bezpečnostní kódy, díl 6.

V klidu a bezpečí (6)

Po krátké pauze se opět vracíme k seriálu o bezpečnostních kódech. Při jejich aplikaci se nám občas stane, že žádný ze známých kódů není pro daný účel dost dobrý. Pro takový případ je vhodné znát alespoň několik základních technik, jejichž pomocí můžeme vybraný kód v jeho "problémových partiích" upravit konkrétnímu zařízení přímo na míru.

I přesto, že většina úprav, se kterými se dnes seznámíme, je ve své podstatě poměrně jednoduchá, jejich přínos pro praktické používání ECC je značný. Některé zdroje tyto techniky dokonce označují jako vytváření nových kódů ze starých. To je možná zas až příliš optimistický termín, neboť "nový" kód, vzniklý těmito úpravami, přejímá většinu svých vlastností od svého předka. Hovořit o tvorbě zcela nového druhu kódu proto není na místě. Realističtějším pohledem je představa "doladění" nejvhodnějšího z kandidátů tak, aby co nejlépe vyhověl konkrétním požadavkům.

V následujícím výkladu se postupně seznámíme s několika často používanými operacemi úprav ECC. Uvedeme si je přitom zhruba v tom pořadí, v jakém se v praxi používají nejčastěji. Pro lepší vazbu na dostupnou literaturu budeme za českým označením dané úpravy uvádět i její anglický název (dle [ROMA92]). Jazyková odlišnost mezi jednotlivými názvy je totiž mnohem menší než odlišnost významová (na což předeme upozorňuji), takže zde může snadno dojít k omylům z příčiny špatné interpretace názvu.

Kvůli jednotnému značení se dále dohodněme, že pro odlišení kódu před úpravou a po ní budeme používat symbol čárky v horním indexu (tedy například: vstupem operace je kód φ a výstupem kód φ', apod.). Dále, pokud nebude řečeno jinak, budeme pod pojmem "kód" rozumět "binární kód".

Rozšíření kódu (Extending a Code)

Obecně se jedná o přidání jedné nebo více souřadnic do vektorů kódových slov. V praxi se nejčastěji používá rozšíření q-árního kódu o paritní znak, kdy ke každému n-znakovému kódovému slovu přidáme ještě jednu souřadnici tak, aby výsledný součet přes všechny znaky ve slově byl nulový. Dále budeme pod pojmem rozšíření rozumět právě tuto operaci. V případě binárního kódu se jedná o přidání sudé parity.

Formální zápis pro novou množinu kódových slov C_k' je tento: $C_k' = \{ c_1c_2...c_nc_{n+1}: c_1c_2...c_n \in C_k, \Sigma_{k=1}^{n+1}c_k = 0 \}$. Označíme-li si parametry kódu před operací rozšíření, respektive po ní jako (n, k, d_{min}) (značení (n,k) budeme občas ještě doplňovat třetím parametrem, a to minimální kódovou vzdáleností), respektive (n', k', d'_{min}), potom platí, že $n' = n+1$, $k' = k$, $d'_{min} = d_{min}$ nebo $d_{min}+1$ – *definice D6.1*.

Hlavní účel této operace je možné spatřovat ve zvětšení minimální kódové vzdálenosti (cena, kterou za to zaplatíme, je prodloužení délky slova o jednu souřadnici – pro binární kódy o jeden bit). V praxi se tato operace používá zejména v souvislosti s tvrzení T2.1, neboť její pomocí můžeme minimální kódovou vzdálenost upravit na tvar $d'_{min} = 2t+2$ a umožnit tak detekci t+1 chyb při současné opravě t chyb.

Pro lepší představu o tom, jak tato operace mění minimální kódovou vzdálenost, si uvedeme následující pomocné tvrzení: Předpokládejme binární kód a operaci rozšíření o sudou paritu. Potom platí, že $d'_{min} = d_{min}$ iff $d_{min} = 2t+2$ a $d'_{min} = d_{min}+1$ iff $d_{min} = 2t+1$ – *tvrzení T6.1*. První věcí, která z tohoto tvrzení plyne, je, že minimální kódová vzdálenost kódu po jeho rozšíření je vždy sudá. Dáme-li toto zjištění do souvislosti s T2.1, pak vidíme, že rozšířený kód je vždy schopen simultánně opravovat t a detekovat t+1 chyb. Podle T2.4 zase dostáváme, že rozšířený kód nemůže být nikdy perfektní.

Druhá věc, která stojí za povšimnutí, je, že pro kódy, jejichž minimální kódová vzdálenost je sudá, nepřináší tato operace nic pozitivního – pouze prodlouží délku slova. Z toho plyne, že tuto operaci má smysl aplikovat pouze jednou, a to navíc na takové kódy, u kterých platí $d_{min}(\varphi) = 2t+1$. Konkrétní aplikaci na Hammingův binární kód (7,4) si ukážeme dále.

Zúžení kódu (Puncturing a Code)

Tuto operaci můžeme považovat za inverzní vůči operaci rozšíření kódu. Obecná definice říká, že se jedná o úpravu založenou na vynechání jedné nebo více souřadnic z vektorů kódových slov. V případě, že q-ární kód měl před úpravou minimální kódovou vzdálenost $d_{min}(\varphi) \geq 2$, potom vynecháním jedné souřadnice vznikne odvozený kód s parametry: $n' = n-1$, $k' = k$, $d'_{min} = d_{min}$ nebo $d_{min} -1$ – *definice D6.2*.

Zajímavou souvislost mezi operacemi rozšíření a zúžení uvádí následující tvrzení: Binární kód typu $(n, k, d_{min} = 2t+1)$ existuje právě tehdy, když existuje binární kód s parametry $(n+1, k, d_{min} = 2t+2)$ – *tvrzení T6.2*. Důkaz tohoto tvrzení, který uvádí [ROMA92], je založen právě na použití operací rozšíření a zúžení.

Důsledek uvedeného tvrzení je pro praxi poměrně užitečný, neboť nám říká, že binární kód s $d_{min}(\varphi) = 2t+1$ můžeme vždy (popsanými operacemi) upravit na kód $d_{min}(\varphi') = 2t+2$ a obráceně. Důvod pro rozšiřování kódu jsme si už uvedli. Jako příklad pro použití operace zúžení nám mohou sloužit například Golayovy kódy, kterým jsme se věnovali minule. Zde jsme využili operaci zúžení k tomu, abychom získali perfektní kód (kód s $d_{min}(\varphi) = 2t+2$ totiž perfektní být nemůže – viz. T2.4).

Zvětšení kódu (Augmenting a Code)

Zatímco předchozí dvě úpravy se týkaly prodlužování či zkracování délek kódových slov, následující dvě operace ovlivňují velikost množiny kódových slov při zachování jejich délky.

Obecně pod pojmem zvětšení kódu rozumíme rozšíření množiny kódových slov C_k o několik dalších prvků. Stejně jako jsme se u předchozích operací víceméně omezili jen na sudou paritu, i zde se budeme zabývat pouze rozšířením binárních kódů tak, aby jejich C'_k obsahovala komplementy všech kódových slov. Pod pojmem komplement slova c přitom rozumíme jeho binární negaci a značíme ji nejčastěji jako neg(c) nebo c^c. Takto popsanou operaci rozšíření kódu φ značíme jako $\varphi' = (\varphi \cup \varphi^c)$ – *definice D6.3*.

Abychom si význam této operace lépe ujasnili, projdeme si nyní postup odvození $d_{min}(\varphi')$. K tomu budeme nejprve potřebovat následující pomocné tvrzení, které nám umožní rozšířit výpočet vzdálenosti dvou kódových slov: Mějme dvě binární slova slova x,y délky n. Potom platí, že $d(x,y^c) = n - d(x,y)$ – *tvrzení T6.3*. Důkaz tohoto tvrzení plyne z následující úvahy: vzdálenost $d(x,y^c)$ udává počet pozic, ve kterých se slova x a y^c liší. Vzhledem k použité operaci binární negace je to zároveň počet pozic, na kterých se slova x a y **ne**liší. Odtud už přímo dostáváme uvedený vztah.

Pomocí právě uvedeného tvrzení dokážeme následující: Nechť φ je kód typu (n, k). Potom $d_{min}(\varphi \cup \varphi^c) = \min\{d_{min}(\varphi), n-d_{max}(\varphi)\}$, kde $d_{max}(\varphi)$ odpovídá maximální kódové vzdálenosti kódu φ – *tvrzení T6.4*. Důkaz, který si zde načrtneme, vychází z následujícího vztahu: $d_{min}(\varphi \cup \varphi^c) = \min\{d_{min}(\varphi), d_{min}(\varphi^c), \min_{c\in Ck,\, d \in neg(Ck)}\{ d(c,d) \} \}$. Tento vztah odráží logický předpoklad, že minimální kódová vzdálenost bude dána minimem vzdáleností přes všechny dvojice slov kódu φ, φ^c a kódů φ a φ^c "navzájem". Výraz uvedený v T6.4 pak získáme úpravou tohoto vztahu pomocí tvrzení T6.3 (za předpokladu $d_{min}(\varphi) = d_{min}(\varphi^c)$).

Posledním naším úkolem bude pomocí T6.4 určit, jak bude popisovaná operace působit na lineární binární kód – tedy na ten typ kódu, se kterým se budeme setkávat nejčastěji. Důvodem, proč není vhodné použít rovnou T6.4, může být například to, že pro lineární kódy umíme výpočet minimální (analogicky i maximální) kódové vzdálenosti převést na jednodušší operaci hledání minima (analogicky maxima) váhy přes všechna nenulová kódová slova (viz. T3.4, rozšíření pro výpočet $d_{max}(\varphi)$ je analogické k důkazu bodu (3)).

S využitím T3.4 potom můžeme formulovat následující tvrzení: Nechť φ je binární lineární kód typu (n,k), který neobsahuje jednotkový vektor **1** = (1,1,...,1). Potom pro kód $\varphi' = (\varphi \cup \varphi^c)$ platí: $n' = n$, $k' = k+1$, $d_{min}(\varphi') = \min\{ d_{min}(\varphi), n - w_{max} \}$, kde w_{max} značí maximum váhy přes všechna kódová slova kódu φ – *tvrzení T6.5*.

Zde se sluší poznamenat, proč jsme do formulace podmínek pro T6.5 zahrnuli požadavek na $\mathbf{1} \notin C_k$. Je to proto, že pokud by lineární binární kód obsahoval jednotkový vektor, potom by platilo, že $C_k^c = C_k$ neboli $\varphi = \varphi^c$. Jinými slovy: daný kód by už obsahoval všechny doplňky svých kódových slov, takže jeho zvětšování popsaným způsobem by nemělo smysl. Důkaz právě nastíněného tvrzení je možné poměrně snadno zkonstruovat, když si uvědomíme, že pro binární slova platí: $c^c = (1,1,...,1)+c$. Jelikož součet dvou kódových slov lineárního kódu musí být kódové slovo, dostaneme, že pokud je jednotkový vektor v kódu obsažen, potom tento kód pro každé kódové slovo c obsahuje též kódové slovo c^c.

Z tvrzení T6.5 vidíme, že popisovaná operace zvětšení kódu se v praxi hodí zejména pro zvýšení

informační kapacity daného kódu o jeden bit. Zmenšení minimální kódové vzdálenosti, které dle T6.5 může nastat, je cena, kterou za tento bit "navíc" musíme zaplatit.

Zmenšení kódu (Expunging / Expurgating a Code)

Obecně se jedná o inverzní operaci ke zvětšení kódu, spočívající v odstranění některých kódových slov z množiny C_k daného kódu. Pro naše demonstrační účely budeme operaci zmenšení kódu definovat pro binární lineární kód typu (n,k), který obsahuje alespoň jedno slovo liché váhy, jako proces odstranění všech kódových slov liché váhy. Měl-li kód před operací parametry (n, k, d_{min}), bude mít po zmenšení parametry (n', k', d'_{min}), kde $n' = n$, $k' = k-1$, $d'_{min} \geq d_{min}$ – *definice D6.4*.

Právě uvedená operace se opírá o zajímavou vlastnost lineárních binárních kódů. Pokud takový kód obsahuje alespoň jedno slovo liché váhy, potom můžeme dokázat, že přesně polovina kódových slov má lichou váhu. Odstraněním všech slov liché váhy se nám tak velikost množiny kódových slov zmenší na polovinu. Protože nám po této operaci zůstanou v kódu pouze slova sudé váhy, zároveň podle T3.4 dostáváme, že d'_{min} musí být sudá. Pro případ, kdy $d_{min} = 2t+1$, tak přechází neostrá nerovnost v D6.4 v ostrou a platí: $d'_{min} > d_{min}$.

Použití této operace nám může přinést zvětšení minimální kódové vzdálenosti na úkor zmenšení informační kapacity upraveného kódu. Někdy se nám může zmenšení kódu hodit pro čistě teoretické účely, kdy jeho pomocí ukážeme, že nějaký kód φ vznikl "pouhým" zvětšením kódu φ', což nám pomůže rozptýlit naše obavy, že jsme přišli na něco převratného.

Zkrácení kódu (Shortening a Code)

Pod tímto pojmem rozumíme operaci, kterou z dané množiny C_k vybereme její podmnožinu (C_k'), ve které mají všechna slova na určené pozici stejný znak (označme ho s). Danou souřadnici (označme ji i) pak z těchto slov vypustíme, neboť už není nositelkou žádné informace. Výsledný kód označujeme jako výřez pro $x_i = s$ – *definice D6.5*.

O tom, jak se konkrétně chová výřez kódu pro $x_i = 0$, nás informuje toto tvrzení: Máme-li binární lineární kód typu (n, k, d_{min}), potom výsledkem výřezu $x_i = 0$ je binární lineární kód typu (n-1, k-1, d_{min}) – *tvrzení T6.6*.

Srovnáme-li operace zkrácení a zúžení kódu, vidíme, že obě dvě jsou v podstatě (z pohledu délky kódu) vhodné pro zmenšení délky kódových slov. Vzájemně se však liší tím, jakou cenu za to musíme zaplatit. V případě zúžení kódu se nám většinou zmenší minimální kódová vzdálenost a tím se zhorší zabezpečovací vlastnosti kódu (zato můžeme obdržet perfektní kód). Zkrácením se nám sice tato vzdálenost nezmění, ale zase nám poklesne informační kapacita kódu (i to může někdy cílem). Jakou úpravu nakonec zvolíme, proto záleží na podmínkách určených konkrétní aplikací.

Příklady

Přímo učebnicové příklady aplikace popsaných metod můžeme v literatuře nalézt v souvislosti s binárními Hammingovými kódy, zejména pak s kódem typu (7, 4). Pro ilustraci si uvedeme obrázek (originál viz. [ADAM89]), který ukazuje, jak jednotlivé operace mění vlastnosti tohoto kódu. Pro přehlednost jsme zde rozšířili zápis typu kódu o udání minimální kódové vzdálenosti.

Vidíme, že operací rozšíření obdržíme kód typu (8, 4, 4), který oproti původnímu kódu nabízí detekci dvou chyb při současné opravě jedné chyby. Tento kód se v literatuře vžil doslova jako vzorový příklad práce s Hammingovými kódy. Zmíníme se proto podrobněji o tom, jak se tato úprava kódu (7, 4) provádí. Vyjdeme přitom opět z kontrolní matice H, kterou upravíme na matici H' podle obrázku. Popíšeme-li tuto úpravu slovně, pak platí, že H' vytvoříme tak, že každý řádek matice H doplníme vpravo nulou a poté přidáme jeden řádek samých jedniček. V případě potřeby pak z této matice ještě dle T3.6 odvodíme generující matici G.

Snadno ověříme, že takto získaná matice H' je kontrolní maticí kódu (8, 4). Její tvar ostatně přesně odráží ono rozšíření o paritní bit, který je v tomto případě v kódovém slově přenášen jako poslední (bráno zleva). Doplněním nul na konce řádků v matici H' jsme (zjednodušeně řečeno) zajistili, že rovnice zde popsané rovnice tento bit "ignorují" a provádějí pouze kontrolu v rámci kódu (7, 4). Poslední řádek zase kontroluje jenom paritu přijatého slova a výsledek této kontroly je promítnut na posledním místě syndromu (zleva po transpozici, označme jej jako s_4).

Oprava chyb pak může probíhat podle následujícího scénáře (předpokládejme nenulový

syndrom): nejprve zkontrolujeme bit s_4. V případě, že je nulový, ohlásíme chybu, neboť víme, že přijaté slovo je zatíženo dvojnásobnou chybou (jinak by muselo platit $s_4 = 1$). V opačném případě provedeme opravu přijatého slova (pomocí $s_1s_2s_3$) dle standardního postupu pro Hammingovy kódy.

Poznamenejme, že tento postup jsme si uvedli záměrně proto, abychom lépe ilustrovali účinek provedeného rozšíření. V praxi se můžeme setkat s modifikací této metody, při které se matice H‘ upraví elementárními úpravami do tvaru, ve kterém mají všechny sloupce lichou paritu. Pro nenulový syndrom přijatého slova pak platí, že je-li lichý, pak došlo k chybě jednonásobné (tj. opravitelné), a je-li sudý, pak k chybě dvojnásobné (tj. neopravitelné). Tato vlastnost plyne z toho, že každý syndrom dvojnásobné chyby je tvořen součtem nějakých dvou syndromů chyby jednonásobné.

Další možnou, i když ne tak často uváděnou operací je zmenšení Hammingova kódu na typ (7, 3, 4). Zabezpečovací schopnosti tohoto kódu jsou stejné jako u (8, 4, 4), oba kódy se však liší délkou slova a počtem informačních bitů. Generující matici pro tento kód můžeme získat například z generující matice kódu (7, 4) – viz 4. díl; v ní první řádek přičteme ke druhému a třetímu a pak jej vynecháme. Takto jsme zaručili, že matice G obsahuje pouze vektory o sudé paritě, a tudíž žádné kódové slovo nebude mít lichou paritu. Vhodnou permutací sloupců potom matici upravíme na tvar uvedený na obrázku. Poznamenejme, že tento kód je duální ke kódu (7, 4) – matice G‘ je až na permutaci sloupců shodná s maticí H.

Závěr

Dnes jsme si ukázali několik základních technik, jejichž pomocí můžeme daný kód lépe přizpůsobit potřebám konkrétní aplikace. V dostupné literatuře je dále možné najít ještě pokročilejší metody, jako je třeba přímý součin dvou kódů, jehož pomocí se dá odvodit například dvourozměrný kód parity. Kvůli přehlednosti jsme zde rozbor těchto metod vynechali. V případě potřeby některé z nich se k nim ještě v průběhu tohoto seriálu vrátíme.

Příští díl bude věnován Reedovým-Mullerovým kódům.

Tomáš Rosa,