Cifrul lui Hill: analiza timpilor de criptare în eventualitatea unui atac de tip forță-brută

Studiu de caz: 'copia simetrică' în operațiile de refacere a informației corupte aferente compresiei/decompresiei datelor

Hill's Cipher: Analysis of the Cryptographic Computational Times in the Eventuality of a Brute-Force Attack

Case study: the 'symmetrical copy' used in restoration scenarios for data corruption in compression/decompression contexts

Vlad-Alexandru GROSU
Faculty of ETTI, University POLITEHNICA of Bucharest, Romania
crisvlad74@yahoo.com

Abstract

For the present article, I use a symmetrical cryptographic method based on symbols transposition: Hill's cipher. This consists in a symmetrical approach based on modulo m arithmetic. The theory says that one has to choose a prime number as the length of the modulo-m space used. Nevertheless, based on my conclusions here, this choice is not mandatory. As a proof of concept for the supportive case study, I chose a computational medium based on a system-on-chip solution offered by Texas Instruments, namely Beagleboard. The main purpose is to emphasize the original concepts that I introduced along my PhD thesis, 'self-recognition' and 'symmetrical copy'. The data restoration operation that accompanies the set of compression/ decompression operations that I proposed requires the generation of a file called 'symmetrical copy'. The overall details of the above mentioned new concepts can be found in my PhD thesis - "Real-time compression/decompression algorithms applied to embedded systems". That very practical context is one of the possible situations within which Hill's cipher still suits.

Index terms: algorithms, symmetrical cryptography, arithmetic, self-recognition, symmetrical copy, data restoration, system-on-chip, Beagleboard

1. Cifrul lui Hill: detalii matematice

Algoritmul folosit este o implementare personală a cifrului lui Hill (*Hill Cipher*). Acesta face parte din clasa bazată pe scheme de transpoziție a simbolurilor. În acest fel, copia de siguranță câștigă în robustețe. Am pornit în implementare de la documentele existente pe această temă, "*Hill Ciphers and Modular Linear Algebra*" de Murray Eisenberg și "*Using Encryption Technique*" (autor fiind Anthony-Claret Onwutalobi de la Universitatea din Wollongong Australia).

Nu am folosit criptările puternice (gen DES, AES, RSA) dată fiind scara acestui proiect: aplicații domestice din clasa aplicațiilor de tip *uz personal* și nu guvernamental sau de altă natură. Odată acest pas fiind făcut, proiectul câștigă o caracteristică importantă, care se poate îmbunătăți. Dacă proiectul va evolua și noi contexte de lucru vor impune acest lucru, voi adopta imediat scheme de criptare mai performante.

Cifrul lui Hill folosește două chei:

- cheia de criptare, și
- cheia de decriptare.

Ambele chei sunt de fapt *matrice pătratice*. Dintre ele, cheia de decriptare se bazează pe calcul matriceal. Fiind o aplicație pe mulțimea numerelor întregi - codurile caracterelor sunt valori din $\mathbb{N}$, toate calculele referitoare la criptare se vor efectua în clasa de resturi modulo M. Numărul M trebuie ales astfel încât să fie în legătură cu alfabetul de intrare folosit.

În prezentul articol intrarea este reprezentată de *fișier*. Contextul utilizării unui fișier este dat de operația de refacere a datelor corupte în cazul unui fișier arhivă.

Un fișier se prezintă întotdeauna într-una din cele două forme distincte:

- fișier de tip *text*;
- fișier de tip *binar*.

Din fericire, pentru ambele situații simbolurile disponibile (mulțimea caracterelor utile) se bazează pe tabela ASCII extinsă. În mod uzual, majoritatea sunt simboluri alfanumerice. Cum tabela ASCII extinsă oferă 256 de combinații utile, alegerea mea pentru numărul M este 256. La limită se poate alege lungimea alfabetului de intrare și un număr *prim*, anume M=257. Acest lucru poate îmbunătăți suplimentar securitatea împotriva unor eventuale atacuri criptografice.

Abordarea referitoare la criptarea datelor nu necesită timp suplimentar de calcul. Având în vedere că M=256 am încorporat (*hard-coding*) în codul sursă ambele chei necesare cifrului: atât matricea directă cât și pe cea inversă. Prin aceasta, în acest studiu de caz am evitat calculul inversei unei matrice. Efectul este benefic având în vedere limitările de procesare ale SoC Beagleboard. Singurele operații la care fac apel în codul sursă sunt cele fundamentale: adunare, scădere, înmulțire și, la nevoie, împărțire - necesare obținerii cheilor de criptare și decriptare.

Mai trebuie spus că, din punctul de vedere al puterii și siguranței criptării, prin faptul că am înglobat cheile de criptare/decriptare direct în codul sursă, am ascuns de fapt cheile de decriptare. În acest fel nu se cunosc nici dimensiunea cheii de criptare și nici dimensiunea alfabetului de intrare, adică numărul M (detalii în secțiunea *Puterea de criptare a cifrului Hill)*.

Pentru decriptarea copiei simetrice, cheia cere calculul modulo M a valorii inverse a determinantului matricei de intrare (*modular multiplicative inverse*):

$$(1/\det(A)) \pmod M$$

unde A este matricea (cheia) de criptare, iar det(A) este determinantul matricei A.

Aici este important de observat că, dată fiind o clasă de resturi modulo M, nu orice număr natural are un astfel de invers - denumit și *simetric* - care să aparțină acelei clase de resturi (modulo M).

În secțiunea următoare sunt prezentate aspectele matematice esențiale legate de cifrul Hill.

1.1. Cifrul Hill și aritmetica modulo M

În cele ce urmează voi defini noțiunile matematice cu care operează cifrul lui Hill [1].

Se desprinde și concluzia principală, anume că numărul M trebuie să fie *număr prim* pentru simplificarea calculelor necesare.

Definiție 1:

Mulțimea $Z_m = \{0, 1, ..., m-1\}$ împreună cu operațiile de adunare și înmulțire se numește sistem de numerație al întregilor modulo m.

Propoziție 1:

Fie m un număr întreg, cu proprietatea m>1. Atunci în sistemul de numerație Z_m au loc proprietățile:

1. Pentru $\forall$ a, b, c $\in Z_m$,
   $(a + b) + c = a + (b + c)$
2. Pentru $\forall$ a, b $\in Z_m$,
   $a + b = b + a$
3. Pentru fiecare a $\in Z_m$,
   $a + 0 = a = 0 + a$
4. Pentru fiecare a $\in Z_m$, există un unic x $\in Z_m$, denumit opusul lui a, astfel încât:
   $a+x = 0 = x+a$
5. Pentru $\forall$ a, b, c $\in Z_m$,
   $(a \cdot b) \cdot c = a \cdot (b \cdot c)$
6. Pentru $\forall$ a, b $\in Z_m$,
   $ab = ba$
7. Pentru fiecare a $\in Z_m$,
   $1 \cdot a = a = a \cdot 1$
8. Pentru fiecare a $\in Z_m$, cu a=0, există un unic y $\in Z_m$, denumit simetricul lui a, astfel încât:
   $a \cdot y = 1 = y \cdot a$
9. Pentru $\forall$ a, b, c $\in Z_m$
   $a \cdot (b + c) = a \cdot b + a \cdot c$
10. În Z_m avem:
    $1 = 0$

În general, proprietatea 8 enunțată mai sus nu este întotdeauna adevărată.

De exemplu, pentru sistemul de numerație Z_4 avem următorul tabel al operației de *înmulțire*:

Tabel 1. Înmulțirea în sistemul de numerație modulo 4, Z4

.	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	0	2
3	0	3	2	1

Se poate observa că valoarea 2 nu are un element simetric mod 4, din moment ce nu există nici o valoare în Z_4 care înmulțită cu 2 să dea rest 1 mod 4. În schimb numărul 3 este inversabil și-l are ca element simetric pe 3 (chiar el însuși).

Teorema generală care justifică acest comportament este următoarea:

Un element a din Z_m are o valoare simetrică (multiplicative inverse) modulo m dacă și numai dacă a este relativ prim cu m, adică 1 este divizorul comun al lui a și m (în sensul obișnuit al împărțirii). Din această teoremă derivă și *Propoziția 2* de mai jos.

Spre deosebire de Z_4, în Z_5 fiecare element are un *simetric*, așa cum ne arată tabelul înmulțirii în Z_5 de mai jos.

Tabel 2. Înmulțirea în sistemul de numerație modulo 5, Z5

.	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

Diferența între Z_4 si Z_5 este aceea că 5 este număr prim, în timp ce 4 nu.

Definiție 2:

Un sistem de numerație închis față de operațiile de adunare și înmulțire se numește *câmp* atunci când toate proprietățile 1-10 de mai sus sunt adevărate.

Observând deosebirea între Z_4 și Z_5 și pornind de la teorema anterioară rezultă propoziția următoare:

Propoziția 2:

Fie m un număr întreg, cu m>1. Atunci Z_m este câmp dacă și numai dacă m este prim.

Privind exemplele anterioare referitoare la Z_4 si Z_5, avem că Z_5 este câmp în timp ce Z_4 nu este. Pentru fiecare număr prim m, câmpul Z_m este finit.

Din punctul meu de vedere - al implementării - aceste rezultate îmi arată cum trebuie ales m pentru simplificarea calculelor: să fie număr prim. Însă, dacă valorile matricei (cheii) de criptare sunt alese cu atenție, atunci este posibil calculul matricei inverse (determinantul matricei directe va avea un simetric în clasa de resturi modulo m) și pentru mulțimi Z_m pentru care m nu este prim.

În studiul de caz avut în vedere, compresia datelor lucrează cu caractere ale tabelei ASCII extinse (codificate pe 1 octet). Aceasta înseamnă o gamă de 256 de combinații posibile la intrare. În consecință, lungimea spațiului de lucru poate fi aleasă drept: m=256, adică mulțimea Z_{256}.

Numărul de coloane al matricei este 4. Această alegere se bazează pe lungimea numărului magic, care intervine în operația de *recunoaștere proprie* (self-recognition). Acest 'număr magic' reprezintă aici o combinație de 4 caractere, utilizată drept *delimitator* al informației existente în fișierul rezultat în urma compresiei. Prin această tehnică permit identificarea unui fișier aparținând noului tip de arhivă construit prin verificarea simultană a două variabile: semnătura fișierului (informație internă fișierului) și amprenta digitală a fișierului pe baza sumei MD5 (informație externă fișierului). Tot în urma utilizării acestui delimitator, tipul de fișier arhivă creat se bazează pe un format *hibrid*. Prin hibrid înțeleg faptul că arhiva conține atât informație binară, rezultată în urma compresiei, cât și informație în clar, ne-alterată, identificabilă prin aceste numere magice. Aceasta este o abordare originală pe care am propus-o în teza de doctorat, "*Algoritmi de compresie/ decompresie în timp real aplicați sistemelor încapsulate*" susținută în cadrul Facultății de Electronică, Telecomunicații și Tehnologia Informației din Universitatea Politehnica din București.

Revenind la algoritmul Hill, dacă se dorește o schimbare a modului în care are loc criptarea, orice programator poate prelua codul propus și poate opera modificările pe care le crede de cuviință, în conformitate cu licența LGPL pe care acest proiect o cuprinde.

Proiectul care susține acest studiu de caz este unul open-source, pentru care am conceput un text propriu de copyright, conform principiilor *General Public License* [2]. Modularizarea codului permite acest lucru.

2. Puterea de criptare a cifrului Hill

Pentru scopul proiectului meu cifrul Hill asigură o protecție suficientă a datelor. Acesta rezistă cu succes la atacurile de tip *forță brută* (*brute-force attack*) în situația în care cheia de decriptare este ascunsă. Se arată că acesta este foarte sigur în situația unor atacuri exclusiv asupra textului criptat (aici: asupra copiei simetrice). Aceasta se datorează faptului că spațiul cheilor (matricelor pătratice) se poate extinde foarte mult prin alegerea valorilor matricei, de exemplu din mulțimea întregilor pe 64 de biți. Mai mult, spațiul cheilor se mai poate extinde suplimentar și prin utilizarea unor matrice de dimensiune mare [3].

Bineînțeles, dacă se cunoaște cheia de criptare *și* textul criptat, atunci operația de spargere a oricărui cifru este imediată.

Mai mult, un atac poate reuși dacă sunt cunoscute, în același timp, următoarele [1]:

- dimensiunea n a matricei de criptare;
- dimensiunea M a alfabetului de intrare;
- textul în clar;

- textul criptat.

În situația proiectului de față, valorile n și M nu sunt publice ci înglobate în codul sursă. Un potențial atacator are la dispoziție doar textul în clar și textul cifrat, ambele putând exista simultan pe dispozitivul de stocare folosit.

Pentru ca un fișier criptat cu cifrul Hill să fie spart, un atacator trebuie să intercepteze n^2 perechi de tipul:

{text în clar <-> text cifrat}

unde n reprezintă dimensiunea cheii de criptare (dimensiunea matricei pătratice de intrare).

Odată cunoscute aceste date, se poate construi un sistem algebric liniar care, în mod obișnuit, se poate rezolva relativ ușor. Dacă cumva sistemul este nedeterminat atunci este necesară adăugarea de perechi suplimentare de tipul celor n^2 deja preluate [1]. Folosind notația asimptotică utilizată în teoria algoritmilor se demonstrează că timpul de calcul al soluției unui astfel de sistem - folosind metode ale algebrei liniare - se situează în limita lui $O(n^2)$.

În secțiunea următoare se poate observa cum evoluează timpul necesar rezolvării unui sistem liniar algebric, dacă se urmărește un atac criptografic.

3. Timpul de calcul necesar rezolvării sistemelor algebrice liniare

Folosind o implementare proprie în limbajul C, am rulat algoritmul metodei *eliminării succesive a lui Gauss* (cunoscută și ca *metoda pivotării*) pentru un sistem liniar pătratic, folosind o buclă de măsurare a timpului constând din 10^6 pași, pentru următoarele ordine ale sistemului:

n = {3, 4, 5, 8, 12}

Am obținut, respectiv, următorii timpi aproximativi de calcul raportați în secunde, ca *medie aritmetică* a trei rulări succesive pentru fiecare caz:

t(n) = {0.285, 0.554, 1.067, 3.093, 10.757} (sec)

Practic, pot spune că acesta este un timp relativ redus: de ordinul zecilor de secunde.

Ceea ce este mai important de observat aici este modul de evoluție a timpului de calcul în funcție de ordinul matricei, *t=f(n)*. Strict vizual (conform figurii 1) această variație se apropie fie de o evoluție *exponențială* fie de una de tip *geometric*. Trecând datele de mai sus printr-un algoritm de *regresie* (de asemenea cunoscut și ca *optimizare)*, pe baza calculului *coeficientului de corelație* (notat în continuare cu r_{xy}) putem găsi într-adevăr funcția cu variația cea mai apropiată de cea cunoscută (vezi secțiunea *Algoritmi de regresie aplicați asupra timpului de calcul*).

Aceste rezultate s-au obținut pe laptop-ul personal, având configurația hardware următoare:

- Intel® Core™ 2 CPU (2 cores), model: T5600, frecvența de tact: 1.83 GHz
- CPU cache: 2048 KB (2 MB)
- Memoria totală: 3082424 KB (3 GB)
- Spațiu stocare, partiția Linux (disc dur, SATA II): 260.000.000 KB (260 GB)

Timpul (în secunde) l-am determinat prin convertirea la tipul 'real dublă precizie' (în limbajul C acesta este double) a *diferenței* câmpurilor tv_sec (aparținând celor două variabile de tip struct timespec), unul reținând momentul de timp de la start și celălalt pe cel de stop:

(double)(ts_stop.tv_sec–ts_start.tv_sec)

Câmpurile tv_sec folosite mai sus au tipul de dată nativ time_t, care din punctul de vedere al sistemelor Linux/Unix și a celor compatibile POSIX este implementat ca *întreg cu semn* (uzual pe 32 sau 64 de biți). Folosind un tip întreg n-aș putea să afișez în mod direct părțile fracționare ale timpului: zecimi, sutimi și miimi de secundă.

În fișierul antet Linux /usr/include/bits/types.h referit și de /usr/include/time.h există următoarea definiție de tip [4]:

typedef long int __time_t;

Prin urmare am de-a face cu un număr din mulțimea **Z** (long int este doar unul dintre tipurile de dată întregi cu semn oferite în limbajul C). Acesta reprezintă numărul de secunde scurse din momentul de început (la t=0) denumit *epoca Unix* (Unix epoch) marcat ca *miezul nopții, în timp universal, al datei de* 1 Ianuarie 1970 (fără a contoriza 'secundele bisecte' denumite și 'secunde intercalate' - *leap seconds*) [5] [6]. Secunda intercalată este o secundă adăugată timpului universal coordonat (UTC) pentru a permite timpului universal definit de ceasurile atomice (*TAI = Timpul atomic internațional*) să compenseze avansul față de ritmul de rotație al Pământului și să fie cât mai aproape de timpul solar mediu [7].

Utilitatea numerelor reale în măsurătoarea efectuată de mine apare ca evidentă din moment ce am dorit ca timpul să conțină atât secundele cât și fracțiunile de secundă (zecimi, sutimi și miimi) scurse în măsurătoare.

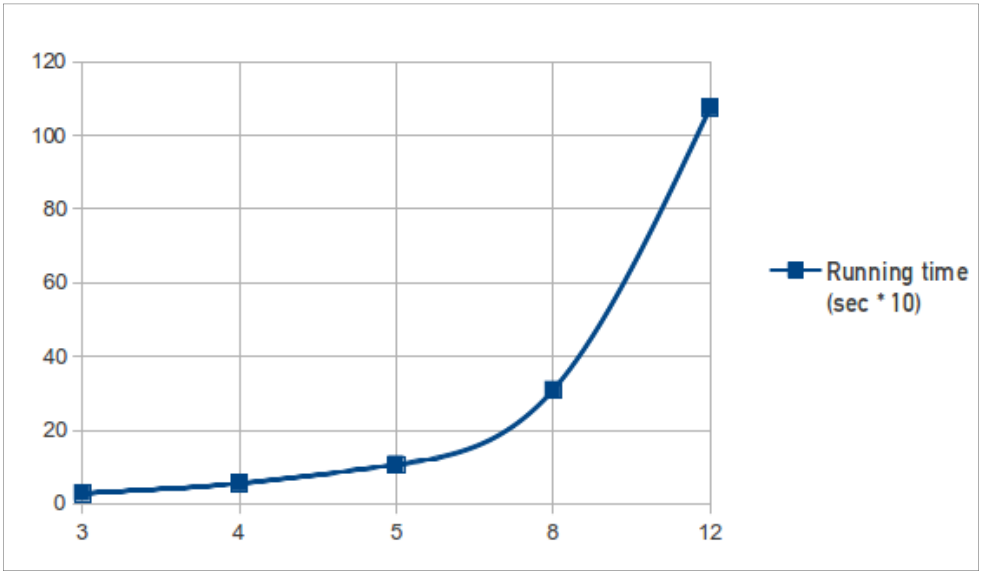

Fig. 1. Timpul necesar rezolvării unui sistem algebric liniar folosind metoda eliminării lui Gauss. Pe **abscisă**: ordinul matricei folosite în calcul. Pe **ordonată**: timpul în 'secunde·10'.

3.1. Algoritmi de regresie aplicați asupra timpului de calcul

Pentru a determina mai exact genul de variație al curbei din figura 3 am preluat ca intrare pentru o serie de *algoritmi de regresie* (*optimizare*) setul de date anterior, definit prin relația:

$$t = f(n)$$

ce oferă timpul de calcul în funcție de ordinul sistemului [8].

M-a interesat coeficientul de corelație (r_{xy}) pentru fiecare algoritm de regresie, pentru că acesta este cel care oferă o imagine clară a *potrivirii* uneia sau alteia dintre funcțiile cu care se face aproximarea peste setul de date de intrare.

În urma calculelor pe care le-am făcut, am determinat *coeficienții de corelație* ai următorilor algoritmi de regresie:

- liniară ($y=a\cdot x+b$) =>
  r_{xy}: 0.9583
- geometrică ($y=a\cdot x^b$) =>
  r_{xy}: 0.9985
- hiperbolică ($1/(a\cdot x+b)$) =>

r_{xy}: -0.8186

- trigonometrică ($a+b\cdot\cos(\omega\cdot x)$) =>

r_{xy}: 0.8083

- exponențială ($y=a\cdot b^x$) =>

r_{xy}: 0.9881

Tabelul 3 prezintă sintetizat aceste date. Am marcat **explicit** cel mai mare coeficient de corelație, care îmi dă tipul de variație dominant.

În urma sortării descrescătoare a valorilor obținute pentru coeficientul de corelație (vezi tabelul 3) concluzionez că, pe baza setului de date de intrare ce a determinat timpul de calcul a cărui variație o am în figura 1, acest timp de calcul se aseamănă cel mai bine cu o *funcție de tip geometric* [9], având forma generală:

$$y = a \cdot x^b$$

Pentru aceasta am calculat și valorile concrete ale coeficienților a și b:

a=0.0158; b=2.5935

Tabel 3. Coeficienții de corelație ai principalilor algoritmi de regresie

Tipul de regresie	Coeficientul de corelație calculat (r_{xy})
Liniară	0.9583
Geometrică	**0.9985**
Hiperbolică	-0.8186
Trigonometrică	0.8083
Exponențială	0.9881

4. Concluzii

Există așadar o dependență cel puțin pătratică (b>2) a timpului de calcul în funcție de ordinul sistemului liniar algebric. Prin aceasta vreau să spun că, pentru a încerca o decriptare (forțată) a datelor criptate cu algoritmul Hill, evident fără a cunoaște cheia de criptare, pe lângă timpul de identificare a celor n^2 perechi {text în clar <-> text cifrat} este nevoie și de rezolvarea unui sistem liniar algebric, pentru care este necesar un timp de calcul cu evoluție geometrică.

Prin *copia simetrică* utilizată în acest studiu de caz ofer posibilitatea refacerii informației în urma coruperii fișierului arhivă original. Se poate reface informația existentă la momentul generării ultimei copii simetrice, adică trebuie menținută o sincronizare permanentă între operația de compresie și cea de actualizare a copiei simetrice. Pentru a reuși restaurarea datelor corupte, utilizatorul trebuie să fie conștient cu privire la următorul compromis: utilitatea și oportunitatea restaurării datelor în detrimentul (minimal) al spațiului de stocare necesar copiei simetrice. O alegere educată este, până la urmă, necesară într-o asemenea situație.

O analogie se poate face între această abordare și cea de '*restoration points*' uzual folosită de sistemele de operare din familia Windows. Multe dintre soluțiile pe care Microsoft le aduce cu titlu de noutate și deci interpretabile ca facilități oferite utilizatorului final provin și din politica de stabilitate a sistemului care se poate spune că este afectată major de atacurile de tip virus, *malware* și mai recent *spyware*.

Deși pentru înlesnirea calculelor se cere ca lungimea spațiului modulo-m să fie număr prim, în studiul de caz prezentat am ales o dimensiune diferită de un număr prim, aleasă

în conformitate cu cerințele practice specifice impuse aplicației și în consecință adaptat acesteia.

Contextul de față (prin copia simetrică) nu cere un algoritm de criptare cu o putere mai mare. Am avut în vedere acest aspect prin modularizarea codului. În acest fel permit înlocuirea modulului asociat algoritmului cifrului Hill cu alte soluții mai performante, specifice unor medii de lucru având cerințe mai stricte de securitate (precum aplicații business, guvernamentale sau militare).

Bibliografie

[1]. M. Eisenberg, *Hill Ciphers and Modular Linear Algebra*, 1999.

[2]. GNU ORG. (2013, Jun. 17) [Online] Available: www.gnu.org/licenses/gpl.html

[3]. K. Avinash, *Classical Encryption Techniques*, Purdue University, 2012.

[4]. Stackoverflow. (2013, Jun. 15) [Online] Available: http://www.stackoverflow.com/questions/471248/what-is-ultimately-a-time-t-typedef

[5]. D. Finkleman *et al.*, "The Future of Time: UTC and the Leap Second," *American Scientist*, vol. 99, no. 4, pp. 312-319, Jul. 2011.

[6]. Wikipedia. (2013, Jun. 11) [Online] Available: http://en.wikipedia.org/wiki/Leap_second

[7]. G. Chiș, *Astronomie. Manual pentru clasa a XII-a*, București: Editura didactică și pedagogică, 1998.

[8]. A. C. Onwutalobi, *Using Encryption Technique*, Department of Computer Science, University of Wollongong Australia, 2005.

[9]. M. R. Bloch and J. N. Laneman, "Secrecy from Resolvability," in *Proc. IEEE Trans. Inf. Theory*, [Online] Available: http://arxiv.org/abs/1105.5419