


Chapter 7

Lecture 7 - Errors, Encryption


Paradigm shift...

Progress report

- You have a progress report due tomorrow. 1-2 pages, neat and tidy. Please include the following information:
  - Your group member's names and Matric ID
  - Your progress so far...
  - Outline proposal for the final project
- If you are having any problems, PLEASE let us know.

Laboratory 2

- Please look at lab 2 - it involves a very small amount of Java programming.
- Pradeep has the booking sheet in the lab.
- Assessment starts next week.

Tut5 Q1:

Consider k *such that* $x = k\phi(n) + (x \bmod \phi(n))$. *We know from Euler that* $a^{\phi(n)} \bmod n = 1$, *and so (of course)* $a^{k\phi(n)} \bmod n = 1$. *We can now apply this to the original formula, and so*

$$\begin{aligned} a^x \bmod n &= a^{k\phi(n)+(x \bmod \phi(n))} \bmod n \\ &= a^{k\phi(n)} \cdot a^{x \bmod \phi(n)} \bmod n \\ &= 1 \cdot a^{x \bmod \phi(n)} \bmod n \\ &= a^{x \bmod \phi(n)} \bmod n \end{aligned}$$

Note also that when $n = pq$, $a^{k\phi(n)+1} \bmod n = a$ *for all* a, *even for the case where* a *and* n *are not relatively prime.*

Tut5 Q2: Access control

	f_1	f_2	f_3	f_4
s_1	`read`		`read`	
s_2		`read`	`read`	`read`
s_3	`read`		`read`	
s_4	`read`	`read`		`read`

Copy the `read` *elements from* $\mathrm{matrix}[\mathrm{row}, \mathrm{col}]$ *to a new* $\mathrm{collusion}[\mathrm{row}, \mathrm{col}]$. *For each subject* s_i, *examine each of the other subjects* $s_{j \neq i}$, *and for each of the columns* k *where* $\mathrm{read} \notin \mathrm{matrix}[s_i, f_k]$ *and* l *where* $\mathrm{read} \in \mathrm{matrix}[s_i, f_l]$, *if* $\mathrm{read} \in \mathrm{matrix}[s_{j \neq i}, f_k]$ *and* $\mathrm{write} \in \mathrm{matrix}[s_{j \neq i}, f_l]$, *then add* `read` *to* $\mathrm{collusion}[s_i, f_k]$.

Tut5 Q3: Carmichael numbers

1. *The first statement is true: the value* $w^{p-1} \bmod p \neq 1$ *is sufficient evidence to claim that* p *is not a prime.*

2. *However, there are special numbers called Carmichael numbers, which are not primes, but exhibit the property that* $w^{p-1} \bmod p = 1$ *if* w *has no factors in common with the prime factors of* p*.*

3. *If we have been unlucky enough to choose a Carmichael number, then the test will often succeed, and of course* p *is not a prime.*

Tut5 Q3: MCQ questions

Question 26: ... $(p+q) \bmod \phi(pq)$ is...

Answer: *Primes must be odd, and so* $\phi(pq)$ *and* $p+q$ *must both be even. So* $(p+q) \bmod \phi(pq)$ *must be even.*

Question 32: A *field* must be of size...

Answer: *There are fields, and finite fields. In this question, a field could be infinite, and so (D) is the answer.*

Question 33: A possible size for a finite field is...

Answer: *Finite fields must be of size* p^n*... (A)*

Error detection

The history of human opinion is scarcely anything more than the history of human errors. [Voltaire]

Simple check codes

✔ Transmit data:

1	65	3	22	47	2

✔ Transmit data+checksum[10]:

1	65	3	22	47	2	140

[Example check program]

[10] checksum=fi ngerprint=message-digest

One-way parity

A	0	1	0	0	0	0	0	1
0	0	0	1	1	0	0	0	0
D	0	1	0	0	0	1	0	0
B	0	1	0	0	0	0	1	0
B	0	1	0	0	0	0	1	0
C	0	1	0	0	0	0	1	1
Check:	0	1	1	1	0	1	1	0

Two way parity

A	0	1	0	0	0	0	0	1	0
0	0	0	1	1	0	0	0	0	0
D	0	1	0	0	0	1	0	0	0
B	0	1	0	0	0	0	1	0	0
B	0	1	0	0	0	0	1	0	0
C	0	1	0	0	0	0	1	1	1
Check:	0	1	1	1	0	1	1	0	X

Simple check codes

- ✔ Parity of bits - detects all 1 bit errors, but...
- ✔ Horizontal and vertical parity - better, but problems with repetitive errors
- ✔ Sum of values - problems with repetitive errors
- ✔ Want better level of error checking

Cyclic redundancy check codes

Treat the stream of transmitted bits as a representation of a polynomial with coefficients of 1:

$$10110 = x^4 + x^2 + x^1 = F(x)$$

Checksum bits are added to ensure that the final composite stream of bits is divisible by some other polynomial $g(x)$.

Cyclic redundancy check codes

- ✔ We can transform any stream $F(x)$ into a stream $T(x)$ which is divisible by $g(x)$.
- ✔ If there are errors in $T(x)$, they take the form of a difference bit string $E(x)$ and the final received bits are $T(x) + E(x)$.
- ✔ When the receiver gets a correct stream, it divides it by $g(x)$ and gets no remainder.

Cyclic redundancy check codes

The question is: *How likely is that $T(x) + E(x)$ will also divide with no remainder?*

Single bits? - No a single bit error means that $E(x)$ will have only one term (x^{1285} say). If the generator polynomial has $x^n + ... + 1$ it will never divide evenly.

Multiple bits? - Various generator polynomials are used with different properties. Must have one factor of the polynomial being $x^1 + 1$, because this ensures all odd numbers of bit errors (1,3,5,7...).

Some common generators:

- **CRC-12** - $x^{12} + x^{11} + x^3 + x^2 + x^1 + 1$
- **CRC-16** - $x^{16} + x^{15} + x^2 + 1$
- **CRC-32** - $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+1$
- **CRC-CCITT** - $x^{16} + x^{12} + x^5 + 1$

Long division is easy!

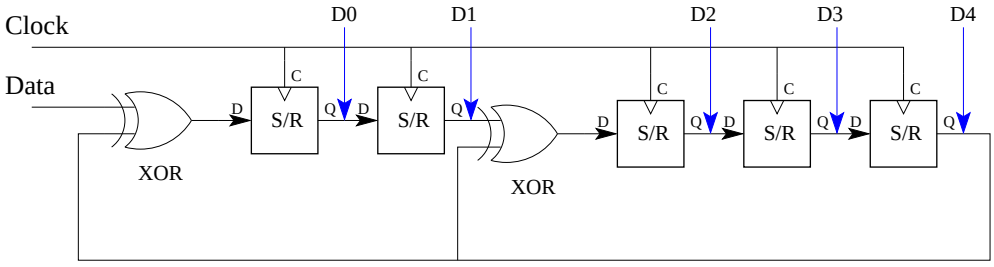
Generator $g(x)$: $x^5 + x^2 + 1$ (100101) and $F(x)$: 101101011. divide $F(x)$ by $g(x)$, and the remainder is appended to $F(x)$ to give $T(x)$:

```
          1010.01000
100101 )101101011.00000
        100101
         100001
         100101
            1001.00
             1001.01
                 1000
```

$T(x) = 10110101101000$.

Long division is easy!

When this stream is received, it is divided but now will have no remainder if the stream is received without errors.



Long division is easy!

Input data	D4	D3	D2	D1	D0	Note
...	0	0	0	0	0	Initial state
1	0	0	0	0	1	First bit
0	0	0	0	1	0	Second bit
1	0	0	1	0	1	Third bit
1	0	1	0	1	1	
0	1	0	1	1	0	
1	0	1	0	0	0	
0	1	0	0	0	0	
1	0	0	1	0	0	
…						

Long division is easy!

Input data	D4	D3	D2	D1	D0	Note
...						
1	0	1	0	0	1	
0	1	0	0	1	0	
0	0	0	0	0	1	
0	0	0	0	1	0	
0	0	0	1	0	0	
0	0	1	0	0	0	

Case study: ethernet

Ethernet is used for networking computers, principally because of its speed and low cost. The maximum size of an ethernet frame is 1514 bytes[11], and a 32-bit FCS is calculated over the full length of the frame.

The FCS used is:

- **CRC-32** - $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+1$

[11] 1500 bytes of data, a source and destination address each of six bytes, and a two byte type identifi er. The frame also has a synchronizing header and trailer which is not checked by a CRC.

Md5SUM

```
hugh@sf0:~[508]$ md5sum ss.c
550114bc3cc3359e55ba33abe8983a85  ss.c
hugh@sf0:~[509]$ cp ss.c XXX.c
hugh@sf0:~[510]$ md5sum XXX.c
550114bc3cc3359e55ba33abe8983a85  XXX.c
hugh@sf0:~[511]$
```

- Message digest, checksum, hash, digital fingerprint.
- A one-way function.

MD5 weaknesses

- ✔ Suspicion that MD5 may have cryptographic weaknesses.
- ✔ Recent revelation (but no details beyond examples) at Crypto2004 of a generated MD5 collision:

  http://eprint.iacr.org/2004/199.pdf

- ✔ Note that this does not reduce the effectiveness of MD5 (yet)

Simple error correction

Methods used to correct errors:

- Ignore errors, while acknowledging correct data. ***ARQ*** (for **A**utomatic **R**epeat re**Q**uest).
- Error correcting codes (for computer memory)

Code types

We can divide error correcting codes (ECC) into continuous and block-based types. Convolutional encodings are used for continuous systems, and the common block-based codes are:

- Hamming codes (for correcting single bit errors),
- Golay codes (for correcting up to three bit errors), and
- Bose-Chaudhuri-Hocquenghem (***BCH***) codes (for correcting block errors).

Combining error correcting codes

- ✔ Different types of error correcting codes can be combined to produce composite codes.
- ✔ For example, *Reed-Solomon* block-codes are often combined with convolutional codes to improve all-round performance.
- ✔ In this combined setup, the convolutional code corrects randomly distributed bit errors but not bursts of errors while the *Reed-Solomon* code corrects the burst errors.

BER and noise

System	(BER) Bit Error Rate (errors/bit)
Wiring of internal circuits	10^{-15}
Memory chips	10^{-14}
Hard disk	10^{-9}
Optical drives	10^{-8}
Coaxial cable	10^{-6}
Optical disk (CD)	10^{-5}
Telephone System	10^{-4}

BER and noise

We can determine the theoretical channel capacity C knowing the SNR:

- BER is 0.01, channel capacity $C \simeq 0.92$ bits/symbol.
- BER is 0.001, channel capacity $C \simeq 0.99$ bits/symbol.
- BER is 0 , channel capacity $C = 1$ bits/symbol (perfect channel)

The theoretical maximum channel capacity is quite close to the *perfect* channel capacity, even if the BER is high.

Reducing BER

- ✔ Increase the signal (power), or
- ✔ Reduce the noise (often not possible), or
- ✔ Use ECC.

The benefit of Error Correcting Codes is that they can *improve* the BER without increasing the transmitted power. This performance improvement is measured as a system ***gain***.

Reducing BER

Example: Consider a system without ECC giving a BER of 0.001 with a *S/N* ratio of *30dB* (1000:1). If we were to use an ECC *codec*, we might get the same BER of 0.001 with a S/N ratio of *20dB* (100:1).

We say that the system gain due to ECC is *10dB* (10:1).

Bad ECC scheme: repetition

Correct transmission errors by repeating bits[12].

```
Data:     0  1  0  0  1  1  1  1  ...
Transmit: 000111000000111111111111...
```

If we send three identical bits for every bit we wish to transmit, we can then use a voting system to determine the most likely bit. If our natural BER due to noise was 0.01, with three bits we would achieve a synthetic BER of 0.0001, but our channel capacity is reduced to about $C = 0.31$ bits/symbol.

[12]Note: there is no point in repeating bits *twice*. you must repeat three times, or 5 times, and then vote to decide the best value.

Bad ECC scheme: repetition

- ✔ We can see from this that the rate of transmission using *repetition* has to approach zero to achieve more and more reliable transmission.
- ✔ However we know that the theoretical rate should be equal to or just below the channel capacity C.
- ✔ Convolutional and other encodings can achieve rates of transmission close to the theoretical maximum.

ECC scheme: Hamming

- ✔ *Hamming* codes are block-based error correcting codes.
- ✔ We add hamming bits to a string
- ✔ Here we derive the inequality used to determine how many extra *hamming* bits are needed for an arbitrary bit string.

ECC scheme: Hamming

The *hamming* distance is a measure of how FAR apart two bit strings are.

```
A:          0 1 0 1 1 1 0 0 0 1 1 1
B:          0 1 1 1 1 1 1 0 0 1 0 1
A XOR B:    0 0 1 0 0 0 1 0 0 0 1 0
```

(In this case: 3)

ECC scheme: Hamming

If we had two bit strings X and Y representing two characters, and the *hamming* distance between any two codes was d, we could turn X into Y with d single bit errors.

- If we had an encoding scheme (for say ASCII characters) and the minimum *hamming* distance between any two codes was $d + 1$, we could ***detect*** d single bit errors[13].
- We can ***correct*** up to d single bit errors in an encoding scheme if the minimum *hamming* distance is $2d + 1$.

[13] Because the code d bits away from a correct code is not in the encoding.

ECC scheme: Hamming

Example encoding. The hamming distance between any two codes is at least 2, and we use 3 bits to encode 4 symbols. (c.f. 8 symbols normally)

```
A:      0 0 0
B:      0 1 1
C:      1 0 1
D:      1 1 0
```

The minimum hamming distance between any two codes is 2. We can detect single bit errors.

ECC scheme: Hamming

If we now encode m bits using r extra *hamming* bits to make a total of $n = m + r$, we can count how many correct and incorrect *hamming* encodings we should have. With m bits we have 2^m unique messages - each with n illegal encodings, and:

$$\begin{aligned} (n+1)2^m &\leq 2^n \\ (m+r+1)2^m &\leq 2^n \\ m+r+1 &\leq 2^{n-m} \\ m+r+1 &\leq 2^r \end{aligned}$$

ECC scheme: Hamming

We solve this inequality, and then choose R, the next integer larger than r.

Example: If we wanted to encode 8 bit values ($m = 8$) and be able to *correct* single bit errors:

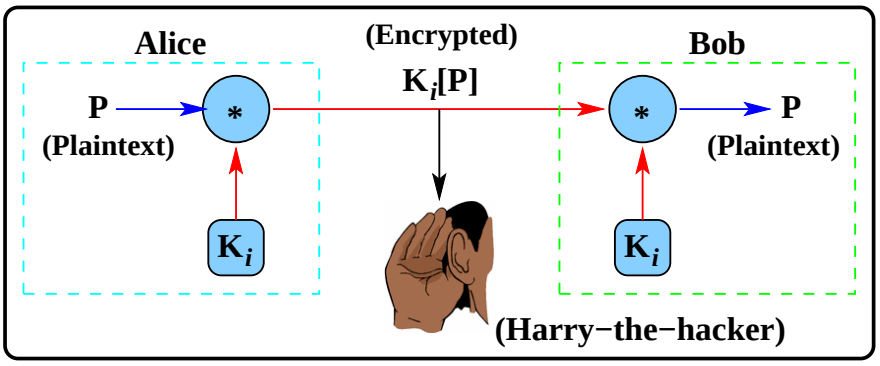
$$\begin{aligned} 8 + r + 1 &\leq 2^r \\ 9 &\leq 2^r - r \\ r &\simeq 3.5 \\ R &= 4 \end{aligned}$$

Encryption introduction

I could have told her the truth - that the same calculation which had served me for deciphering the manuscript had enabled me to learn the word - but on a caprice it struck me to tell her that a genie had revealed it to me. This false disclosure fettered Madame d'Urfé to me. That day I became the master of her soul, and I abused my power. [Casanova]

We call these systems *symmetric* key systems...

Symmetric key systems



Simple ciphers - transposition

Transposition ciphers just re-order the letters of the original message. This is known as an anagram:

- *parliament* is an anagram of *partial men*
- *Eleven plus two* is an anagram of *Twelve plus one*

Perhaps you would like to see if you can unscramble "*age prison*", or "*try open*".

Transposition

- ✔ Detect a transposition cipher with the frequencies of the letters, and letter pairs.
- ✔ If the frequency of single letters in ciphertext is correct, but the frequencies of letter pairs is wrong, then the cipher may be a transposition.
- ✔ This sort of analysis can also assist in unscrambling a transposition ciphertext, by arranging the letters in their letter pairs.

Simple ciphers - substitution

- ✔ Substitution cipher systems encode the input stream using a substitution rule.
- ✔ The Cæsar cipher is an example of a simple substitution cipher system, but it can be *cracked* in at most 25 attempts by just trying each of the 25 values in the keyspace. The original Cæsar cipher can be *cracked* in 1 attempt.

Substitution

Code	Encoding
A	Q
B	V
C	X
D	W
...	...

If the mapping was more randomly chosen it is called a monoalphabetic substitution cipher, and the keyspace for encoding 26 letters would be $26! - 1 = 403,291,461,126,605,635,583,999,999$.

Substitution

- ✔ If we could decrypt $1,000,000$ messages in a second, then the average time to find a solution would be about $6,394,144,170,576$ years!
- ✔ We might be lulled into a sense of security by these big numbers, but of course this sort of cipher can be subject to frequency analysis.

Frequency analysis

In the English language, the most common letters are: "E T A O N I S H R D L U..." (from most to least common), and we may use the frequency of the encrypted data to make good guesses at the original plaintext.

✔ We may also look for *digrams* and *trigrams* (th, the).

Frequency analysis algorithm

1. Count occurences of each character in as much ciphertext as you can
2. Arrange letters in frequency order
3. Replace letters by the standard ETAOINSHRDLU ones:

Frequency analysis

```
EV YQS CVV MIWK FRPC FRQF FRV IQFV WM
FIQSCKPCCPWS ACPSN IVJVFPFPWS RQC FW
QJJIWQYR ZVIW FW QYRPVDV KWIV QSB KWIV
IVTPQXTV FIQSCKPCCPWS. RWEVDVI EV LSWE
FRQF FRV FRVWIVFPYQT IQFV CRWATB ...
```

V occurs most often, F next and so on, so replace V with E, F with T...

Frequency Analysis

```
EV YQS CVV MIWK FRPC FRQF FRV IQFV WM
-E -A- -EE F-O- THI- THAT THE -ATE OF

FIQSCKPCCPWS ACPSN IVJVFPFPWS RQC FW
T-A---I--IO- --I-- -E-ETITIO- HA- TO

QJJIWQ YR ZVIW FW QYRPVDV KWIV QSB KWIV
A---OA -H -E-O TO A-HIE-E -O-E A-- -O-E
```

Vigenère

- ✔ The Vigenère cipher is a polyalphabetic substitution cipher invented around 1520.
- ✔ We use an encoding/decoding sheet, called a *tableau*, and a keyword or key sequence.

Vigenère

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**...**
A	A	B	C	D	E	F	G	H	...
B	B	C	D	E	F	G	H	I	...
C	C	D	E	F	G	H	I	J	...
D	D	E	F	G	H	I	J	K	...
E	E	F	G	H	I	J	K	L	...
F	F	G	H	I	J	K	L	M	...
G	G	H	I	J	K	L	M	N	...
H	H	I	J	K	L	M	N	O	...
...	...	...	...	...	...	...	...	...	...

Vigenère

If our keyword was BAD, then encoding HAD A FEED would result in

Key	B	A	D	B	A	D	B	A
Text	H	A	D	A	F	E	E	D
Cipher	I	A	G	B	F	H	F	D

If we can discover the length of the repeated key (in this case 3), and the text is long enough, we can just consider the cipher text to be a group of interleaved monoalphabetic substitution ciphers and solve accordingly.

Analysis

The index of coincidence is the probability that two randomly chosen letters from the cipher will be the same, and it can help us discover the length of a key

$$IC = \frac{1}{N(N-1)} \sum_{i=0}^{25} F_i(F_i - 1)$$

where F_i is the frequency of the occurences of symbol i and N is the length of the cipher.

Index of coincidence

Period:	1	2	3	4	...
Expected IC	0.066	0.052	0.047	0.045	...

```
#!/usr/bin/perl
$skip=$ARGV[0] ;
@text=<stdin> ;
$all=join(",@text) ;
$all =~ tr/a-z/A-Z/ ;
$all =~ tr/A-Z//cd ;
$header=substr($all,0,$skip) ;
$shifted = substr($all,$skip).$header ;
@alltxt=split(//,$all) ; @shiftxt=split(//,$shifted) ;
foreach $i(0..$#alltxt){
     if($alltxt[$i] eq $shiftxt[$i]) { $count++ ;}
}
printf("Index of Coincidence is: %2f\n",$count/$#alltxt) ;
```


Index of coincidence

- ✔ The ideas here were developed by William F. Friedman in his Ph.D.
- ✔ Friedman also coined the words "cryptanalysis" and "cryptology".
- ✔ Friedman worked on the solution of German code systems during the first (1914-1918) world war, and later became a world-renowned cryptologist.