COMP3260 Data Security

Lecture 2

Prof Ljiljana Brankovic

Lecture Overview

1. Euclid's algorithm for computing gcd
2. Extended Euclid's algorithm to compute multiplicative inverses
3. Euler's totient function
4. Solving general equations
5. Example
6. Galois Fields
7. Introduction to Information Theory
8. Entropy
9. Theoretical Secrecy

Number Theory and Finite Fields

- Text: Chapter 2 “Introduction to Number Theory“ [1]
- Text: Chapter 5 “Finite Fields” [1]
- “Cryptography and Data Security” by D. Denning [2]

Note that in-text references and quotes are omitted for clarity of the slides. When you write as essay or a report it is very important that you use both in-text references and quotes where appropriate.

Euclid's algorithm for computing gcd

Euclud's algorithms is based on the fact that if a and b, $a > b$, are both divisible by c, so is $a \bmod b$.

Indeed, if we write a as

$$a=kb+r$$

and if both a and b are divisible by c, we have

$$hc=klc + r$$

then $r=(h-kl)c$ and thus r is also divisible by c.

Euclid's algorithm for computing gcd

For example, 100 and 22 are both divisible by 2, and so is 100 mod 22 = 12.

Therefore, instead of looking for the greatest common divisor (gcd) of 100 and 22, we can look for the greatest common divisor of 22 and 12; and so on…

100, 22
22, 12
12, 10
10, 2
2, 0 ← when we hit "0 ", the previous value is the gcd(a.b)

Euclid's algorithm for computing gcd

```
Input: a, n
Output: None
gcd(a,n) {
      g[0] = n
      g[1] = a
      i = 1
      while (g[i] ≠  0){
            g[i+1] = g[i-1] mod  g[i]
            i = i +1
      }
      return g[i-1]
}
```

Euclid's algorithm extended to compute inverses

```
Input: a, n
Output: None
inv(a,n) {
      g[0] = n; g[1] = a
      u[0] = 1; u[1] = 0
      v[0] = 0; v[1] = 1
      i = 1
      while (g[i] ≠  0) // “g[i] = u[i]n + v[i]a”
      {
            y = g[i-1] / g[i]  //integer division
            g[i+1] = g[i-1] – y × g[i]
            u[i+1] = u[i-1] – y × u[i]
            v[i+1] = v[i-1] – y × v[i]
            i = i +1
      }
      if v[i-1] ≥ 0 then  return v[i-1] else return v[i-1] + n
}
```

Euclid's extended algorithm: 3x mod 10 =1

i	y	g	u	v
0		10	1	0
1		3	0	1
2	3	1	1	-3
3	3	0		

```
Input: a, n
Output: None
inv(a,n) {
       g[0] = n; g[1] = a
       u[0] = 1; u[1] = 0
       v[0] = 0; v[1] = 1
       i = 1
       while (g[i] ≠  0) // “g[i] = u[i]n + v[i]a”
       {
              y = g[i-1] / g[i]  //integer division
              g[i+1] = g[i-1] – y × g[i]
              u[i+1] = u[i-1] – y × u[i]
              v[i+1] = v[i-1] – y × v[i]
              i = i +1
       }
       if v[i-1] ≥ 0 then  return v[i-1] else return v[i-1] + n
}
```

Euler's Totient Function

Definitions:

A positive integer *p* is a prime number iff *p>1* and the only positive integer divisors of *p* are *1* and *p*.

Two integers *n* and *m* are relatively prime iff *gcd(n,m)=1*.

Fundamental Theorem of Number Theory:

Every positive integer $n > 1$ can be written uniquely in the form

$$n = p_1^{e_1} p_2^{e_2} \dots p_t^{e_t}$$

where p_i is a prime number and $p_1 < p_2 < \dots < p_t$.

Euler's Totient Function

- For every integer n, the Euler's totient function $\phi(n)$ is the number of positive integers less than or equal to n which are relatively prime to n.

- If n is prime then $\phi(n) = n-1$.

- If n is the product of two primes, $n = p \times q$ then $\phi(n) = (p-1) \times (q-1)$.

Euler's Totient Function

- In general,
  1. $\phi(1) = 1$
  2. For $n > 1$, if $n = p_1^{e_1} p_2^{e_2} \ldots p_t^{e_t}$

     then $$\phi(n) = \prod_{i=1}^{t} p_i^{e_i - 1}(p_i - 1).$$

Examples:

- $\phi(5) = 5 - 1 = 4$
- $\phi(10) = \phi(5\times2) = (5-1) \times (2-1) = 4$
- $\phi(40) = \phi(5\times2^3) = (5-1) \times 2^2\times(2-1) = 4\times4\times1 = 16$

Euler's Totient Function

Fermat's Little Theorem: Let p be a prime. Then for every a such that $gcd(a,p) = 1$, $a^{p-1} \bmod p = 1$.

Euler's generalization: For every a and n such that $gcd(a,n)=1$, $a^{\phi(n)} \bmod n =1$.

Euler's generalization gives us an algorithm for finding multiplicative inverses, that is, for solving equations of the type $ax \bmod n = 1$. The inverse (x) is given by

$$x = a^{\phi(n)-1} \bmod n.$$

$$ax = a\, a^{\phi(n)-1} \bmod n = a^{\phi(n)-1\ +1} \bmod n = a^{\phi(n)} \bmod n = 1$$

General equations

Solving general equations of the form $ax \bmod n = b$:

When $gcd(a,n) = 1$, find solution x_0 to $ax \bmod n = 1$;
$ax_0 \bmod n = 1$ implies $abx_0 \bmod n = b$ and $x = bx_0 \bmod n$.

When $gcd(a,n) = g$:

- If g divides b, that is, $b \bmod g = 0$, $ax \bmod n = b$ has g solutions of the form
  $x = ((b/g)x_0 + t(n/g)) \bmod n$, for $t=0,1,\ldots,g-1$,
  where x_0 is the solution to $(a/g)x \bmod (n/g) = 1$.
- If g does not divide b then there are no solutions.

General equations

Example: Solve $6x \bmod 10 = 4$.

$g = \gcd(6,10) = 2$ and 2 divides 4 so there are 2 solutions.

Compute x_0 from $(6/2)x \bmod (10/2) = 1$. Get $x_0 = 2$.

Now calculate the two solutions:

$t=0$ gives $x = 2(4/2) + 0\,(10/2) \bmod 10 = 4$

$t=1$ gives $x = 2(4/2) + 1(10/2) \bmod 10 = 9$

Check:

$6 \times 4 \bmod 10 = 24 \bmod 10 = 4$

$6 \times 9 \bmod 10 = 54 \bmod 10 = 4$

Example

Solve *31x mod 42 = 1*.

We shall use 3 different techniques to solve this equation and find multiplicative inverse of *31* modulo *42*.

1. Euclid's extended algorithm for gcd
2. Chinese Remainder Theorem
3. Euler Totient Function

Euclid's extended algorithm for gcd

```
Algorithm inv(a,n)
begin
    $g_0$ := n; $g_1$ := a; $u_0$ = 1;  $v_0$ := 0;  $u_1$ := 0;

    $v_1$ := 1; i := 1;

    while $g_i \neq 0$ do "$g_i = u_i n + v_i a$"
      begin
        y := $g_{i-1}$ div $g_i$;   $g_{i+1}$ := $g_{i-1}$ - y *$g_i$;

        $u_{i+1}$ := $u_{i-1}$ - y *$u_i$;  $v_{i+1}$ := $v_{i-1}$ - y *$v_i$;

        i := i + 1
      end;

    x := $v_{i-1}$

    if $x \geq 0$ then inv := x else inv := x+n
end
```

i	y	g	u	v
0		42	1	0
1		31	0	1
2	1	11	1	-1
3	2	9	-2	3
4	1	2	3	-4
5	4	1	-14	19
6	2	0		

Chinese Remainder Theorem

$42 = 2 \times 3 \times 7$ so $d_1 = 2$, $d_2 = 3$ and $d_3 = 7$.

We first find solutions x_1 and x_2:

$31x \bmod 2 = 1 \rightarrow x_1 = 1$

$31x \bmod 3 = 1 \rightarrow x_2 = 1$

$31x \bmod 7 = 1 \rightarrow x_3 = 5$

We now apply Chinese reminder theorem to find a common solution to the equations

$x \bmod 2 = x_1 = 1$

$x \bmod 5 = x_2 = 1$

$x \bmod 7 = x_3 = 5$

Chinese Remainder Theorem

First find y_1 and y_2 such that

$(42/2)y_1 \bmod 2 = 1 \rightarrow y_1 = 1$
$(42/3)y_2 \bmod 3 = 1 \rightarrow y_2 = 2$
$(42/7)y_3 \bmod 7 = 1 \rightarrow y_3 = 6$

We now have

$x = (42/2)y_1x_1 + (42/3)y_2x_2 + (42/7)\,y_3x_3 \bmod 42 =$
$= 21 \times 1 \times 1 + 14 \times 1 \times 2 + 6 \times 5 \times 6 \bmod 42 = 19$

Thus *19* is the multiplicative inverse of *31* modulo *42*.

Euler Totient function

- $42 = 2 \times 3 \times 7$
- $\phi(42) = 1 \times 2 \times 6 = 12$
- $x = a^{\phi(n)-1} \bmod n = 31^{11} \bmod 42$

Fast Exponentiation

```
Algorithm fastexp(a, z, n)
begin "return $x = a^z \bmod n$"
  a1:= a; z1 :=z;  x := 1;
  while z1 $\neq$ 0 do
    begin
      while z1 mod 2 = 0 do
        begin "square a1 while
               z1 is even"
          z1 := z1 div 2; a1 :=
               (a1*a1) mod n;
        end;
      z1 := z1 - 1; x := (x*a1)
               mod n;
    end;
  fastexp := x;
end
```

	a	z	x
0	31	11 (1011)	1
1	31	10 (1010)	31
2	37	5 (101)	31
3	37	4 (100)	13
4	25	2 (10)	13
5	37	1 (1)	13
6	37	0 (0)	19

Groups, Rings and Fields

Group

(A1) Closure under addition:	If a and b belong to S, then $a + b$ is also in S
(A2) Associativity of addition:	$a + (b + c) = (a + b) + c$ for all a, b, c in S
(A3) Additive identity:	There is an element 0 in R such that $a + 0 = 0 + a = a$ for all a in S
(A4) Additive inverse:	For each a in S there is an element $-a$ in S such that $a + (-a) = (-a) + a = 0$

Abelian Group

(A5) Commutativity of addition:	$a + b = b + a$ for all a, b in S

Ring

(M1) Closure under multiplication:	If a and b belong to S, then ab is also in S
(M2) Associativity of multiplication:	$a(bc) = (ab)c$ for all a, b, c in S
(M3) Distributive laws:	$a(b + c) = ab + ac$ for all a, b, c in S $(a + b)c = ac + bc$ for all a, b, c in S

Commutative Ring

(M4) Commutativity of multiplication:	$ab = ba$ for all a, b in S

Integral Domain

(M5) Multiplicative identity:	There is an element 1 in S such that $a1 = 1a = a$ for all a in S
(M6) No zero divisors:	If a, b in S and $ab = 0$, then either $a = 0$ or $b = 0$

Field

(M7) Multiplicative inverse:	If a belongs to S and $a \neq 0$, there is an element a^{-1} in S such that $aa^{-1} = a^{-1}a = 1$

Galois fields GF(p) and GF(2^n)

- Real and integer arithmetic is not suitable for cryptography because information is lost through rounding or truncation.

- Also, for efficiency, we prefer to work with integers that have a given number of bits. For that reason we prefer to use modular arithmetic - integers modulo n.

- If n is a prime number, we have a finite field of order p, which is referred to as Galois field, in honour of Évariste Galois, a French mathematician who first studied them.

Galois fields GF(p) and GF(2^n)

- Many recent ciphers are based on arithmetic in a Galois field *GF(p)*, where *p* is a prime number.

- Note that because *p* is a prime, every integer $a \in [1,p-1]$ is relatively prime to *p* and thus has an inverse modulo *p*.

- The set of integers *mod p* together with binary operations 'addition' and 'multiplication' is called a field: it is an integral domain where every element besides 0 has a multiplicative inverse.

- That means that we can do addition, subtraction, multiplication and division without leaving the set.

Galois fields GF(p) and GF(2^n)

- Another type of Galois field used in cryptography is $GF(q^n)$ where elements are polynomials of degree $n-1$ of the form

  $a = a_{n-1}x^{n-1} + \ldots + a_1x + a_0$

  where the coefficients a_i are integers *mod q*

- Each element a is a residue *mod* $p(x)$ where $p(x)$ is an irreducible polynomial of degree n. (Irreducible means that p cannot be factored into polynomials of degree less than n)

Galois fields GF(p) and GF(2^n)

- We are particularly interested in the fields $GF(2^n)$ where the coefficients are binary digits 0 and 1. Computing in $GF(2^n)$ is very efficient and addition and subtraction correspond to $\oplus$ (exclusive-or) of the coefficients. Multiplication can also be done efficiently, but requires dividing by $p(x)$.

- Advanced Encryption Standard (AES) uses arithmetic in the finite field $GF(2^8)$ with the irreducible polynomial $p(x) = x^8 + x^4 + x^3 + x + 1$. Every element a of this field can be represented as bit vector of length 8, e.g., 11011011.

Galois fields $GF(p)$ and $GF(2^n)$

- Computing in $GF(2^n)$ is more efficient than computing in $GF(p)$, where $2^{n-1} < p < 2^n$. Why?

- Space efficient: All n-bit vectors correspond to elements of $GF(2^n)$, which is not the case for $GF(p)$.

- Time efficient: Arithmetic is more efficient in $GF(2^n)$ than in $GF(p)$.

Galois fields GF(p) and GF(2^n)

- Multiplicative inverses:

  Note that in *GF(2^n)*, *p(x)* is irreducible, so it is relatively prime to all polynomials of order *n-1*, but we are not counting the polynomial whose coefficients are 0 (0-vector).

Thus:

$\phi(p(x)) = 2^n-1$

$a^{-1} = a^{\phi(p(x))-1} \bmod p(x) = a^{(2^n)-2} \bmod p(x)$

Alternatively

$a^{-1} = inv(a, p(x))$ (Using Euclid's algorithm).

Galois fields GF(p) and GF(2^n)

1. Let a = 10101 and b = 01100. In GF(2^5), find c = a+b.

   Solution:

```
  a = 1 0 1 0 1
⊕ b = 0 1 1 0 0
  --------------
  c = 1 1 0 0 1
```

2. Let a = 10101 and b = 01100. In GF(31), find c = a+b.

   Solution:

```
a =    1 0 1 0 1        (21)
b =    0 1 1 0 0        (12)
c = 1 0 0 0 0 1         (33)
```

33 mod 31 = 2 hence c = 0 0 0 1 0 (2)

Galois fields GF(p) and GF(2^n)

3. Let a = 10111001 and b = 01101110. In GF(2^8), find c = a-b.

   Solution: c = 11010111

4. Let a = 1101 and b=0101. In GF(2^4), find c = a+b.

   Solution: c = 1000

Galois fields GF(p) and GF(2^n)

5. Let a = 101. In GF(2^3) with irreducible polynomial $p(x) = x^3 + x + 1$ find d=a*a.

Solution: Multiply a * a

```
       101
       101
       ---
       101
      000
     101
     -------
      10001
```

		1	0	1
		1	0	1
		1	0	1
	0	0	0	
1	0	1		
1	0	0	0	1

Divide by p(x) = 1011:

```
     _____
1011)10001
     1011
    ------
       111 = d
```

Galois fields GF(p) and GF(2)

6. Let a = 111 and b = 100. In GF(2^3) with irreducible polynomial p(x) = x^3+x +1 find d=a*b

Solution: Multiply a*b:

```
          1 11
           100
           ---
           000
          000
        1 11
         --------
         1 1100
```

Divide by p(x) = 1011:

```
1011 ) 11100
       1011
         ------
         1010
         1011
         ------
         0001 = d      d=001
```

Galois fields GF(p) and GF(2^n)

7. Let a=100. If GF(2^3) with irreducible polynomial p(x)= $x^3 + x + 1$ find a^{-1} and verify that $a \times a^{-1}$ mod p(x)=1

Solution:

$\phi(p(x)) = 2^3-1 = 7$, $a^{-1} = 100^6 \bmod 1011$

$100^2 = 100 * 100$

```
          100
          100
          -----
          000
         000
        100
       ---------
       10000
```

Divide by p(x) = 1011:

```
          _____
1011 ) 10000
       1011
       -------
         110
```

Hence $100^2 = 110$

Galois fields GF(p) and GF(2^n)

100^4 = 110 * 110

```
   110
   110
   ---
   000
  110
 110
 --------
 10100
```

Divide by p(x) = 1011:

```
1011 ) 10100
       1011
       --------
         010
```

Hence 100^4 = 010

Galois fields GF(p) and GF(2^n)

$100^6 = 100^2 * 100^4 =$ 110 * 010

```
  110
  010
  ---
  000
 110
000
--------
01100
```

Divide by p(x) = 1011:

```
        ____
1011 ) 1100
       1011
       --------
         111
```

Hence $100^6 = 111$

$a^{-1} = 100^6 \bmod 1011 = 111$

Introduction to Information Theory Article on Claude Shannon

Claude Shannon, father of information theory, dies at 84

Murray Hill, N.J. (Feb. 26, 2001) -- Claude Elwood Shannon, the mathematician who laid the foundation of modern information theory while working at Bell labs in the 1940s, died on Saturday. He was 84.

Shannon's theories are as relevant today as they were when he first formulated them. "It was truly visionary thinking," said Arun Netravali, president of Lucent Technologies' Bell labs. "As if assuming that inexpensive, high-speed processing would come to pass, Shannon figured out the upper limits on communication rates. First in telephone channels, then in optical communications, and now in wireless, Shannon has had the utmost value in defining the engineering limits we face."

In 1948 Shannon published his landmark paper
"A mathematical theory of communication."

He begins this pioneering paper on information theory by observing that "the fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at another point." He then proceeds to so thoroughly establish the foundations of information theory that his framework and terminology remain standard.

Shannon's theory was an immediate success with communications engineers and stimulated the technology which led to today's information age.

Another example is Shannon's 1949 paper entitled "*Communication theory of secrecy systems.*" This work is now generally credited with transforming cryptography from an art to a science.

Shannon was born in Petoskey, Michigan, on April 30, 1916. He graduated from the University of Michigan in 1936 with bachelor's degrees in mathematics and electrical engineering. In 1940 he earned both a master's degree in electrical engineering and a Ph.D. in mathematics from the Massachusetts Institute of Technology (MIT).

Shannon joined the mathematics department at Bell Labs in 1941 and remained affiliated with the labs until 1972. He became a visiting professor at MIT in 1956, a permanent member of the faculty in 1958, and a professor emeritus in 1978.

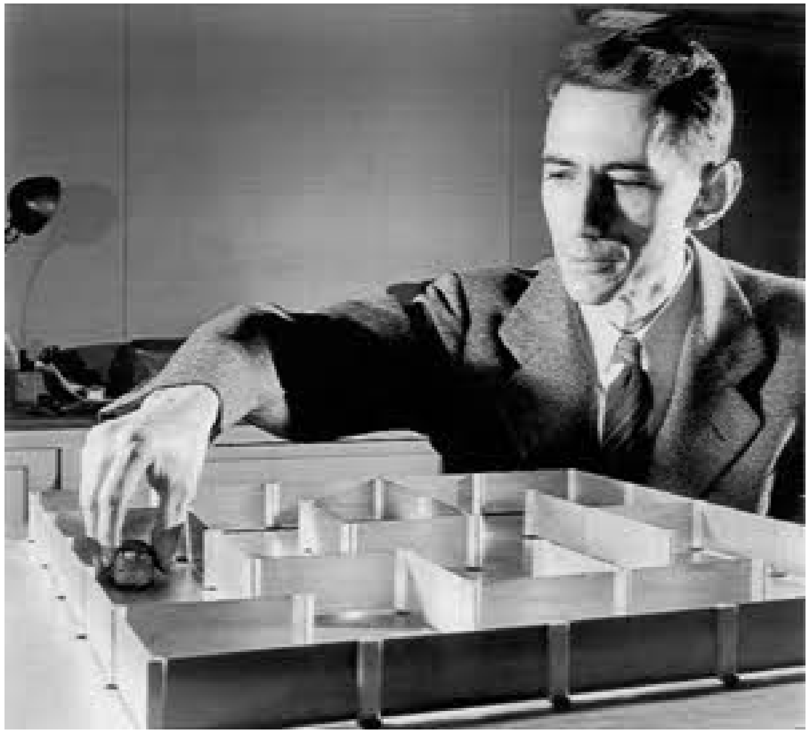
Shannon was renowned for his eclectic interests and capabilities. A favourite story describes him juggling while riding a unicycle down the halls of Bell labs.

He designed and built chess-playing, maze-solving, juggling and mind-reading machines. These activities bear out Shannon's claim that he was more motivated by curiosity than usefulness.

In his words "I just wondered how things were put together."

Claude Shannon

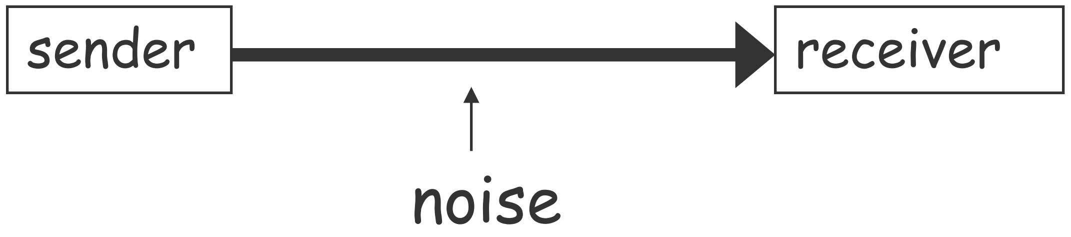
Claude Shannon's clever electromechanical mouse, which he called Theseus, was one of the earliest attempts to "teach" a machine to "learn" and one of the first experiments in artificial intelligence.



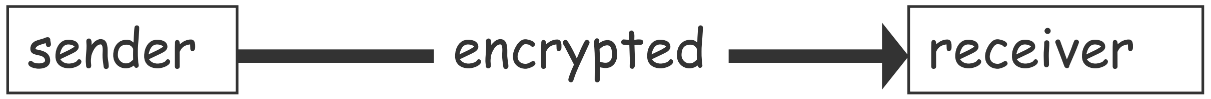
Information Theory

In 1948 Shannon published a paper "A mathematical theory of communication" where he provided a mathematical theory to measure:

1. The **uncertainty** of the receiver of a message passed through a ***noisy channel***



2. The **secrecy** of a message passed through an ***encryption channel***



The two concepts are closely related.

Both need a measure of the **amount of information** or **entropy** or **randomness** of a message.

In a noisy channel problem, a sender transmits a message M over a noisy channel to a receiver. If a distorted message M′ is received, the receiver would like to recover the true message M.

To make this possible, the sender adds redundant bits called error control codes to M in such a way that transmission errors can be corrected, or at least detected so that the receiver can request retransmission.

Analogy Between Noisy Channel and Encryption

- The enciphering transformation corresponds to the noise.

- The ciphertext corresponds to the received message M′.

- The role of the cryptanalyst is similar to the role of the receiver in the noisy channel problem.

- The role of the sender in the two problems is very different:
  - In the noisy channel the objective is to make M directly recoverable from M′.
  - In the secrecy problem the objective is to make recovery of M from M′ infeasible (without the knowledge of the deciphering key).

Entropy

Information theory measures the amount of information in a message by the average number of bits needed to encode all possible messages in an optimal encoding.

The **entropy** of a given message *X* is defined by

$$H(X) = -\sum_{i=1}^{n} p(X_i)\log_2 p(X_i) =$$

$$= \sum_{i=1}^{n} p(X_i)\log_2 \frac{1}{p(X_i)}$$

where

- the sum is over all possible messages $X_1, X_2, \ldots, X_n$
- $p(X_i)$ is the probability that the message X_i is sent.

Entropy

In some sense, the entropy is the minimum size of a compressed version of X.

If the probability distribution is **uniform** then the entropy of the message is the **highest**; the message is random and contains much information.

If the probability distribution is **skewed**, then the entropy of the message is **low**; message is regular and predictable and doesn't tell us much new.

Entropy

Note: each term $\log_2(1/p(X_i))$ represents the number of bits needed to encode message X_i in optimal encoding. The weighted average $H(X)$ gives the expected number of bits in optimally encoded messages.

Because $\log_2(1/p(X_i))$ decreases as $p(X_i)$ increases, an optimal encoding uses short codes for frequently occurring messages and longer codes for infrequent messages. This principle is applied in Morse and Huffman coding.

Examples

Example 1. The messages are about the weather and there are three possible messages: F=fine, S=showers, R=rain.

Each message has the probability of being sent: $p(F)=p(S)=0.25$, $p(R)=0.5$

Note that $\Sigma p(X)$ over all messages X is 1.

The entropy of this set of messages is

$$H(X) = 0.25 \times 2 + 0.25 \times 2 + 0.5 \times 1 = 1.5$$

An optimal encoding assigns 1-bit code to R and 2-bit codes to F and S.

Example 1

For example, R is encoded as 0, S as 10 and F as 11.

Using this encoding, the 8-letter sequence RSRRFRSF is encoded as the 12-bit sequence 010001101011:

```
R  S   R  R  F  R   S   F
0  10  0   0  11 0  10  11
```

The average number of bits per letter is 12/8=1.5.

More precisely, we need to factor in the probability here.

So for RFS example we have:

$$0.25 \times 2 + 0.25 \times 2 + 0.5 X 1 = 1.5 \text{ bits}$$

Summary

1. Euclid's algorithm for computing gcd

2. Finding multiplicative inverses:
   1. Chinese Remainder Theorem
   2. Extended Euclid's algorithm to compute multiplicative inverses
   3. Euler's totient function

3. Solving general equations

1. Introduction to Information Theory: Entropy

Entropy Continued
Example 2

The messages describe penalty shoot-outs; we have g=goal and m=miss, with probabilities

$p(g)=0.9$ and

$p(m)=0.1.$

$$H(X) = -(0.9 \times \log_2(0.9) + 0.1 \times \log_2(0.1))$$

$$= -(-0.13680 - 0.3321) = 0.469$$

Example 3

X is one of the k characters $c_1, c_2, \ldots, c_k$, where each character has probability 1/k.

$$H(X) = -\sum_{1}^{k} \left(\frac{1}{k}\right) \times \log_2 \left(\frac{1}{k}\right) = \log_2 k$$

If k=256 then H(X)=8.

That is, if every character in the set of 256 characters has the same probability then the average number of bits per character is 8.

In general, for $n=2^k$, H(X)=k, that is, k bits are needed to encode each possible message.

Example 4

Suppose $n=1$ and $p(X)=1$. What is the entropy of the message?

$$H(X) = \log_2 1 = 0$$

There is no information in this message because there is no choice.

Entropy - Cont.

Given n messages, H(X) is maximal for

$$p(X_1) = p(X_2) = \ldots = p(X_n) = 1/n,$$

that is, when all messages are equally likely.

H(X) decreases as the distribution of messages becomes more and more skewed, reaching a minimum of $H(X)=0$ when $p(X_i)=1$ for some message X_i.

The entropy of a message measures its **uncertainty**: it gives the number of bits of information that must be learned when the message has been distorted by a noisy channel or hidden in ciphertext.

Theoretical Secrecy

Information theory provides a theoretical foundation for cryptography.

Information theory measures the theoretical **secrecy** of a cipher by the uncertainty about a plaintext given the corresponding ciphertext.

Perfect secrecy is achieved if no matter how much ciphertext is intercepted, nothing can be learned about the plaintext.

The only perfectly secret cipher is **one-time pad**.

Theoretical Secrecy

All other ciphers leave some information about the plaintext in the ciphertext.

As the length of the ciphertext increases, the uncertainty about the plaintext usually decreases, eventually reaching 0. At that point there is enough information to determine the plaintext uniquely and the cipher is breakable (at least in theory).

Most ciphers are **theoretically** breakable with only a few characters of ciphertext.

Theoretical Secrecy

This does not necessarily mean that the ciphers are insecure: the computational requirements needed to determine the plaintext may exceed available resources.

The important question is not whether a cipher is **unconditionally secure** but whether it is **computationally (practically) secure**.

Next Week

1. Rate of the Language
2. Redundancy
3. Equivocation
4. Perfect Secrecy
5. One-Time Pad
6. Symmetric Cipher Model
7. Kerckhoffs' Laws
8. Codes and Ciphers

- "Cryptography and Data Security" by D. Denning [2]

References

1. W. Stallings. “Cryptography and Network Security”, 7th Edition, Pearson Education Australia, 2017.

1. D. Denning. “Cryptography and Data Security”, Addison Wesley, 1982.