukční krok hotový. □

Lemma 1.7. Hloubka násobného komutátoru složitosti alespoň 2^n je alespoň n.

Důkaz. Tvrzení dokážeme indukcí podle n. Násobný komutátor složitosti 1 má hloubku 0. Násobný komutátor složitosti alespoň 2^n pro $n \geq 1$ je tvaru $w = [u, v]$. Alespoň jeden z násobných komutátorů u, v má složitost alespoň 2^{n-1}. Z indukčního předpokladu je tedy hloubka tohoto komutátoru alespoň $n - 1$. Odtud již plyne, že hloubka w je alespoň n. □

Lemma 1.8. V $\mathbb{Z}_q$ jsou všechny násobné komutátory složitosti alespoň q nulové.

Důkaz. Podle lemmatu 1.7 je hloubka uvažovaného násobného komutátoru alespoň $\log_2 q$. Tedy dle lemmatu 1.6 je prvních $\log_2 q$ bitů nulových. Násobný komutátor nad $\mathbb{Z}_q$ má ale právě $\log_2 q$ bitů. A proto jsou nulové všechny jeho bity. □

Nyní se už konečně můžeme pustit do vlastního důkazu.

Důkaz věty 1.3. Označme M množinu všech funkcí $\mathbb{Z}_q^k \to \mathbb{Z}_q$ tvaru (1.1). Nyní chceme ukázat, že každá funkce z $F_{k,q}$ (tedy AX funkce) patří do M. A naopak, že každá funkce z M patří do $F_{k,q}$.

Důkaz první inkluze nebude příliš obtížný. Funkce $x, x \oplus y, x + y$ leží v M (i-tý bit $x + y$ záleží na x_i, y_i a přenosu z nižších řádů). Stačí tedy ukázat, že složením funkcí z M dostaneme opět funkci z M. Mějme $F, f \in M$, potom

$$\begin{aligned}(F(x, y, \dots))_i &= G(x_i, y_i, \dots; x_{i-1}y_{i-1} \dots; \dots),\\ (f(s, t, \dots))_i &= g(s_i, t_i, \dots; s_{i-1}t_{i-1} \dots; \dots),\end{aligned}$$

kde G a g jsou funkce v algebraické normální formě váhy nejvýše jedna bez absolutního členu.

Pro složení funkcí pak platí

$$\begin{aligned}(F(f(s, t, \dots), y, \dots))_i &= G((f(s, t, \dots))_i, y_i, \dots; (f(s, t, \dots))_{i-1}, y_{i-1}, \dots; \dots)\\ &= G(g(s_i, t_i, \dots; s_{i-1}, t_{i-1}, \dots; \dots), y_i, \dots;\\ &\qquad g(s_{i-1}, t_{i-1}, \dots; s_{i-2}, t_{i-2}, \dots; \dots), y_{i-1}, \dots; \dots).\end{aligned}$$

Po roznásobení takto získáme polynom v ANF bez absolutního členu. Váha polynomu nepřekročí 1, protože váha polynomu $g(s_i, t_i, \dots; s_{i-1}, t_{i-1}, \dots; \dots)$ je nejvýše jedna, váha polynomu $g(s_{i-1}, t_{i-1}, \dots; s_{i-2}, t_{i-2}, \dots; \dots)$ nejvýše $1/2$, atd. Tím je první inkluze dokázána.

Pro důkaz druhé inkluze využijeme popis z věty 1.4. O něm již víme, že je ekvivalentní dokazované větě. Stačí nám dokázat, že každý výraz ve tvaru (1.2),tj.

$$f(x, y, \dots) = \bigoplus_i \left((2^{k_{i,1}}x) \odot (2^{k_{i,2}}x) \odot \dots \odot (2^{l_{i,1}}y) \odot (2^{l_{i,2}}y) \odot \dots\right),$$

kde pro každé i platí nerovnost $2^{-k_{i,1}} + 2^{-k_{i,2}} + \dots + 2^{-l_{i,1}} + 2^{-l_{i,2}} + \dots \leq 1$, lze vyjádřit pomocí $\oplus$ a $+$.

Dokážeme silnější tvrzení. Ukážeme, že každý výraz tvaru

$$f = (2^k u) \odot (2^l v) \odot (2^m w) \odot \dots \tag{1.5}$$

kde $2^{-k} + 2^{-l} + 2^{-m} + \dots \leq 1$ a u, v, w jsou násobné komutátory, lze vyjádřit pomocí $\oplus$ a $+$.

Pro spor předpokládejme, že to není možné. Potom umíme najít výraz ve tvaru (1.5), který není možné vyjádřit pomocí $\oplus$ a $+$, pro který platí rovnost

$$2^{-k} + 2^{-l} + 2^{-m} + \dots = 1. \tag{1.6}$$

Platí totiž, že $1 - (2^{-k} + 2^{-l} + 2^{-m} + \dots)$ je zlomek tvaru $\frac{s}{2^k}$, kde k je maximum z čísel $k, l, m, \dots$. Proto výraz

$$f \odot \underbrace{(2^k u) \odot (2^k u) \odot \dots \odot (2^k u)}_{s \text{ činitelů}}$$

představuje stejnou funkci jako f a z nerovnosti se stává rovnost.

Vyberme si ze všech funkcí, které nejsou vyjádřitelné pomocí $\oplus$ a $+$, splňujících vztah (1.5) i rovnost (1.6) všechny s minimálním počtem činitelů. Z nich pak ty s maximální složitostí násobných komutátorů $u, v, w, \ldots$. Taková funkce f existuje dle lemmatu 1.8. To říká, že násobné komutátory příliš velké složitosti nemá smysl uvažovat.

Funkce f má alespoň dva činitele. Jinak by totiž bylo možné ji vyjádřit pomocí $\oplus$ a $+$ podle definice násobného komutátoru. Z rovnosti (1.6) plyne, že dva největší z exponentů $k, l, m, \ldots$ jsou stejné. Nechť BÚNO $k = l$.

Pomocí vztahu (1.4) dostaneme

$$\begin{aligned}
&(2^k u) \odot (2^k v) = (2 \cdot 2^{k-1} u) \odot (2 \cdot 2^{k-1} v) \\
&= [2^{k-1}u, 2^{k-1}v] \oplus (2[2^{k-1}u, 2^{k-1}v]) \odot (2 \cdot 2^{k-1}u) \oplus (2[2^{k-1}u, 2^{k-1}v]) \odot (2 \cdot 2^{k-1}v) \\
&= 2^{k-1}[u, v] \oplus (2^k[u, v]) \cdot (2^k u) \oplus (2^k[u, v]) \cdot (2^k v)
\end{aligned}$$

Výraz (1.5) tedy můžeme psát jako součet tří členů

$$\begin{aligned}
f = \big((2^{k-1}t) \odot (2^m w) \odot \ldots\big) \oplus \big((2^k t) \odot (2^k u) \odot (2^m w) \odot \ldots\big) \oplus \\
\oplus \big((2^k t) \odot (2^k v) \odot (2^m w) \odot \ldots\big), \text{ kde } t = [u, v].
\end{aligned}$$

Všechny tři členy zřejmě splňují rovnost (1.6).

První člen představuje AX funkci, protože počet jeho činitelů je menší, než má funkce f. A ta byla z funkcí, které nejsou AX vybrána tak, aby měla počet členů minimální možný.

Druhý a třetí člen mají stejně činitelů jako f, ale vyšší součet složitostí, protože složitost $t = [u, v]$ je o jedna vyšší než součet složitostí u a v. Proto i tyto členy představují AX funkce kvůli výběru f.

Vyjádřili jsme tedy f jako XOR tří AX funkcí, takže f je nutně AX. To je ale spor s volbou f. Námi požadovaná funkce tedy nemůže existovat. Tím je důkaz ukončen. □

Pomocí právě dokázané věty můžeme přesně určit, kolik AX funkcí existuje. Důkaz následujícího tvrzení je uveden ve výrazně podrobnější podobě než byl prezentován v původním článku [3].

Důsledek 1.9 (počet AX funkcí)**.** Počet různých AX funkcí k proměnných nad $\mathbb{Z}_q$ je přesně

$$2^{\frac{1}{k!}\left(\frac{q}{2}+1\right)\left(\frac{q}{2}+2\right)\ldots\left(\frac{q}{2}+k\right)-1}$$

Důkaz. Označme $q = 2^n$. Uvažujme nejdříve případ $k = 1$. Ukážeme, že v tomto případě máme přesně $q/2$ monomů váhy nejvýše jedna. K tomu využijeme jednoznačnost zápisu čísla ve dvojkové soustavě. Váhu jedna má monom x_{n-1}. Monom s nejmenší váhou je $x_0 = x_{(n-1)-(n-1)}$. Jeho váha je rovna $2^{-(n-1)} = 2/q$. Umíme sestrojit právě monomy s váhou $s \cdot (2/q)$, kde $s \in \{1, 2, \ldots, q/2\}$. Konkrétně se jedná o monomy

$$x_{n-1-l_1} x_{n-1-l_2} \ldots x_{n-1-l_p}, \text{ kde } s = 2^{n-1-l_1} + 2^{n-1-l_2} + \cdots + 2^{n-1-l_p}.$$

Obdobný postup můžeme použít o pro obecné k. Počet monomů váhy nejvýše jedna odpovídá počtu uspořádaných k-tic nezáporných celých čísel $(r_1, \ldots, r_k)$

se součtem nejvýše $q/2$. Hodnota $r_i \cdot 2/q$ zde odpovídá váze vzhledem k i-té proměnné.

Zbývá počet k-tic spočítat. Označme si $t = q/2$. Naší úloze odpovídá situace, že máme t bodů (příspěvků 1 do celkového součtu) a ty se snažíme rozdělit pomocí „přepážek." Hodnotě r_1 odpovídá počet bodů, které jsou před první přepážkou, hodnotě r_2 počet bodů mezi první a druhou přepážkou, atd. Až hodnotě n odpovídá počet bodů mezi $n-1$-ní a n-tou přepážkou. Body za n-tou přepážkou nevyužijeme. To odpovídá tomu, že celkový součet může být menší než t. Umístění přepážek může být libovolné (i více vedle sebe), jen nesmí být všechny před prvním bodem. V takovém případě bychom totiž dostali nulový monom.

Mějme t bodů. Postupně k nim budeme přidávat přepážky. Mezi t body (včetně okrajů) máme $t+1$ pozic, pro umístění první přepážky máme proto $t+1$ možností. Druhou přepážku umisťujeme mezi $t+1$ symbolů (t bodů a jednu přepážku), pro její umístění máte tedy $t+2$ možností. Stejně pokračujeme dále. Dostáváme tak celkem $(t+1)(t+2)\dots(t+k)$ možností. U výsledné polohy přepážek nám ale nezáleží na pořadí, v jakém jsme je umisťovali. Proto musíme výsledek ještě podělit počtem různých uspořádání přepážek, což je $k!$. A nesmíme zapomenout odečíst speciální variantu, kdy jsou všechny přepážky před prvním bodem (v libovolném pořadí).

Dostáváme tak, že počet k- tic je je roven

$$\frac{(\frac{q}{2}+1)(\frac{q}{2}+2)\dots(\frac{q}{2}+k)}{k!} - 1.$$

Všechny funkce dostaneme tak, že libovolně kombinujeme monomy váhy nejvýše jedna. Odtud již plyne tvrzení věty. □

Hlavně ale můžeme sestrojit slibovaný algoritmus určující, zda je funkce AX či nikoli.

Algoritmus 1.10 (AX kritérium)**.**

vstup: funkce $f : \mathbb{Z}_q^k \to \mathbb{Z}_q$, kde $q = 2^n$
výstup: ANO, pokud funkce f je AX, NE jinak

1. Vyjádři nejlevější bit $(f(x, y, \dots))_{n-1}$ funkce f jako polynom v proměnných $x_{n-1}, y_{n-1}, \dots; x_{n-2}, y_{n-2}, \dots; \dots$ v ANF. Nechť to je $g_{n-1}(x_{n-1}, y_{n-1}, \dots; x_{n-2}, y_{n-2}, \dots;$. Ověř, že váha tohoto polynomu (pro $i = n-1$) je nejvýše jedna a absolutní člen je nulový. Pokud ne, odpověz NE, jinak pokračuj.

2. V polynomu g_{n-1} proveď následující substituce:

$$\begin{aligned} &x_{n-1} \to x_{n-2}, x_{n-2} \to x_{n-3}, \dots, x_1 \to x_0, x_0 \to 0, \\ &y_{n-1} \to y_{n-2}, y_{n-2} \to y_{n-3}, \dots, y_1 \to y_0, y_0 \to 0, \dots \end{aligned} \tag{1.7}$$

   a ověř, že takto získaný polynom g_{n-2} je polynom odpovídající $(n-2)$-hému bitu funkce f. Pokud ne, odpověz NE, jinak pokračuj.

⋮

$n-1$. V polynomu g_2 proveď substituce (1.7) a ověř, že takto získaný polynom g_1 dává stejné hodnoty jako druhý bit zprava funkce f. Pokud ne, odpověz NE, jinak pokračuj.

n. V polynomu g_1 proveď substituce (1.7) a ověř, že takto získaný polynom g_0 odpovídá nejpravějšímu bitu funkce f. Pokud ne, odpověz NE, jinak odpověz ANO.

Správnost algoritmu plyne přímo z věty 1.3.

Poznámka. Dle algoritmu vidíme, že například funkce vracející součin dvou čísel modulo q není AX. Algoritmus se zastaví už v prvním kroku kvůli váze polynomu g. Na druhou stranu funkce $x \mapsto xy$ je AX pro každé pevné $y \in \mathbb{Z}_q$.

Toť práce Antona Klyachka a Ekateriny Menshové. S ohledem na předchozí části se nabízí otázka, jak je to s funkcemi AX+C. Můžeme pomocí sčítání XORu a konstant zapsat libovolnou funkci?

1.5 Neúplnost AX+C

Pomocí algoritmu 1.10 můžeme snadno odvodit, že rotaci o jedna vpravo (a ani žádnou jinou rotaci) nelze zapsat jako AX funkci. Nemůžeme proto postupovat obdobně jako v případě AR funkcí a dokázat tvrzení o úplnosti AX+C.

Naopak, existují funkce, které pomocí AX+C zapsat neumíme. Jedna z nich je například zmíněná rotace o jedna vpravo.

Věta 1.11 (neúplnost AX+C)**.** Nechť $q = 2^n$, $k \in \mathbb{N}$ a

$$\mathcal{F}_k = \{f : \mathbb{Z}_q^k \to \mathbb{Z}_q\}.$$

Pak množina funkcí $\{+, \oplus\} \cup K$, kde $K \subseteq \mathbb{Z}_q$ jsou libovolné konstanty, negeneruje všechny funkce z $\mathcal{F}_k$.

Důkaz. Ukážeme, že pomocí výše popsané množiny funkcí neumíme zapsat funkci $\ggg_1$. Pomocí algoritmu 1.10 ověříme, že pomocí funkcí ADD a XOR rotaci o jedna vpravo zapsat nelze. Důvod je ten, že poslední bit potřebujeme přesunout na první pozici.

A s tímto problémem nám nepomůže ani případné zapojení konstant. Pokud konstantu ke slovu XORuji, pouze měním hodnoty bitů, ne ale jejich polohu. Při přičítání konstanty se mi může stát, že bit posunu o jedna vlevo. Ale to pouze za podmínky, že hodnota bitu je 1. Pokud je bit roven 0, mohu opět měnit pouze jeho hodnotu. Konstanty nám tedy v úsilí rotovat bity (univerzálně, pro všechny možné vstupy) nijak nepomohou. □

Nyní máme již zcela zodpovězenou otázku ohledně generátorů všech funkcí $f : \mathbb{Z}_q^k \to \mathbb{Z}_q$ využívajících pouze sčítání, XOR, rotaci a konstanty. Víme, že všechny funkce lze zapsat pomocí ARX+C a AR+C. Z věty 1.11 a úvah před větou 1.2 naopak plyne, že jiné možnosti, jak zapsat všechny funkce jen pomocí požadovaných operací, nemáme.

2. Rotační kryptoanalýza

Hlavním tématem této práce je rotační kryptoanalýza. V této kapitole ji prozkoumáme z teoretického pohledu. V následující kapitole se pak pustíme do její aplikace na konkrétní šifry.

2.1 Základní principy

Nejdříve se zaměříme pouze na ARX schémata nevyužívající předem stanovené konstanty. Hlavním metodou rotační kryptoanalýzy je zkoumání dvojic slov, z nichž jedno je rotací druhého.

Definice. Fixujme $\ggg$ jako otočení o r bitů vpravo, $x, y \in \mathbb{Z}_q$, $q = 2^n$. Dvojici $(x, \overrightarrow{x})$ pak nazveme *rotačním párem*.

Nechť $\diamond$ je libovolná operace na $\mathbb{Z}_q$. Řekneme, že operace $\diamond$ *zachovává* rotační pár, pokud zobrazení $\ggg$ je endomorfismem algebry $\mathbb{Z}_q(\diamond)$. V případě binární operace to znamená, že

$$\overrightarrow{x \diamond y} = \overrightarrow{x} \diamond \overrightarrow{y}.$$

Budeme zkoumat, pro které operace $\diamond$ je $\ggg$ endomorfismus $\mathbb{Z}_q(\diamond)$. Tedy které operace zachovávají rotační páry. Případně, pokud je nezachovávají vždy, tak s jakou pravděpodobností.

Rotační pár určitě zachovává každá transformace, které se vyhodnocuje zvlášť na jednotlivých bitech. Tedy speciálně také operace XOR. Snadno si můžeme rozmyslet, že rotační pár zachovává i rotace.

Trochu složitější situace je u modulárního sčítání. Následující lemma pochází z práce Magnuse Dauma [5]. V ní požadované tvrzení vyplyne jako důsledek hlouběji budované teorie. My nabízíme radši vlastní elementární důkaz.

Lemma 2.1. Nechť $q = 2^n$ a nechť $x, y \in \mathbb{Z}_q$. Uvažujme rotaci o r bitů vpravo. Pak platí

$$\Pr\left[\overrightarrow{x + y} = \overrightarrow{x} + \overrightarrow{y}\right] = \frac{1}{4}\left(1 + 2^{r-n} + 2^{-r} + 2^{-n}\right) \tag{2.1}$$

Důkaz. Označme si standardně bity x zleva $x_{n-1}, \ldots, x_0$, bity y pak $y_{n-1}, \ldots, y_0$. Bity ve výsledku jsou závislé na příslušném bitu z x, z y a na přenosu z nižšího řádu. Pokud výsledek označíme v, po bitech $v_{n-1}, \ldots, v_0$, pak $v_i = x_i \oplus y_i \oplus c_i$, kde přenos $c_i = x_{i-1}y_{i-1} \oplus c_{i-1}(x_{i-1} \oplus y_{i-1})$.

Při rotaci vlastně sčítaná slova rozdělíme na dvě části, části prohodíme a pak sčítáme klasicky zprava od nejnižších bitů.

$$\begin{array}{cccc|cccccc} & & & & \multicolumn{6}{c}{\longleftrightarrow} \\ x_{n-1} & x_{n-2} & \ldots & x_r & x_{r-1} & x_{r-2} & x_{r-3} & \ldots & x_1 & x_0 \\ y_{n-1} & y_{n-2} & \ldots & y_r & y_{r-1} & y_{r-2} & y_{r-3} & \ldots & y_1 & y_0 \end{array}$$

Rotační pár se nezachová, pokud se při prohození částí pokazí nějaký přenos. Tedy pokud $c_r = 1$ nebo $c_n = 1$. Jinak, tedy právě, když platí $c_r = 0$ a $c_n = 0$, bude pár zachován.

Stačí tedy pro $k \in \{1, \ldots n\}$ spočítat $\Pr[c_k = 0] = 1 - \Pr[c_k = 1]$. Vzorec odvodíme induktivně. Aby se dostal přenos do vyššího řádu, musí být alespoň dvě

z čísel x_i, y_i, c_i rovny jedné. Snadno rozebereme všechny možné případy. Uvažujme nezrotované proměnné x, y. Platí $c_0 = 0$, tedy $\Pr[c_1 = 1] = 1/4$. Dále

$$\begin{aligned}
\Pr[c_2 = 1] &= \frac{1}{4} + \frac{1}{2}\Pr[c_1 = 1] = \frac{1}{4}\left(1 + \frac{1}{2}\right), \\
\Pr[c_3 = 1] &= \frac{1}{4} + \frac{1}{2}\Pr[c_2 = 1] = \frac{1}{4}\left(1 + \frac{1}{2} + \frac{1}{4}\right), \\
&\vdots \\
\Pr[c_k = 1] &= \frac{1}{4} + \frac{1}{2}\Pr[c_{k-1} = 1] = \frac{1}{4}\left(1 + \frac{1}{2} + \frac{1}{4} + \cdots + \frac{1}{2^{k-1}}\right) = \\
&= \frac{1}{4}\left(1 + 1 - 2^{-(k-1)}\right) = \frac{1}{2}\left(1 - 2^{-k}\right).
\end{aligned}$$

Tedy $\Pr[c_k = 0] = 1 - \Pr[c_k = 1] = 1 - \frac{1}{2}\left(1 - 2^{-k}\right) = \frac{1}{2}\left(1 + 2^{-k}\right)$.

Máme vzorec pro pravděpodobnost nulového přenosu pro zprava libovolně vzdálený bit. Nyní si stačí uvědomit, že pokud víme, že přenos na nějakém bitu byl roven nule, začínáme měřit vzdálenost až od tohoto bitu (situace je ekvivalentní jako bychom na tomto bitu začínali a neměli žádnou informaci navíc).

Můžeme tedy dle věty o podmíněné pravděpodobnosti dopočítat

$$\begin{aligned}
&\Pr[\, c_n = 0 \quad \& \quad c_r = 0\,] = \Pr[\, c_n = 0 \mid c_r = 0\,] \cdot \Pr[\, c_r = 0\,] = \\
&= \frac{1}{2}\left(1 + 2^{-(n-r)}\right) \cdot \frac{1}{2}\left(1 + 2^{-r}\right) = \frac{1}{4}\left(1 + 2^{-r} + 2^{-n+r} + 2^{-n}\right).
\end{aligned}$$

A to je přesně (2.1). □

Obvykle nás budou zajímat hodnoty pravděpodobnosti pro velká n a malá r. Označme pro pevně zvolené r

$$p_r = \lim_{n\to\infty} \Pr\left[\, \overrightarrow{x+y} = \overrightarrow{x} + \overrightarrow{y}\,\right].$$

Hodnoty p_r pro malá r zachycuje následující tabulka.

r	p_r	$\log_2(p_r)$
1	0.375	−1.415
2	0.313	−1.676
3	0.218	−1.831

Pro velká n a rostoucí r se hodnota $\Pr\left[\, \overrightarrow{x+y} = \overrightarrow{x} + \overrightarrow{y}\,\right]$ snižuje až do $r = n/2$, kdy se blíží k $1/4$. Poté se symetricky opět zvyšuje.

Zabývejme se nyní obecným ARX schématem, tedy funkcí obsahující pouze sčítání modulo $q = 2^n$, XOR a rotace. Pro něj nyní můžeme spočítat pravděpodobnost zachování rotačního páru.

Pro jednoduchost budeme předpokládat, že pravděpodobnost zachování rotačního páru při každém sčítání je nezávislá na tom, jestli byl zachován při ostatních sčítáních.

Věta 2.2 (pravděpodobnost zachování rotačního páru, [1])**.** Uvažujme libovolné ARX schéma $\mathcal{S}$. Buď a počet jeho modulárních sčítání a I vstup do schématu. Označme $\overrightarrow{I}$ vstup zrotovaný o r bitů vpravo. Potom za předpokladu nezávislosti zachování rotačního páru při jednotlivých operacích platí $\mathcal{S}(\overrightarrow{I}) = \overrightarrow{\mathcal{S}(I)}$ s pravděpodobností alespoň $(p_r)^a$.

Důkaz. Dle lemmatu 2.1 se pro pevné r pravděpodobnost zachování rotačního páru při sčítání pro rostoucí n snižuje. Proto pro žádné n nebude nižší než p_r.

Operace XOR a rotace zachovávají rotační pár vždy. Každé sčítání s pravděpodobností alespoň p_r.

Tvrzení dokážeme snadnou indukcí dle počtu sčítání a. □

Pro náhodnou funkci P do $\mathbb{Z}_2^t$ je pravděpodobnost $P(\overrightarrow{I}) = \overrightarrow{P(I)}$ pro náhodný vstup I rovna 2^{-t}. Tedy pokud ARX funkci lze zapsat pomocí nejvýše q modulárních sčítání, kde $(p_r)^q > 2^{-t}$, tak se nechová zcela náhodně. Právě toho rotační kryptoanalýza využívá.

Například pro $r = 1$ lze využít rotační kryptoanalýzu na každou ARX funkci, kterou lze zapsat pomocí méně než $t/1.415$ sčítání.

Celý útok probíhá v modelu umožňujícím získat šifrové texty, které vznikly z otevřeného textu zašifrováním pomocí transformovaného klíče (related-key scenario) [6]. V našem případě transformaci představuje bitová rotace.

To je jistě velmi silný předpoklad. A také výrazný nedostatek popisovaného útoku. Na druhou stranu i takové útoky jsou občas velmi účinné a nebezpečné. Například ve stejném modelu byly uskutečněny útoky na proudovou šifru RC4 stojící za algoritmem WEP [7] pro šifrování v bezdrátových sítích.

Předpokládáme, že všechny vstupy tvoří rotační páry. Pokud první klíč a otevřený text mají hodnoty $(k_0, \ldots, k_{m-1})$ a $(p_0, \ldots, p_{n-1})$, tak druhý (zpárovaný) klíč a otevřený text jsou $(\overrightarrow{k_0}, \ldots, \overrightarrow{k_{m-1}})$ a $(\overrightarrow{p_0}, \ldots, \overrightarrow{p_{n-1}})$.

Postupně zkoušíme různé vstupy a sledujeme, zda zašifrovaný vstup a zrotovaný vstup zašifrovaný pomocí stejně zrotovaných klíčů tvoří rotační pár. Pokud má šifra dostatečně malý počet sčítání, můžeme předpokládat že objevený rotační pár nevznikl náhodou, ale díky zachování rotačního páru při všech sčítáních. To nám dává informaci o určitých bitech přenosu při všech sčítáních. Společně se znalostí vstupu a šifrovacího schematu pak dokážeme odhalit určité bity klíče. K tomu potřebujeme, aby k nějakému sčítání došlo tak brzy, abychom dokázali všechny transformace vstupního textu až do tohoto sčítání snadno simulovat.

Následně stačí klíč vhodně zrotovat a celý postup opakovat. Názornější bude postup u konkrétních aplikací v následující kapitole.

2.2 Konstanty

Existuje celá řada útoků na iterační schemata s identickými rundami. Proto se často používají pro různé rundy rozdílné rundovní konstanty. To techniku rotační kryptoanalýzy značně komplikuje. Stále je ale možné ji využít.

Definice. Uvažujme rotaci o pevné r. Nechť $X, Y \in \mathbb{Z}_q$ jsou dvě slova. Výraz

$$E(X, Y) = \overrightarrow{X} \oplus Y$$

pak nazýváme *rotační chybou*.

Zřejmě $E(X, \overrightarrow{X}) = 0$. Rotační chyba popisuje stav, kdy rotační pár kvůli přičtení konstanty nevytvoří rotační výstup. Výstup bude pokažený právě o rotační chybu.

Ne každé přičtení či XOR konstanty ale musí chybu způsobit. Například pokud konstanta $C = \overrightarrow{C}$, tak $\overrightarrow{X} \oplus C = \overrightarrow{X \oplus C}$ a pravděpodobnost zachování rotačního páru při sčítání se řídí lemmatem 2.1. Právě takového stavu se budeme snažit dosáhnout.

Také se může stát, že se rotační chyba vyruší sousedním modulárním sčítáním. To je pravděpodobnější pro konstanty s menší Hammingovou váhou (tedy pro takové, které mají co nejméně nenulových bitů).

Při praktických útocích bývá potřeba si dopředu dopočítat (hrubou silou, například pravděpodobnostně metodou Monte Carlo) pomocné konstanty, které nepříznivé působení konstant v šifře co nejvíce zmírní.

I tento postup je ale použitelný pouze pro konstanty s malou Hammingovou váhou (například čítač rund). Dobře zvolená konstanta útok pomocí rotační kryptoanalýzy prakticky znemožňuje.

Pro příznivější konstanty postupujeme následovně:

Předpokládejme, že máme otevřený text i klíč rozdělený na bloky o velikosti $q = 2^n$. Předpočítáme si konstanty d_i a e_i. Pro klíč K pak definujeme zpárovaný klíč K' po složkách vztahem

$$K'_i = \overrightarrow{K_i} \oplus e_i$$

a pro otevřený text P bude obdobně zpárovaný otevřený text P' roven

$$P'_i = \overrightarrow{P_i} \oplus d_i.$$

Celý útok se pak bude řídit následujícím algoritmem:

1. Vygeneruj náhodný otevřený text P a zašifruj ho pomocí klíče K.
2. Spočítej P' a zašifruj ho pomocí K'.
3. Ověř, jestli $\overrightarrow{E_K(P)} = E_{K'}(P')$. Pokud ano máme rotační pár. Jinak opakuj stejný postup.

Z rotačního páru pak stejně jako v případě bez konstant můžeme získat hodnotu nějakých bitů klíče. Pro další bity klíče opakujeme stejný postup se zrotovaným klíčem.

3. Užití rotační kryptoanalýzy

Nyní se můžeme pustit do aplikace rotační kryptoanalýzy na konkrétní šifry. Ta je obvykle velmi přímočará.

3.1 Threefish

Threefish [8] je bloková šifra využívaná v hašovací funkci Skein, jednom z finalistů soutěže na nový standard SHA-3. Právě pro tuto šifru Dmitry Khovratovich a Ivica Nikolić metodu rotační kryptoanalýzy vytvořili. Popsané útoky pocházejí z jejich článku [1].

3.1.1 Popis šifry

Threefish je *modifikovatelná bloková šifra* (tweakable block cipher [9]) pracující s 64-bitovými slovy. Blok šifrovaného textu a šifrovací klíč jsou vždy stejně dlouhé a to buď 256 bitů, 512 bitů nebo 1024 bitů. První dvě varianty mají 72 rund, poslední pak 80.

Označme N_ω počet slov klíče a tedy i otevřeného textu ($N_\omega \in \{4, 8, 16\}$). Vstupem šifry pak je klíč $K = (k_0, \ldots, k_{N_\omega - 1})$, *modifikující vektor* (tweak) délky 128 bitů $T = (t_0, t_1)$ a otevřený text $P = (p_0, \ldots, p_{N_\omega - 1})$.

Zaměřme se nejdříve na tvorbu klíče. Označme si $t_2 = t_0 \oplus t_1$ a vyrobme ještě jedno speciální klíčové slovo

$$K_{N_\omega} = C_5 \oplus \bigoplus_{i=0}^{N_\omega - 1} k_i, \text{ kde } C_5 = \lfloor 2^{64}/3 \rfloor.$$

Jednotlivé složky s-tého rundovního klíče k_s dostaneme pomocí rovnic

$$\begin{aligned}
k_{s,i} &= k_{(s+i) \bmod (N_\omega+1)}, \qquad i = 0, \ldots, N_\omega - 4, \\
k_{s,N_\omega-3} &= k_{(s+N_\omega-3) \bmod (N_\omega+1)} + t_{s \bmod 3}, \\
k_{s,N_\omega-2} &= k_{(s+N_\omega-2) \bmod (N_\omega+1)} + t_{(s+1) \bmod 3}, \\
k_{s,N_\omega-1} &= k_{(s+N_\omega-1) \bmod (N_\omega+1)} + s.
\end{aligned}$$

Vlastní šifrování probíhá podle schématu na obrázku 3.1. Jednu rundu tvoří nejdříve funkce Mix sloužící k důkladné změně v malé oblasti (konfusi) a potom $Permute$ zajišťující rozmělnění bitů (difusi) po celé šifrované zprávě. Na začátku a po každých čtyřech rundách se přičítá rundovní klíč.

Funkce Mix, znázorněná na schématu 3.2, má na vstupu dvě slova x_0, x_1 a výstupu dvě slova y_0, y_1, pro která platí

$$\begin{aligned}
y_0 &= x_0 + x_1, \\
y_1 &= (x_1 \lll_{R_{(d \bmod 8)+1,i}}) \oplus (x_0 + x_1).
\end{aligned}$$

Hodnoty konstant $R_{r,i}$ lze nalézt ve specifikaci šifry [8]. Pro náš útok nejsou nijak podstatné.

Funkce $Permute$ je pouze permutace bitů. Její přesnou podobu lze opět nalézt ve specifikaci šifry.

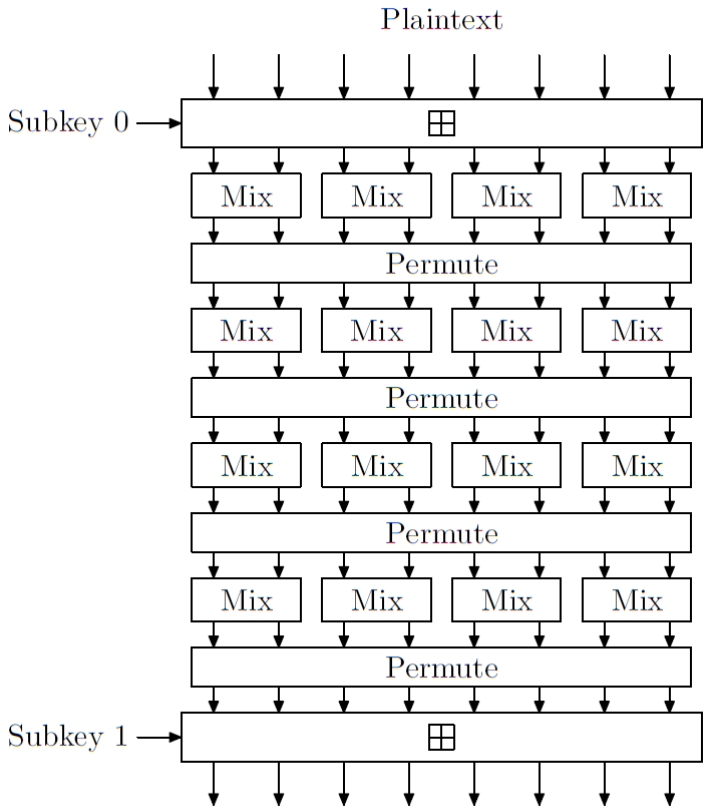


Schéma 3.1: První 4 rundy šifry Threefish-512, převzato z [8].

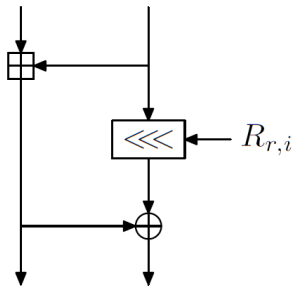


Schéma 3.2: Funkce Mix, převzato z [8].

3.1.2 Útoky na zjednodušenou verzi

Celou dobu budeme předpokládat, že $t_0 = t_1 = 0$, tedy i $t_2 = 0$. Vhodně zvolené hodnoty modifikujícího vektoru mohou útok metodou rotační kryptoanalýzy na tuto šifru téměř znemožnit.

V rámci šifry Threefish jsou dvě místa, kde se setkáme s konstantami. Jednak je to konstanta C_5 XORovaná na K_{N_ω}. A potom k poslednímu slovu rundovního podklíče $k_{s,N_\omega-1}$ je přičítána hodnota čítače s.

Zkusme nejdříve obě tyto konstanty vypustit. V rotačním páru budeme uvažovat rotaci o 1, abychom dostali co nejvyšší pravděpodobnost zachování rotačního

páru při sčítání. Konkrétně je tato pravděpodobnost rovna $2^{-1.415}$.

Funkce Mix obsahuje pouze jedno sčítání, proto runda Threefish-256 obsahuje pouze dvě sčítání (Threefish-256 obsahuje v každé rundě dvě funkce Mix). Dokážeme sestrojit útok až na 59 rund. Během nich dojde k $2 \cdot 59 = 118$ sčítání v rámci funkce Mix a k $4 \cdot 15 = 60$ sčítání při přičítání slov rundovního podklíče. Pravděpodobnost, že bude zachován rotační pár je tedy rovna $2^{-1.415 \cdot (118+60)} \geq 2^{-252}$. To je víc než pro náhodnou permutaci. Pokud bychom se spokojili s menším počtem rund, dostali bychom větší jistotu, že nebyl rotační pár zachován náhodou.

Každý rotační pár nám dává informaci o dvou bitech přenosu — v tomto případě konkrétně o přenosu z nultého na první bit a o přenosu z nejlevějšího bitu. Tu máme díky tomu, že víme, že první sčítání (otevřený text a první podklíč) zachovává rotační pár. Rotační pár se zachovává i při dalších sčítáních — jak ve funkci Mix, tak při přičítání rundovního podklíče. Před tímto sčítáním ale už otevřený text podstoupil tak složitou transformaci, že nejsme schopni rozumně takto získané informace využít.

Naše znalosti se tedy redukují na následující dvě pozorování:

1. Pokud byl poslední bit (lsb = least significant bit = nejpravější bit) i-tého slova otevřeného textu roven 1, tak poslední bit i-tého klíče ($k_{0,i} = k_i$) je roven 0. Jedničkový přenos by totiž pokazil rotační pár.

2. Pokud byl první bit (msb = most significant bit = nejlevější bit) i-tého slova otevřeného textu roven 1, tak první bit i-tého klíče ($k_{0,i} = k_i$) je roven 0. Jedničkový přenos by opět pokazil rotační pár.

Vzhledem k tomu, že otevřený text si sami volíme, můžeme ho volit tak vhodně, aby nám nalezení rotačního páru opravdu dalo informaci o dvou bitech každého ze slov klíče.

Po nalezení rotačního páru stačí slova klíče zrotovat o dva vpravo (to nám předpoklady našeho útoku umožňují) a můžeme stejným způsobem pokračovat v hledání dalších dvou bitů.

Dostáváme útok zjišťující informaci o bitech klíče pro Threefish-256 omezený na 59 rund se složitostí přibližně 2^{252}.

Analogicky můžeme vytvořit také útok pro 59 rund šifry Threefish-512 se složitostí 2^{504} a pro 59 rund Threefish-1024 se složitostí 2^{1008}. Zvětšení složitosti je úměrné prodloužení otevřeného textu a klíče.

Uvažujme nyní situaci, kdy se ke K_{N_ω} XORuje konstanta C_5. V takovém případě rotaci o 1 využít nemůžeme. Máme ale velké štěstí, protože při rotaci o dva platí $\overrightarrow{C_5} = C_5$. Budeme tedy rotovat o dva. To nám umožní útoky na šifry s 50 rundami. Jejich složitost bude pro Threefish-256 rovna $2^{1.67 \cdot (2 \cdot 50 + 4 \cdot 13)} < 2^{254}$, pro Threefish-512 bude 2^{508} a konečně pro Threefish-1024 bude rovna 2^{1016}.

Pro Threefish bez konstant existuje i třída slabých klíčů, při jejichž použití dostaneme útok na plný počet rund. Konkrétně to jsou klíče, jejichž všechna slova mají tři nejlevější bity nulové. To je podmínka silná, ale i tak ji vyhovuje mnoho klíčů.

Pro ně je pak pravděpodobnost zachování rotačního páru $2^{-0.28}$. A dostáváme tak složitost útoku na plný počet rund Threefish-256 rovnu 2^{224}. Pro všechny rundy Threefish-512 složitost vychází 2^{448} a pro Threefish-1024 pak 2^{950}.

3.1.3 Útok na skutečný kryptosystém

Nyní se budeme zabývat šifrou Threefish tak, jak byla navržena. Tedy včetně přičítání konstant. Zachováme si pouze omezení $t_0 = t_1 = 0$. Budeme postupovat tak, jak bylo popsáno v sekci 2.2.

Nejdříve hrubou silou dopočítáme pro jednotlivé verze šifry konstanty d_i a e_i. Jejich hodnoty lze nalézt v původním článku [1]. Kvůli přítomnosti konstant neumíme přesně vyjádřit pravděpodobnost zachování rotačního páru. Musíme si tedy vystačit s přibližným řešením metodou Monte-Carlo.

Takto získáme útok na 39 rund Threefish-256 se složitostí 2^{253}, 42 rund Threefish-512 se složitostí 2^{507} a na 43 rund Threefish-1024 se složitostí 2^{1015}.

3.1.4 Zhodnocení útoků

Obecně na metodě rotační kryptoanalýzy lze nalézt hned několik nedostatků. V první řadě je to velmi silný předpoklad nezávislosti zachování rotačního páru při jednotlivých sčítáních během průchodu šifrou. Ten může skutečné výsledky značně zkreslovat. To připouští i autoři útoku ve svém dalším článku [2].

Další nepříjemností je teoretická maximální schopnost útoku. Není naděje, že bychom pomocí něj uměli někdy prolomit všechny rundy šifry Threefish. Navíc už vhodná volba modifikačního vektoru, který jistě nemá být při praktickém užití šifry vždy nulový, téměř zcela znemožňuje jakýkoli útok rotační kryptoanalýzou.

K tomu se hodí podotknout, že samotná šifra Threefish prochází drobnými změnami. Její verze 1.3 [10] již místo konstanty C_5 využívá konstantu $C_{240} =$ `0x1BD11BDAA9FC1A22`. Pro ni již žádná vhodná rotace, která by se konstanty zbavila, neexistuje. Tedy nelze na ni použít ani výše popsané útoky.

3.2 Další šifry

3.2.1 Pro jaké šifry je útok vhodný

Při hledání dalších šifer, na které by bylo možné rotační kryptoanalýzu aplikovat narážíme na řadu omezení. Potřebujeme, aby se nepříliš upravený klíč (abychom dokázali vrátit jeho úpravy zpět) přičítal k nepříliš modifikovanému otevřenému textu (opět musíme být schopni vrátit úpravy zpět). To už z principu vylučuje použití rotační kryptoanalýzy na proudové šifry.

Není možné ani užití při příliš komplikované tvorbě rundovních klíčů. To od rotační kryptoanalýzy ochrání například šifru RC5 [11], která by se jinak jevila jako dobrý kandidát pro útok. Z pohledu rotační kryptoanalýzy je RC5 ukázkovým příkladem jednoduché šifry, která zcela bez problémů útokům odolá.

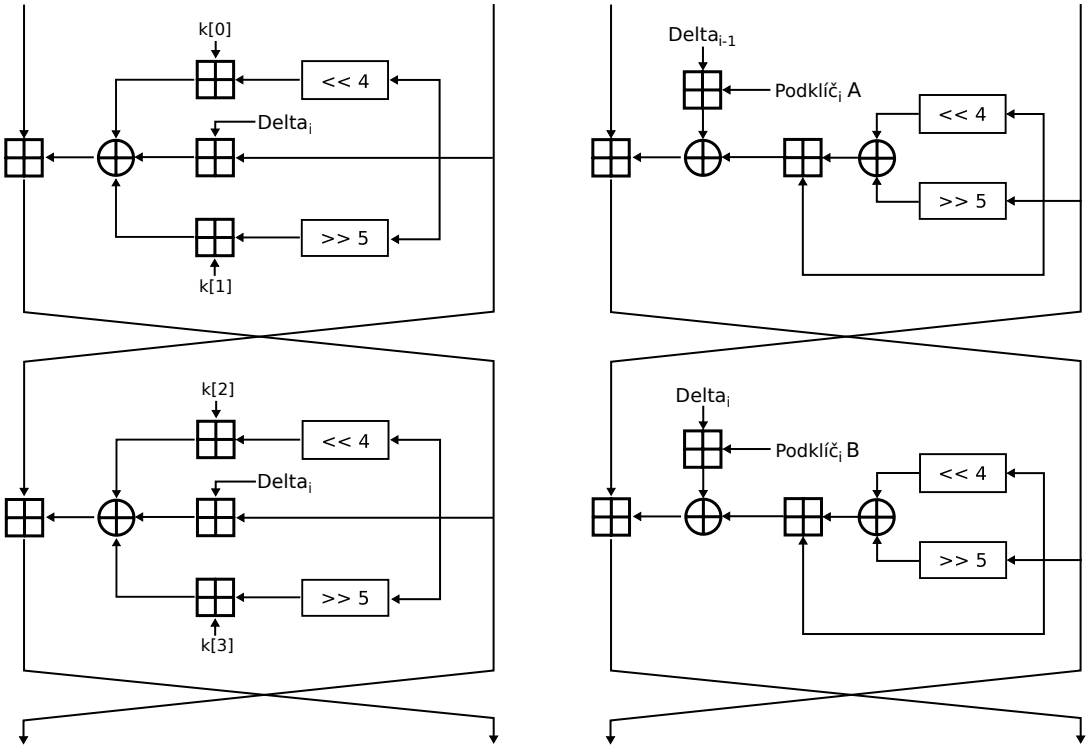
Dalším zakázaným prvkem, který se v šifře nesmí objevit, jsou S-boxy. Ty bývají vyjádřeny velmi složitou funkcí. Tu bychom sice teoreticky uměli přepsat, aby využívala pouze sčítání, XOR a rotaci. Ale počet sčítání by byl téměř jistě příliš vysoký.

Přitom S-boxy využívá většina současných blokových šifer. Nabídka šifer vhodných pro útok se tedy výrazně ztenčuje. Navíc často (podobně jako v případě Threefish) musíme šifru ochudit o nějakou konstantu, aby byl útok vůbec proveditelný. Tak tomu bude i v případě útoku na šifry TEA a XTEA.

Známy jsou nám nepříliš přesvědčivé útoky pomocí rotační kryptoanalýzy na šifru Shabal [12] a BMW [13]. Žádné další aplikace se nám najít nepodařilo. To odpovídá velkým omezením, které jsou na šifry vhodné pro rotační kryptoanalýzu kladeny. V následující části nabízíme útok na zjednodušené verze šifer TEA a XTEA.

3.2.2 TEA a XTEA

Blokovou šifru TEA [14] navrhli společně David Wheeler a Roger Needham. XTEA [15] je potom její modifikace od stejných autorů řešící několik slabých míst původní šifry. Obě šifry jsou založené Feistelově schematu. Během výpočtu využívají pouze ARX operace a operaci shift (s tím se budeme muset vypořádat) a mají velmi jednoduchou tvorbu rundovních klíčů. Jeví se tedy jako vhodní kandidáti pro rotační kryptoanalýzu.



(a) TEA (b) XTEA

Schéma 3.3: Dvě rundy šifer TEA a XTEA, překresleno s využitím obrázků z anglické Wikipedie.

Šifry pracují se slovy délky 32 bitů. Očekávají vstup dlouhý 64 bitů, který se interpretuje jako dvě 32- bitová slova, a 128-bitový klíč. Klíč se rozdělí na čtvrtiny označené $k[0], k[1], k[2], k[3]$ a ty se potom už rovnou přičítají k (upravenému) otevřenému textu. V případě šifry TEA se ve všech rundách přičítají klíče stejně, u XTEA je rozhodování, který ze čtyř podklíčů přičíst, trochu složitější.

Jak obě šifry pracují, je naznačeno na schématech 3.3. Implementace obou šifer je velmi jednoduchá, proto níže uvádíme pro obě kompletní implementaci v jazyce C. V nich lze nalézt všechny detaily, které by ze schemat nemusely být patrné. Doporučený počet rund je 64. Poznamenejme, že v jazyce C znak `^` značí XOR, `<< k` shift vlevo o k bitů a `>> k` shift vpravo o k bitů.

```
void tea_encipher(long* v, long* k)      /* TEA */
{
    unsigned long  y=v[0],z=v[1],sum=0,
                   delta=0x9E3779B9,n=32;
    while(n-->0)
    {
        sum += delta;
        y += (z~<< 4) + k[0] ^ z + sum ^ (z~>> 5) + k[1];
        z += (y << 4) + k[2] ^ y + sum ^ (y >> 5) + k[3];
    }
    v[0]=y; v[1]=z;
}

void xtea_encipher(long* v, long* k)     /* XTEA */
{
    unsigned long  y=v[0],z=v[1],sum=0,
                   delta=0x9E3779B9,n=32;
    while(n-->0)
    {
        y += (z << 4 ^ z~>> 5) + z ^ sum + k[sum&3];
        sum += delta;
        z += (y << 4 ^ y >> 5) + y ^ sum + k[sum>>11 & 3];
    }
    v[0]=y; v[1]=z;
}
```

Při rotační kryptoanalýze narážíme na problém s konstantou `delta`. Neexistuje rotace, která by ji převedla opět ni samou. Proto nám nezbude než vytvořit útok pouze na zjednodušenou verzi šifer, kde je `delta` zvolena pro nás přívětivěji. Dále budeme předpokládat, že `delta` je invariantní vůči rotaci o jedna vpravo. Snadno bychom ale mohli náš útok upravit třeba pro případ, kdyby konstanta byla rovna C_5 ze specifikace Threefish.

Další nepříjemností jsou operace shift. S těmi se dokážeme vypořádat, ale bude nás to stát několik sčítání v každé rundě navíc. Přitom pokud bychom místo shiftu využívali rotace, konfuze v rámci jedné rundy bude naopak vyšší. Lze předpokládat, že shift byl využit hlavně kvůli vyšší rychlosti. Zkusme tedy nejdřív vymyslet útok na upravené šifry, kde se místo shiftu používá rotace. Označme si je rotTEA a rotXTEA.

V jedné rundě rotTEA dochází ke třem sčítáním. Pravděpodobnost zachování rotačního páru při sčítání je $2^{-1.415}$. Pokud by se šifra chovala jako náhodná permutace, vznikl by rotační pár s pravděpodobností 2^{-64}. Platí $\lfloor 64/(1.415 \cdot 3) \rfloor = 15$. Tedy umíme rozeznat nenáhodné chování při nejvýše 15 rundách.

Pokud nám zůstane zachován rotační pár, známe dva bity přenosu při přičítání klíčů $k[0]$ a $k[1]$. Při vhodné volbě otevřeného textu, tedy umíme zjistit první a poslední bit z klíčů $k[0]$ a $k[1]$. Vhodná volba otevřeného textu je zde (v analogii s útokem na Threefish) taková, aby u druhého slova byl po rotaci vlevo o čtyři i vpravo o pět první i poslední bit roven jedné. To nám dává podmínku na čtyři bity otevřeného textu. Můžeme se ale také spokojit s předepsáním jen jednoho bitu otevřeného textu. Pak získáme informaci o jednom bitu jednoho z klíčů.

Zrotováním klíčů a opakováním tohoto postupu dostaneme postupně celé klíče $k[0]$, $k[1]$. S klíči $k[2]$ a $k[3]$ se pracuje až v druhé rundě. Do té doby prošlo pravé slovo otevřeného textu změnami, které nedokážeme odsimulovat. Proto s jejich zjišťováním počkáme až na chvíli, kdy budeme znát $k[0]$ a $k[1]$. V tuto chvíli bychom už průběh první rundy mohli dobře simulovat. Jednodušší ale je celý klíč k zrotovat tak, aby se $k[2]$ a $k[3]$ dostali na místo, kde původně byli $k[0]$ a $k[1]$. Pak můžeme postupovat stejně jako v případě prvních dvou klíčů.

Dostali jsme tedy útok na 15 rund rotTEA se složitostí přibližně 2^{64}.

V případě rotXTEA je situace ještě příznivější. V jedné rundě tu jsou pouze dvě sčítání. Jsme tedy schopni vytvořit útok na $\lfloor 64/(1.415 \cdot 2) \rfloor = 22$ rund se složitostí přibližně 2^{63}.

Získávání klíčů bude stejné jako v případě rotTEA. Jen budeme muset získávat klíče postupně po jednom. Pro některé konstanty se může stát, že se nevyužijí všechny podklíče. Speciálně při volbě konstanty invariantní vůči rotaci o jedna by se používal vždy jen jeden. U konstanty invariantní vůči rotaci o dva (to je třeba již zmíněná C_5 z Threefish) se použijí pouze dva ze čtyř.

Nyní se vraťme k původním šifrám. Abychom mohli rotační kryptoanalýzu využít, musíme si shift vyjádřit pomocí sčítání XORu a rotace.

Shift vlevo odpovídá vynásobení vhodnou mocninou dvojky. Konkrétně shift vlevo o k odpovídá násobení 2^k. Násobit umíme pomocí sčítání. Binárním algoritmem dokážeme násobit číslem 2^k pomocí k sčítání. Shift vpravo o k si můžeme rozepsat jako rotaci vpravo o k (tím se bity, které chceme nulovat, dostanou na nejlevější pozice), násobení 2^k (neboli shift vlevo) a následně opět rotaci vpravo o k (tím se vynulované bity dostanou kam patří).

Budeme postupovat obdobně jako v případě rotTEA a rotXTEA. Jen počet sčítání v rundě se zvýší.

Pro slovo, které dostaneme po operaci shift, si nemůžeme předepsat libovolný bit. Některé bity budou totiž vždy nutně nulové. Vždy tedy dokážeme dostat jedničku jen na první nebo poslední pozici (podle toho, zda se jedná o shift vlevo či vpravo). Takže při nalezení jednoho rotačního páru dokážeme zkonstruovat pouze jeden bit z klíčů $k[0]$ a $k[1]$.

V jedné rundě šifry TEA zapsané pomocí ARX dochází k 12 sčítání. Umíme tedy vytvořit útok na $\lfloor 64/(1.415 \cdot 12) \rfloor = 3$ rundy se složitostí přibližně 2^{51}. Podobně v jedné rundě XTEA dochází k 11 sčítání. Dokážeme zaútočit na 4 rundy se složitostí přibližně 2^{62}.

Vidíme, že počet rund, na které jsme maximálně schopní útočit je značně menší než doporučený počet. Navíc jsme šifru značně oslabili nevhodnou volbou konstanty `delta`. Ani v tomto případě tedy určitě nelze mluvit o úspěšném útoku.

4. Rotační rozpínavý útok

Zajímavou modifikací rotační kryptoanalýzy je *rotační rozpínavý útok* (rotational rebound attack) popsaný v článku [2], jehož autory jsou Dmitry Khovratovich, Ivica Nikolić a Christian Rechberger. Tato část se pokusí jejich postup kombinující rotační kryptoanalýzu a rozpínavý útok (rebound attack) přiblížit.

Naším cílem tentokrát nebude získat klíč, ale pouze ukázat, že se daná šifra nechová zcela jako ideální bloková šifra, tedy jako náhodná permutace. Klíč bude naopak patřit k tomu, co si můžeme zvolit (jedná se o chosen-key attack).

Opět budeme útok demonstrovat na šifře Threefish popsané v části 3.1. Tentokrát ale dokážeme zaútočit na 53 rund Threefish-256 a 57 rund Threefish-512. Navíc se vypořádáme i s nenulovým modifikujícím vektorem (tweakem).

4.1 Rozpínavý útok obecně

Rozpínavý útok [16, 17] je variantou diferenční kryptoanalýzy, jehož užití je známé především u šifer konstrukčně podobných AES. Blokovou šifru (případně kompresní funkci) W si při něm rozložíme na tři části, $W = W_{fw} \circ W_{in} \circ W_{bw}$. Nejdříve pracujeme s W_{in} a pak teprve s krajními.

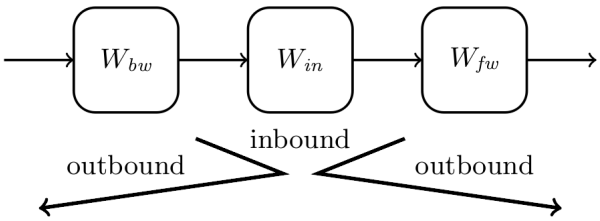


Schéma 4.1: Znázornění rozpínavého útoku, převzato z [17].

Celý útok je znázorněn na schématu 4.1. Útok má dvě základní části – vnitřní a vnější fázi. V nich se snažíme hledat dvojice vstupů, které odpovídají stanovené *diferenční cestě* (differential path) [18], tedy seznamu požadavků na rozdíly mezi vstupy.

Obecně by diferenční cesta měla být volena tak, aby po vnitřní fázi měla stále co nejvíce *stupňů volnosti*, tedy aby bylo z jedné nalezené vyhovující dvojice možné utvořit co nejvíce dalších. Navíc chceme, aby byla velká pravděpodobnost zachování diferenční cesty ve vnější fázi.

Ve *vnitřní fázi* (inbound phase) se pracuje s W_{in}. Jako W_{in} se používá taková část funkce, která obsahuje jedno přičítání klíče a nějaké jeho okolí, kde se s klíči nepracuje, z obou stran. Nejdříve najdeme mnoho textů, které vyhovují diferenční cestě v úsecích před a za přičítáním klíče. Ty následně kombinujeme tak, aby diferenční cesta byla zachována i přes úsek s klíčem. Klíč si můžeme libovolně určit. Tento postup bývá označován jako technika meet-in-the-middle. S využitím všech stupňů volnosti se připraví co nejvíce vyhovujících dvojic textu a klíče.

Při *vnější fázi* (outbound phase) se výpočtem na obě strany od místa vnitřní fáze ověřuje, jestli nalezené dvojice vyhovují celé diferenční cestě.

Výsledkem je potom seznam co nejvíce dvojic otevřených textů a klíčů. Pro ty platí, že pokud otevřený text šifrujeme pomocí příslušného klíče, postupné

modifikace otevřeného textu vyhovují předem stanovené diferenční cestě. Jakým způsobem tyto dvojice následně využijeme k dokončení útoku, už záleží na konkrétní aplikaci.

4.2 Rotační kolize

Naším cílem bude ukázat, že u Threefish umíme vytvořit mnoho dvojic otevřených textů a klíčů požadovaných vlastností s menší složitostí, než by se nám mohlo povést v případě ideální blokové šifry. Musíme proto odhadnout zdola složitost vytvoření stejného počtu dvojic u ideální blokové šifry.

Při rotačním rozpínavém útoku bude zachování diferenční cesty odpovídat zachování rotačního páru. Kvůli přítomnosti čítače (counter) při tvorbě klíčů v Threefish budeme potřebovat opět doplnit korekce pro zvýšení pravděpodobnosti zachování rotačního páru. Oproti sekci 2.2 se budou ale korekce týkat pouze zrotovaných klíčů.

Definice. Uvažujme rotaci o pevně zvolené r vlevo. Buď e konstanta, $E_K(\bullet)$ bloková šifra s klíčem K a P otevřený text. Pokud platí

$$\overleftarrow{E_K(P)} = E_{\overleftarrow{K}+e}(\overleftarrow{P}),$$

tak dvojici (K, P) nazveme *rotační kolizí* (rotational collision).

Pokud hodnota konstanty e není pevně daná, je složitost nalezení rotační kolize podle narozeninového paradoxu přibližně $2^{n/2}$, kde n je počet bitů P. My ale chceme najít více rotačních kolizí se stejnou hodnotou e. To přesně odpovídá hledání dvojic otevřených textů a klíčů zachovávajících rotační pár při rozpínavém útoku.

Definice. Buď opět $E_K(\bullet)$ bloková šifra. Množinu

$$\{e; (P_1, K_1), (P_2, K_2), \ldots, (P_s, K_s)\}$$

nazveme *rotační s-kolizní množinou* (rotational s-collision set), pokud platí

$$\begin{aligned}
\overleftarrow{E_{K_1}(P_1)} &= E_{\overleftarrow{K_1}+e}(\overleftarrow{P_1}),\\
\overleftarrow{E_{K_2}(P_2)} &= E_{\overleftarrow{K_2}+e}(\overleftarrow{P_2}),\\
&\vdots\\
\overleftarrow{E_{K_s}(P_s)} &= E_{\overleftarrow{K_s}+e}(\overleftarrow{P_s}).
\end{aligned}$$

Hodnota e je určena první rovnicí. Složitost nalezení každé další dvojice v množině je přibližně 2^n. Přesněji situaci popisuje následující lemma.

Útočník na ideální blokovou šifru nemůže využít žádnou slabinu v její konstrukci. Jedné co mu zbývá je tedy klást dotazy na zašifrování nebo rozšifrování zvolených zpráv při zvolených klíčích.

Lemma 4.1 (složitost nalezení rotační kolizní množiny, [2])**.** Na zkonstruování rotační s-kolizní množiny pro ideální blokovou šifru s n-bitovými bloky potřebuje útočník průměrně alespoň $\mathcal{O}\left(s \cdot 2^{\frac{s-2}{s+2}n}\right)$ dotazů.

Důkaz tohoto lemmatu je poměrně dlouhý. Je však analogický důkazu multicollison lemma, který je velmi pěkně a přehledně sepsán v článku [19].

4.3 Aplikace rotační varianty

Nyní známe složitost získání s-kolizní množiny pro ideální blokovou šifru. Ukážeme, že v případě Threefish je tato složitost nižší.

Uvažujme šifru Threefish-256 omezenou na rundy 3–55, případně Threefish-512 omezený na rundy 3–59. Označme si K šifrovací klíč a T modifikující vektor. Pro naše účely bude přirozené vnímat modifikující vektor jako součást klíče.

Od nyní budeme rotací vždy rozumět rotaci o dva vlevo. Rotaci o dva volíme podobně jako v útoku klasickou rotační kryptoanalýzou popsaným v části 3.1.2 proto, abychom docílili co nejlepší pravděpodobnosti zachování rotačního páru při sčítání a abychom se zároveň vyhnuli konstantě C_5.

Vnitřní fáze

Ve vnitřní fázi útoku využijeme všech stupňů volnosti, abychom vytvořili co nejvíce kandidátů na rotační kolizi pro pevně zvolené e. Těmto kandidátům budeme říkat *vyhovující dvojice*.

Vnitřní fáze probíhá v 8 po sobě jdoucích rundách, uprostřed nichž se přičítá rundovní klíč k_s. To je bod, kde se budou potkávat řetězce zkonstruované v rundách před a po přičítání klíče.

Nejdříve ve čtveřicích rund před a po přičítání klíče najdeme co nejvíce řetězců, které zachovávají během výpočtu rotační pár. Tedy chceme, abychom dostali stejný výsledek, pokud řetězec zrotujeme v libovolné fázi při průchodu rundami. Takovým řetězcům budeme říkat *vyhovující*. Je dobré si uvědomit, že čtyři rundy mezi přičítáním klíče jsou vždy zcela stejné. Tedy nemusíme rozlišovat řetězce vyhovující první a druhé čtveřici rund.

Následně se snažíme najít takový klíč k_s, abychom jeho přičtením k výsledku aplikace čtveřice rund na vyhovující řetězec dostali nějaký (jiný) vyhovující řetězec. Navíc ještě k_s musí mít některé bity rovny předepsaným hodnotám (viz dále).

Nalezená dvojice vyhovujícího řetězce a klíče (druhý vyhovující řetězec je jimi již jednoznačně určen) je hledanou vyhovující dvojicí.

Každá vyhovují dvojice v sobě nese značnou část informace o celém klíči K. Kvůli přítomnosti modifikujícího vektoru T při tvorbě rundovních klíčů v Threefish ho to ale neurčuje zcela. Stále můžeme k_{s-1} a k_{s+1} mírně modifikovat.

Zrychlující fáze

Mezi vnitřní a vnější fázi oproti obvyklému rozpínavému útoku přidáme ještě *zrychlující fázi* (acceleration phase). Můžeme ji vnímat též jako první část vnější fáze. Zrychlující fáze obklopuje vnitřní fázi. Probíhá tři rundy po směru šifrování a dvě rundy proti směru.

Při zrychlující fázi se snažíme využít zbývajících stupňů volnosti ve volbě klíčů k_{s-1}, k_{s+1} a z každé vyhovující dvojice nalezené ve vnitřní fázi vytvořit co nejvíce různých kandidátů na rotační kolizi vyhovujícím požadavkům na zachování rotačního páru i v rundách patřících do zrychlující fáze. Těmto kandidátům budeme říkat *nadějné dvojice*.

Nadějnou dvojici tvoří už kompletní klíč a řetězec, který při vstupu do začátku zrychlující fáze přes celou zrychlující, vnitřní i opět zrychlující fázi zachovává

při užití daného klíče rotační pár. Pro hledání nadějných dvojic můžeme využít metodu neutrálních bitů (neutral bits) [20].

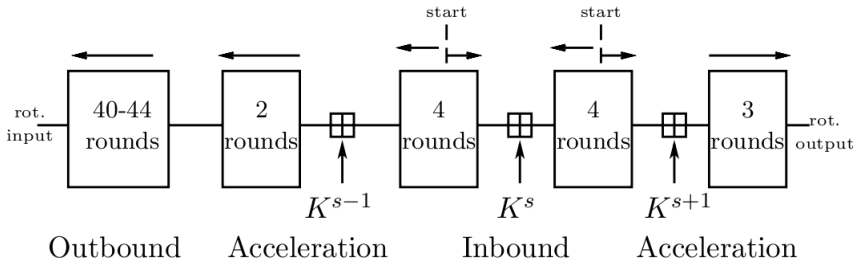


Schéma 4.2: Rotační rozpínavý útok pro Threefish-256 a -512, převzato z [2].

Vnější fáze

Vnitřní a zrychlující fáze jsou umístěny až na samotný závěr šifrování. Vnější fáze tedy probíhá celá proti obvyklému směru šifrování. Během ní ověřujeme, jestli nalezené nadějné dvojice opravdu určují rotační kolizi. Tedy jestli otevřený text daný nadějnou dvojicí (ten je už určen jednoznačně) a příslušný klíč tvoří rotační kolizi. Simulujeme pozpátku chod šifrou a ověřujeme, jestli je zachováván rotační pár.

U Threefish-256 je vnější fáze dlouhá 40 rund (jedná se o rundy 3–42 běžné šifry), u Threefish-512 se jedná o rundy 3–46, tedy celkem o 44 rund. Celý útok neprobíhá na rundy od první, protože vnitřní fáze potřebuje přičítání klíče ve správnou chvíli. Pokud bychom útočili od první rundy, dostali bychom méně rund, na které umíme útočit.

Podobně jako v případě rotační kryptoanalýzy Threefish 3.1.3 zavedeme pro zkoumání zachování rotačních párů předpočítané *korekce* pro klíče. Tentokrát je budeme ale ke klíčům přičítat a nikoli XORovat. Budeme tedy zkoumat dvojice $(K_i, \overleftarrow{K_i}+e_i)$, kde e_i je konstanta s malou Hammingovou váhou (tedy málo nenulovými bity), jejímž úkolem je co nejvíce vykompenzovat efekt čítače u posledního rundovního podklíče.

V článku [2] je poměrně podrobně popsána heuristika, jak takového konstanty hledat. Sami autoři ale na závěr píší, že nejlepších výsledků se jim podařilo dosáhnout při dopočítávání konstant hrubou silou. Vždy zkoušeli, kolik rotačních párů zůstane zachováno přes několik rund Threefish se zvolenou konstantou. A pak vybrali mezi konstantami tu nejvhodnější.

Aby dosáhli co největší pravděpodobnosti zachování rotačního páru, rozhodli se pevně zafixovat 6 bitů každého slova klíče. Vždy čtyři bity zleva a dva zprava. Přesné hodnoty fixovaných bitů lze nalézt v původním článku [2]. Pravděpodobnost zachování rotačního páru v průběhu vnější fáze pak v případě Threefish-256 je 2^{-244}, u Threefish-512 vychází 2^{-494}.

Zhodnocení

Složitost vnitřní a zrychlující fáze není vysoká kvůli mnoha stupňům volnosti. Složitost výpočtu v rundách obsažených v těchto fázích proto můžeme považovat

za zhruba srovnatelné s jedním průchodem blokovou šifrou. U Threefish-256 strávíme čas 2^{224} na vytvoření 2^{224} nadějných dvojic pro vnější fázi, a nichž v průměru jedna opravdu bude rotační kolizí. Tedy rotační kolizi umíme najít se složitostí průměrně 2^{224}.

Na nalezení rotační 2^7-kolizní množiny u Threefish-256 potřebujeme tedy $2^{224+7} = 2^{251}$ dotazů. V případě ideální blokové šifry by to ale bylo alespoň $2^7 \cdot 2^{\frac{128-2}{128+2} \cdot 512} \doteq 2^{255}$. Je tedy zřejmé, že Threefish-256 omezený na rundy 3 až 55 se nechová jako náhodná permutace.

Obdobně v případě Threefish-512 umíme najít rotační kolizi se složitostí přibližně 2^{495}. Na nalezení rotační 2^8-kolizní množiny tedy potřebujeme průměrně pouze $2^{495+8} = 2^{503}$ dotazů. Zatímco v náhodné permutace by to bylo alespoň 2^{512} dotazů. Takže ani Threefish-512 omezený na rundy 3 až 59 se nechová jako ideální bloková šifra

Oproti předchozím kapitolám nebylo naším cílem získat používaný klíč. Chtěli jsme pouze ukázat, že se šifra nechová zcela jako ideální bloková šifra, tedy jako náhodná permutace. To naznačuje, jakým směrem by se mohly ubírat nějaké případné další útoky. A to se nám skutečně pro redukované verze šifer podařilo.

Závěr

Za hlavní přínos této práce považuji uspořádání a doplnění poznatků ohledně ARX a ARX+C funkcí a ukázání, že pokud se při tvorbě nějakého kryptografického primitivu rozhodneme využívat pouze tyto funkce, nijak se tím neochudíme. Vzhledem k rychlosti hardwarové implementace sčítání, XORu i rotace se domnívám, že se jedná o vhodnou skupinu funkcí pro tvorbu mnohých nových primitivů.

Z uvedených tří funkcí bývá obvykle nejobtížnější výpočet rotace, která se typicky provádí pomocí dvou shiftů a XORu. Zatímco sčítání i XOR bývají implementovány přímo v procesoru. Z tohoto pohledu je tedy zajímavý i výsledek, že pomocí AX+C všechny funkce získat nemůžeme a tedy pokud vynecháme při návrhu rotaci, tak se určitým způsobem ochuzujeme.

Pokud ale budeme chtít navrhovat hardwarově co nejméně náročné primitivy, můžeme místo rotace využívat pouhý shift. Pomocí sčítání, XORu, shiftu vlevo i vpravo a přičítání konstanty už zase můžeme libovolnou funkci vytvořit.

Druhým, neméně významným, přínosem potom je prozkoumání metody rotační kryptoanalýzy. Ačkoli tato metoda působí na první pohled zajímavě, její aplikace jsou velmi vzdálené reálně využitelnému útoku. Navíc je proti útoku rotační kryptoanalýzou velmi jednoduchá a účinná obrana spočívající v přítomnosti dostatečně náhodné konstanty. A tu nalezneme v naprosté většině běžně používaných šifer.

Na druhou stranu představuje rotační kryptoanalýza zajímavou metodu, jak poukázat na nepravidelnosti v chování určitého primitivu. I když toto poukázání nevyústí v přímý útok, může se jednat o varování, kudy by nějaký případný pokročilejší útok mohl směřovat.

Pokud tedy tvůrci kryptografických primitivů budou při svém návrhu brát v potaz i odolnost vůči rotační kryptoanalýze, získáme tak kvalitnější a tedy bezpečnější primitivy.

Nesmíme zapomínat ani na skutečnost, že metoda rotační kryptoanalýzy je poměrně nová. Třeba se budoucnu dočkáme nějakého útoku založeného na zachovávání rotačních párů, který již bude i prakticky aplikovatelný.

Seznam použité literatury

[1] D. Khovratovich and I. Nikolić, "Rotational cryptanalysis of ARX," in *Proceedings of the 17th international conference on Fast software encryption*, vol. 6147 of *Lecture Notes in Computer Science*, pp. 333–346, Springer-Verlag, 2010.

[2] D. Khovratovich, I. Nikolić, and C. Rechberger, "Rotational rebound attacks on reduced Skein," in *Advances in Cryptology — ASIACRYPT 2010*, vol. 6477 of *Lecture Notes in Computer Science*, pp. 1–19, Springer-Verlag, 2010.

[3] A. A. Klyachko and E. V. Menshova, "The identities of additive binary arithmetics," *CoRR*, vol. abs/1102.5555, 2011. arXiv:1102.5555v3.

[4] D. Stinson, *Cryptography: Theory and Practice, Second Edition*. CRC/C&H, 2002. ISBN 978-1584882060.

[5] M. Daum, *Cryptanalysis of Hash functions of the MD4-family*. PhD thesis, Universität Bochum, Fakultät für Mathematik, 2005.

[6] E. Biham, "New types of cryptanalytic attacks using related keys," in *Advances in cryptology – EUROCRYPT '93*, vol. 765 of *Lecture Notes in Computer Science*, pp. 398–409, Springer-Verlag, 1994.

[7] E. Tews, R.-P. Weinmann, and A. Pyshkin, "Breaking 104 bit WEP in less than 60 seconds," in *Information Security Applications, 8th International Workshop*, vol. 4867 of *Lecture Notes in Computer Science*, pp. 188–202, Springer-Verlag, 2007.

[8] N. Ferguson, S. Lucks, B. Schneier, D. Whiting, M. Bellare, T. Kohno, J. Callas, and J. Walker, "The Skein hash function family." Available online at `http://www.skein-hash.info/sites/default/files/skein.pdf`, 2008.

[9] M. Liskov, R. L. Rivest, and D. Wagner, "Tweakable block ciphers," in *Advances in Cryptology - CRYPTO 2002*, vol. 2442 of *Lecture Notes in Computer Science*, pp. 31–46, Springer-Verlag, 2002.

[10] N. Ferguson, S. Lucks, B. Schneier, D. Whiting, M. Bellare, T. Kohno, J. Callas, and J. Walker, "The Skein hash function family, version 1.3." Available online at `http://www.skein-hash.info/sites/default/files/skein1.3.pdf`, 2010.

[11] R. L. Rivest, "The RC5 encryption algorithm," in *Fast Software Encryption*, vol. 1008 of *Lecture Notes in Computer Science*, pp. 86–96, Springer-Verlag, 1995.

[12] G. V. Assche, "A rotational distinguisher on Shabal's keyed permutation and its impact on the security proofs." Available online at `http://gva.noekeon.org/papers/ShabalRotation.pdf`, 2010.

[13] I. Nikolić, J. Pieprzyk, P. Sokołowski, and R. Steinfeld, "Rotational cryptanalysis of (modified) versions of BMW and SIMD." Available online at `https://cryptolux.org/mediawiki/uploads/0/07/Rotational_distinguishers_(Nikolic,_Pieprzyk,_Sokolowski,_Steinfeld).pdf`, 2010.

[14] D. J. Wheeler and R. M. Needham, "TEA, a tiny encryption algorithm," in *Fast Software Encryption*, vol. 1008 of *Lecture Notes in Computer Science*, pp. 363–366, Springer-Verlag, 1995.

[15] R. M. Needham and D. J. Wheeler, "TEA extensions," tech. rep., University of Cambridge, 1997. Available online at `http://www.cix.co.uk/~klockstone/xtea.pdf`.

[16] F. Mendel, C. Rechberger, M. Schläffer, and S. S. Thomsen, "The rebound attack: Cryptanalysis of reduced Whirlpool and Grøstl," in *Fast Software Encryption*, vol. 5665 of *Lecture Notes in Computer Science*, pp. 260–276, Springer-Verlag, 2009.

[17] M. Lamberger, F. Mendel, C. Rechberger, V. Rijmen, and M. Schläffer, "The rebound attack and subspace distinguishers: Application to Whirlpool." Cryptology ePrint Archive, Report 2010/198, 2010.

[18] J. Daemen and V. Rijmen, *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer-Verlag, 2002. ISBN 3-540-42580-2.

[19] A. Biryukov, D. Khovratovich, and I. Nikolić, "Distinguisher and related-key attack on the full AES-256," in *Advances in Cryptology - CRYPTO 2009*, vol. 5677 of *Lecture Notes in Computer Science*, pp. 231–249, Springer, 2009.

[20] E. Biham and R. Chen, "Near-collisions of SHA-0," in *Advances in Cryptology - CRYPTO 2004*, vol. 3152 of *Lecture Notes in Computer Science*, pp. 290–305, Springer-Verlag, 2004.

Seznam použitých zkratek

Za každou zkratkou je v závorce uvedeno číslo strany, kde je podrobně vysvětlena.

ANF (4)
: algebraická normální forma

ARX funkce (3)
: funkce, které lze zapsat s využitím pouze modulárního sčítání, XORu a rotace

ARX+C funkce (3)
: funkce, které lze zapsat s využitím pouze modulárního sčítání, XORu, rotace a předem zvolených konstant

AR+C funkce (3)
: funkce, které lze zapsat s využitím pouze modulárního sčítání, rotace a předem zvolených konstant

AX funkce (6)
: funkce, které lze zapsat s využitím pouze modulárního sčítání a XORu

AX+C funkce (3)
: funkce, které lze zapsat s využitím pouze modulárního sčítání, XORu a předem zvolených konstant

TEA (21)
: bloková šifra, Tiny Encryption Algorithm

XTEA (21)
: bloková šifra, Extended TEA