Bezpečnostní kódy, díl 2.

V předchozím, úvodním díle našeho miniseriálu jsme se zabývali základními vlastnostmi bezpečnostních kódů a motivacemi pro jejich tvorbu vůbec. Dnes na tento základ navážeme výkladem o dalších zajímavých vlastnostech, které se bezpečnostních kódů obecně týkají.

V klidu a bezpečí (2)

Ještě než se pustíme do výkladu nových informací, vrátíme se malinko zpět k tvrzením o detekci a opravě chyb. Snahou minulého dílu bylo, aby byl co nejméně náročný na pochopení, a proto jsem zde záměrně tato tvrzení příliš nespojoval dohromady. Pro další výklad je však třeba uvedené informace rozšířit.

Detekce a oprava chyb zároveň

Představme si, že daný kód slouží k opravě maximálně t chyb. Dále předpokládejme, že příslušný dekodér pracuje tak, že pokud je přijaté slovo x chybné, automaticky jej opraví na takové kódové slovo c, které má od x nejmenší vzdálenost. Pokud je přijaté slovo x kódové, potom samozřejmě prochází dekodérem bez úpravy.

Podle tvrzení *T1.2* víme, že popsaný kód má minimální kódovou vzdálenost $d_{min}(\varphi) = 2t + 1$. Za předpokladu, že by byl použit pouze pro detekci chyb, dostáváme podle *T1.1* schopnost detekce 2t chyb. Nyní se však musíme zeptat: Platí zde *T1.1* i v případě, že je použit výše zmíněný automatický dekodér, který samočinně provádí opravu přijatého slova? Odpověď zní: Ano, ale ne zcela přesně. Neboli platí, ale není nám k užitku.

Abych to nějak vysvětlil: o tom, že jsme v této situaci u libovolného přijatého slova stále schopni detekovat 2t chyb, není pochyb. Potíž je zde v tom, že automatický dekodér nám tyto chyby bez našeho zásahu rovnou opravuje, přičemž chyby způsobující odchylku větší než t(d(c, x) > t) opravuje chybně, neboť přijaté slovo nahradí nesprávným kódovým slovem. Podle *T1.1* tedy chyba zjištěna byla – tvrzení je v pořádku, avšak zároveň byla nesprávně opravena, což je de facto to samé, jako kdybychom si jí nevšimli.

Platnost tvrzení se nezměnila. Co se však změnilo, je naše chápání pojmu detekce chyby, neboť říkáme, že dekodér detekuje chybu právě tehdy, když přijal poškozené slovo, které je buď schopen správně opravit, anebo které není schopen opravit vůbec. Nesprávnou opravu přijatého slova nepovažujeme za úspěšnou detekci chyby.

Podíváme-li se na geometrické znázornění sfér kódových slov (viz minulý díl), zjistíme, že slova, u kterých jsme schopni detekovat chybu v právě uvedeném významu, leží buď přímo v původní sféře

vyslaného slova, anebo v prostoru, který není pokryt žádnou jinou ze sfér. Ve sférách sousedních kódových slov potom leží ta chybná slova, která dekodér nesprávně opraví – ta nás ale teď nezajímají.

Na základě této geometrické úvahy nyní můžeme zformulovat nové tvrzení, které budeme používat vždy, když se budeme ptát, kolik chyb je daný kód schopen detekovat, když předpokládáme, že je zároveň použit i k opravě chyb. Platí, že pokud je $d_{min}(\varphi) = 2t + 1$, potom je kód schopen opravit nejvýše t chyb a zároveň maximálně t chyb detekovat. V případě, že $d_{min}(\varphi) = 2t + 2$, potom kód opravuje všechny t-násobné chyby a zároveň je schopen t + 1 chyb detekovat (*tvrzení T2.1*).

Perfektní kódy

Jak už bylo řečeno, přidává ECC do původní zprávy jistou redundanci, která je však podstatou ECC, a tudíž je nevyhnutelná. Je přitom samozřejmé, že při návrhu daného kódu se snažíme o to, aby výsledná nadbytečnost byla vzhledem k potřebným vlastnostem kódu co nejmenší. Formálně potom tento proces nazýváme hledáním takzvaného perfektního kódu.

Slovně můžeme perfektní kód charakterizovat jako kód, který má vzhledem k daným požadavkům (počet informačních bitů a minimální kódová vzdálenost) minimální nadbytečnost.

K tomu, abychom si pojem perfektního kódu popsali matematicky, neboť jen tak jej můžeme nakonec použít, potřebujeme nejprve definovat některé pomocné prvky. Začneme uvedením definice sférického okolí kódového slova, které jsme si minule předvedli v jeho geometrické podobě. Sférickým okolím o poloměru r kódového slova c nazveme množinu
$Sq(c, r) = \{y \in C: d(c, y) \leq r\}$ (*definice D2.1*). V tomto zápise netřeba hledat žádnou vědu, neboť jen shrnuje nám již dobře známý slovní popis, který říká, že do Sq(c, r) patří všechna slova z množiny C, která mají od c kódovou vzdálenost menší nebo rovnu r. Poznamenejme ještě, že index q, který jsme použili (ještě mnohokrát použijeme), znamená, že daný kód je konstruován nad q-ární abecedou neboli nad abecedou, která má q znaků. Nejčastější pro nás bude samozřejmě abeceda binární, kde q = 2.

Vlastní definice množiny Sq(c, r) nám sama o sobě nestačí, neboť je to jen popis už známého prvku. Pro nás je důležité umět spočítat, kolik prvků vlastně množina Sq(c, r) obsahuje. Bez důkazu (není těžký – můžete si jej zkusit sami) si zde proto uvedeme tvrzení, které nám tento výpočet umožní. Toto tvrzení říká, že pro velikost množiny Sq(c, r), kterou označíme jako $Vq(n, r) = |Sq(c, r)|$, platí, že $Vq(n, r) = \sum_{k=0}^{r} \binom{n}{k}(q-1)^k$, kde n je délka kódového slova (*tvrzení T2.2*). Výraz $\binom{n}{k}$ přitom znamená kombinační číslo, které můžeme vypočítat jako $\binom{n}{k} = n!/((n-k)!k!)$. Jako návod pro ty, kdo si chtějí zkusit udělat důkaz tohoto tvrzení, připomínám, že číslo $\binom{n}{k}$ nám říká, kolika způsoby můžeme z množiny o n prvcích vybrat k prvků tak, aby se nám v této k-tici žádné elementy neopakovaly. Poznamenejme dále ještě, že velikost množiny Sq(c, r) nezávisí na svém středu – tedy na kódovém slově c.

Poslední pomocné tvrzení, které budeme potřebovat, nám v podstatě opět jen shrnuje to, co už víme. Říká totiž, že pokud máme kód φ s $d_{min}(\varphi) = 2t + 1$, potom libovolné n-znakové slovo $y \in C$ patří nejvýše do jedné sféry Sq(c, t) (*tvrzení T2.3*). Jinými slovy nám toto tvrzení říká, že při $d_{min}(\varphi) = 2t + 1$ mají sféry o poloměru t kolem všech kódových slov prázdný průnik. To ale pro nás není nic nového, neboť jsme zde jen formálně zapsali to, co jsme si už minule názorně vyložili na obrázku, když jsme se bavili o schopnosti kódu opravovat chyby.

Zavedení pojmu perfektní kód je nyní již snadné. Za perfektní kód považujeme takový kód, jehož sférická okolí všech kódových slov mají navzájem prázdný průnik a dohromady pokrývají celou

množinu C. Velikost C je dána délkou kódového slova, kterou značíme jako n. Je-li kód konstruován nad q-ární abecedou, potom /C/ = q^n – to není nic objevného. Nyní se nám ale musí podařit celé C pokrýt pomocí všech sférických okolí. Na tomto místě musíme přibrat podmínku, že $d_{min}(\varphi) = 2t + 1$, neboť potom víme (viz *T2.3*), že žádné dvě Sq(c, t) se nepřekrývají a mají maximální poloměr. Proto nám při pokrývání množiny C stačí pouze kontrolovat počet obsažených slov – víme, že se žádné z nich nebude opakovat. Dále je vhodné si uvědomit, že sférických okolí je na C tolik, kolik je kódových slov, tedy q^k (uvažujeme kód typu (n, k)), a že velikost každého z nich je dána funkcí Vq(n, t). Odtud již můžeme napsat následující nerovnici: $q^n \geq Vq(n, t)*q^k$. Tato nerovnice přitom přechází v rovnici právě tehdy, když je daný kód perfektní (*tvrzení T2.4*).

Ohledně právě uvedeného tvrzení se sluší učinit ještě několik poznámek. Za prvé je třeba připomenout, že uvedená nerovnice pouze popisuje vztah mezi délkou kódového slova a počtem informačních znaků při daném $d_{min}(\varphi)$. Pokud přejde tato nerovnice v rovnici, potom víme, že daný kód má minimální možnou nadbytečnost – je perfektní. Nikdo nám ale už nezaručuje, že takový kód (n, k) skutečně existuje! Tvrzení *T2.4* nám pouze umožňuje určit, jaké by takový kód musel mít parametry, kdyby existoval. Později si ukážeme, že v praxi se používají perfektní kódy pouze dvou druhů (Hammingovy a Golayovy) a že tyto kódy existují pouze pro určité hodnoty (n, k).

Druhá poznámka se týká podmínky, že $d_{min}(\varphi) = 2t + 1$, kterou jsme do našeho tvrzení přibrali. Nutnost jejího splnění nám zde ukazuje souvislost mezi vlastností kódu být perfektním a mezi schopností detekovat chyby při současně prováděné opravě (viz. *T2.1*). Srovnáme-li obě uvedená tvrzení, potom vidíme, že pokud po kódu chceme, aby v okamžiku, kdy opravuje t chyb, byl schopen ještě t + 1 chyb detektovat, potom takový kód nemůže být perfektní. Obráceně platí, že je-li kód perfektní, potom není při současné opravě t chyb schopen detekce chyby v t + 1 znacích – tato chyba bude nesprávně opravena. Z uvedeného můžeme vyvodit, že perfektní kódy nemusejí být vždy pro danou aplikaci nutně “perfektní” v přesném slova smyslu.

Systematický kód

V minulém díle jsme si zavedli pro kód označení (n, k), kde n udává celkovou délku kódového slova a k říká, kolik je v tomto slově informačních znaků. Toto označení je však třeba u zcela obecného kódu chápat také zcela obecně, neboť nám v podstatě říká jen tolik, že daný q-ární kód obsahuje q^k kódových slov. Nikdo nám už ale nezaručuje, že u přijatého kódového slova můžeme lokalizovat přesně k pozic, jejichž vybráním získáme přenášenou (zakódovanou) informaci. Tuto vlastnost máme zaručenu pouze u kódů, které se označují jako systematické.

Pro ilustraci si uveďme příklad. Předpokládejme, že máme kód (3, 2), jehož množina kódových slov C_k = {000, 100, 010, 001}. Celkem snadno nahlédneme, že $d_{min}(\varphi) = 1$, takže od kódu nemůžeme prakticky vůbec nic zajímavého očekávat. Nicméně můžeme si na něm demonstrovat, jak vypadá nesystematický kód. Výběrem libovolné dvojice (kód přenáší dva bity informace) souřadnic v kódovém slově se nám totiž nepodaří jednoduše přímo získat hodnotu přenášené informace – vždy budou dvě kódová slova, která budou mít vybrané souřadnice stejné.

Jako příklad systematického kódu si uvedeme kód sudé parity, jehož množina kódových slov pro typ (3, 2) vypadá takto: C_k = {000, 110, 101, 011}. Vidíme, že kód má nejen $d_{min}(\varphi) = 2$, ale také že přenášenou informaci můžeme velmi snadno získat restrikcí přijatého slova na jeho první dvě souřadnice.

Nyní můžeme naše pozorování shrnout do definice systematického kódu: q-ární kód φ typu (n, k) nazveme systematickým, pokud můžeme najít k pozic (i_1, i_2, ..., i_k) takových, že vybráním těchto pozic ze všech kódových slov obdržíme množinu všech (q^k) možných slov délky k. Pozice (i_1, i_2, ..., i_k) přitom označujeme jako takzvané informační znaky, přičemž zbylých n-k pozic nazýváme jako kontrolní znaky (*definice D2.1*).

Poznamenejme, že zde uvedená obecná definice systematického kódu neklade požadavky na to, aby vybrané pozice (i_1, i_2, ..., i_k) informačních znaků tvořily souvislý blok od začátku kódových slov. V některé literatuře [ADAM89] se naproti tomu tento požadavek na systematický kód klade, takový kód budeme označovat jako souvisle systematický (*definice D2.2*). Jistě snadno nahlédneme, že pokud je kód systematický podle *D2.1*, potom je možné jej prostou permutací souřadnic (v technické realizaci se jedná o takzvané překřížení drátů) převést na souvisle systematický podle *D2.2*.

Typy a rodiny ECC

Jak jistě mnozí z vás tuší, existuje poměrně velké množství jednotlivých druhů kódů, které v tomto seriálu (naštěstí) ani nestačíme všechny probrat. Abychom si ale udělali alespoň hrubý obrázek o tom, jaké možnosti nám může současný rozvoj ECC poskytnout, pokusím se pro vás tuto oblast v rychlosti shrnout v následujícím přehledu.

Dále se budeme bavit o typech a rodinách kódů, takže se jistě sluší vysvětlit, co si máme pod těmito pojmy představit. Již jsme si řekli, že každý kód má nějakou minimální kódovou vzdálenost a že podle její velikosti můžeme určit jeho schopnost detekovat či opravovat chyby v přijatých slovech. O náročnosti vlastního procesu kódování a dekódování jsme se však zatím ještě nebavili. Má-li být tento proces efektivní, což znamená, že nebudeme používat nejjednodušší metody založené na popisu kódování a dekódování pomocí tabulky (někdy se jí ale také nevyhneme), musí jej být možné nějak šikovně matematicky popsat. Proto se snažíme, aby množina všech kódových slov vytvářela nad danou abecedou nějakou vhodnou (nejčastěji algebraickou) strukturu, se kterou se dá už pomocí dobře zvládnutých nástrojů současné matematiky pracovat. Typ daného kódu nám přitom říká, nad jakou konkrétní strukturou je tento definován.

Nad stejnou strukturou může být konkrétní kód vytvořen několika různými způsoby. O tom, jaký způsob je u daného kódu použit, bude potom vypovídat jeho příslušnost k určité rodině.

Lineární kódy

Asi nejznámějším a patrně též nejdůležitějším typem kódů jsou takzvané lineární kódy. Jak už jejich název napovídá, chápou se zde kódová slova jako vektory (každý znak odpovídá jedné dimenzi), se kterými je možné pracovat pomocí pravidel lineární algebry.

Množina všech slov lineárního kódu tvoří lineární prostor V(n, q) (prosím neplést se symbolem pro velikost sféry – viz *T2.4*), přičemž množina všech kódových slov potom tvoří jistý podprostor $L \subseteq V(n, q)$. Toto uspořádání nám dává možnost snadno ověřovat, je-li přijaté slovo kódové, či nikoliv, podle toho, je-li prvkem podprostoru L, či nikoliv. Operace kódování je také poměrně snadná, neboť se jedná o zobrazení kódovaného slova (které pro tento účel též chápeme jako vektor) do podprostoru L. Obě operace je přitom možné snadno popsat pomocí maticových operací.

Příjemnou vlastností lineárních kódů je, že každý z nich je možné převést na systematický.

Cyklické kódy

Ačkoliv jsou pro řadu aplikací lineární kódy postačující, v některých případech může být jejich struktura chudá na jisté operace (například násobení dvou kódových slov). V takových případech přicházejí ke slovu cyklické kódy, ve kterých se kódová slova chápou jako polynomy zbytkových tříd v nějakém konečném tělese. Více si k této problematice řekneme, až budeme probírat konkrétní zástupce cyklických kódů. Prozatím postačí, když si řekneme, že cyklické kódy po svém úspěšném zvládnutí vynikají zejména možností flexibilního přizpůsobení kódu přímo na míru dané aplikaci (BCH kódy) a dále možností opravy takzvaných shluků chyb (Reedovy-Solomonovy kódy).

Pokud si představíme rozdíl v chápání kódových slov v lineárních a cyklických kódech, zjistíme, že se zde provádí celkem podobný "trik" v přepisu posloupnosti přenášených znaků do koeficientů příslušné algebraické struktury. Například slovo (1011) bude v lineárním kódu chápáno jako vektor v = (1,0,1,1), zatímco v cyklickém kódu to bude polynom $f(x) = 1*x^3 + 0*x^2 + 1*x^1 + 1*x^0 = x^3 + x^1 + 1$. Zde se nabízí logicky otázka, zda existuje i nějaká hlubší souvislost mezi těmito typy kódů, a ukazuje se, že ano. Jisté rodiny lineárních kódů lze totiž zároveň považovat za kódy cyklické. Konkrétně je lineární kód možné považovat za cyklický, pokud pro něj platí, že je-li vektor $v = (c_1, c_2, ..., c_k)$ kódovým slovem, potom je též vektor $w = (c_k, c_1, c_2, ..., c_{k-1})$, tj. cyklický posuv, kódovým slovem (*definice D2.3*). Tato vlastnost se nám bude hodit později při výkladu o cyklických kódech.

Nelineární kódy

Lineární a cyklické kódy dohromady tvoří nejčastěji používané typy kódů. Nicméně občas se objeví i některé typy založené například na různých zajímavých kombinatorických strukturách, které se (pro svou vlastnost nelinearity) souhrnně označují jako kódy nelineární. Časem si v našem seriálu ukážeme nějakého jejich zástupce, avšak zatím nám postačí vědět, že tyto kódy existují.

Rodina Hammingových kódů

Takzvané Hammingovy kódy, které byly objeveny nezávisle Marcelem Golayem v 1949 a o rok později Richardem Hammingem, jsou dnes bezesporu "latinou" v oblasti ECC vůbec. Jsou lineární, perfektní a všechny binární Hammingovy kódy jsou též cyklické. Můžeme najít též některé obecné q-ární Hammingovy kódy, které jsou také cyklické.

Mezi nejčastěji používané patří binární Hammingovy kódy, které jsou typu (n, k), kde $n = 2^r - 1$, $k = n - r$. Jejich minimální kódová vzdálenost je 3 a v případě potřeby je možné ji rozšířením kódu o sudou paritu zvětšit na 4. Kódy se vzdáleností 3 se v literatuře často označují jako SEC (Single Error Correcting) a se vzdáleností 4 jako SEC-DED (Single Error Correcting – Double Error Detecting). Zkratka SEC-DED odráží fakt, že rozšířený Hammingův kód (který však už není perfektní – viz *T2.4*) je schopen při současné opravě jedné chyby (SEC) detekovat i chyby dvojnásobné (viz *T2.1*).

Golayovy kódy

Tyto kódy byly objeveny Marcelem Golayem roku 1948. Jedná se celkem o čtyři druhy lineárních kódů, z nichž jsou dva binární a dva ternární. Mezi binární patří G_{24} s parametry (24, 12), $d_{min}(\varphi) = 8$ a dále G_{23} s parametry (23, 12), $d_{min}(\varphi) = 7$, který je perfektní a vznikne zúžením G_{24}. Vidíme, že na rozdíl od Hammingových kódů jsou G_{24} a G_{23} schopny opravovat až trojnásobné chyby. Kód G_{23} je navíc cyklický.

Ternární kódy tvoří G_{12} s parametry (12, 6), $d_{min}(\varphi) = 6$ a dále jeho perfektní zúžení G_{11} s parametry (11, 6), $d_{min}(\varphi) = 5$. Vidíme, že tyto kódy jsou schopny opravovat dvojnásobné chyby. Kód G_{11} je též cyklický.

Příkladem použití těchto kódů může být třeba kosmická sonda Voyager, která s úspěchem používala G_{24} pro přenos barevných fotografií Jupiteru a Saturnu.

A ty další

Existuje ještě mnoho zajímavých druhů kódů, na které v průběhu našeho seriálu jistě přijde řeč. Na rozdíl od předchozích dvou rodin ale již bohužel není možné jejich vlastnosti obdobně stručným způsobem shrnout. Sem patří zejména Reedovy-Mullerovy kódy, BCH kódy a Reedovy-Solomonovy kódy. Poslední dvě rodiny jsou v poslední době stále oblíbenějšími zástupci cyklických kódů, a bude jim proto později věnována odpovídající pozornost.

Na závěr

Dnes jsme si rozšířili přehled obecných vlastností ECC a uvedli jsme si základní členění v současnosti nejpoužívanějších metod. Příští díl bude věnován kompletně výkladu o lineárních kódech, kde se zaměříme zejména na Hammingovy kódy.

Tomáš Rosa (tomas.rosa@decros.cz)

Literatura:

[ADAM89] Adámek, J.: Kódování, SNTL Praha, 1989.

[ROMA92] Roman, S.: Coding and Information Theory, Springer-Verlag, 1992.