Information Theory and Mathematical Background

- Modern ***information theory*** was first developed by Claude Shannon in 1948. Shannon's work was mostly concerned with the transmission of "messages" in the presence of "noise".

- Information theory defines the amount of information in a message as the minimum number of bits required to encode all possible meanings of that message, assuming all messages are equally likely.

Basic Concepts in Probability

- A real **random variable** is defined as a real function of the elements of a **sample space**, **S**.

- The random variable is represented by an uppercase letter (X, Y, Z) and any particular value of the random variable by a lowers case letter (x, y, z).

- A Random Variable's set of values is the **Sample Space**, S. A Random Variable is a set of **possible values** from a random experiment.

- The space S is called the **certain event**, and its **elements** are called **experimental outcomes**. The **subsets** of S are called **events**. For a coin tossing experiment, the space S, consists of outcomes Heads (h), and Tails (t):

  S = {h, t}

- We can assign a random variable X to experiment, which will result in two possible events, either a Heads or Tails. We can **assign values to X**, such as **Heads=0** and **Tails=1** and we have a Random Variable "X" with a set of values:

  X = {0, 1}

- Note that the value we assign to X is arbitrary; it could have been 10 and 100.

- A random variable X is a variable which has certain values with certain probabilities. So, for our experiment above, we have the following probabilities of occurrence:

  $P(X = h) = 1/2$
  $P(X = t) = 1/2$

- Thus, P(h) + P(t) = 1

- Now let us consider an experiment of tossing a coin three times. The outcomes generate the following space (all possible outcomes):

  S = {hhh, hht, hth, htt, thh, tht, tth, ttt}

- Now, our random variable X will have the following probability of the event of getting three heads:

  P(*X* = hhh) = 1/8

- Similarly, the probability of the event {heads at the first two tosses} is found as:

  P(*X* = hhh, hht) = 2/8 = 1/4

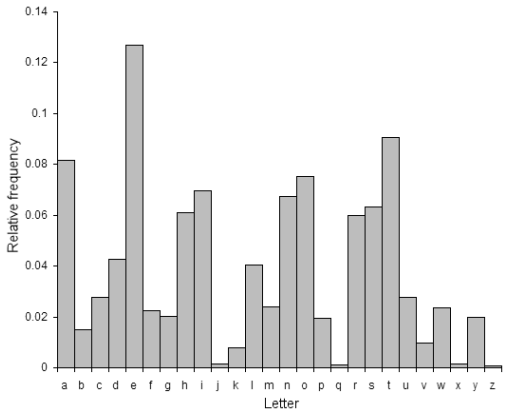
- To find the joint probability of two independent events that occur in sequence, find the probability of each event occurring separately, and then multiply the probabilities:

  P(X and Y) = P(X) * P(Y)

- For example, consider an experiment where a coin is tossed and a single 6-sided die is rolled. We can find the probability of landing on the head side of the coin (X), and rolling a 3 on the die (Y). Since the two events, X and Y, are **independent**, the probability of both occurring is:

  P(*X* = h, Y=3) = P(h) * P(3) = 1/2 * 1/6= 1/ 12

- Applying this to cryptanalysis, consider the following graph, which illustrates the relative distributions of the 26 letters in an average English text:



- Then the space S = {a, b, …..z} represents the English alphabet; let **X be the random variable representing letters in English text**, then:

  P(X = a) = 0.082, P(X = e) = 0.127, P(*X* = z) = 0.001

- Summarizing what we have done so far, let *X* and *Y* be random variables:
    - P(X = x) is the probability that X takes the value x
    - P(*Y* = *y*) is the probability that Y takes the value y.
    - The joint probability is defined as P(X = x,Y = y) is the probability that X takes the value x and Y takes the value y. X and Y are independent iff P(X = x,Y = y) = P(X = x) * P(Y = y) for all values of x and y.

- The "**Conditional Probability**" of X given Y, P(X|Y) means "Event X **given** Event Y", or in other words, given that event Y has already occurred, what is the probability of event X occurring? We define is as follows:

    P (X and Y) = P(X) * P(Y|X)
    P (X and Y) = P(Y) * P(X|Y)

- We can also state this as the probability that X takes the value x given that Y takes the value y. This gives us the following two relationships:

    P (X = x, Y = y) = P(X=x) * P(Y=y|X=x)
    P (X = x, Y = y) = P(Y=y) * P(X=x|Y=y)

- **Example**: Assume we will draw two cards from a deck of 52 cards:

  Event X is drawing a king first, and **event Y is drawing a king a second time**.

  For the first card the chance of drawing a King is 4 out of 52 (there are 4 Kings in a deck of 52 cards):

  P(X) = 4/52

  Since one king has been removed, the probability of the 2nd card drawn is **less** likely to be a King (only 3 of the 51 cards left are Kings), thus:

  P(Y|X) = 3/51

  and so:

  P(X and Y) = P(X) * P(Y|X) = (4/52) * (3/51) = 12/2652 = **1/221**

  the probability of getting 2 Kings is 1 in 221, or about 0.5%

- Using the above results, we can develop a very important theorem in probability known as **Bayes Theorem**.

    - We start with: **P(X and Y) = P(X) * P(Y|X)**
    - Use the algebraic property of swapping sides: **P(X) * P(Y|X) = P(X and Y)**
    - Divide by P(X): **P(Y|X) = P(X and Y) / P(X)**

- This gives us **Bayes Theorem**:

$$P(Y|X) = \frac{P(X\ and\ Y)}{P(X)}$$

- In terms of random variables, as long as P(Y=y) > 0, we can write:

$$P(X = x|Y = y) = \frac{P(X = x, Y = y)}{P(Y = y)}$$

- The above simply states that the probability of **event X given event Y** equals the probability of **event X and event Y** divided by the probability of **event Y**

- Note that X and Y are independent iff P(X=x|Y=y) = P(X=x), i.e., the value of X does not depend on the value of Y.

<u>Probability and Ciphers</u>

- We will know apply the concepts above to cryptography:

    - Let,
        M denote the set of all the possible plaintexts
        K denote the set of all possible key values
        C denote the set of all possible ciphertexts

    - Now M, K, C are random variables with the following probabilities with an assumption that M and K are independent):

        P(M=m); P(K=k); P(C=c)

- Then the set of ciphertext values generated using the key value k, is defined as:

$$\boldsymbol{C(k) = \{e_k(m): m \in M\}}$$

- The flowing relationship gives us the probability of a ciphertext, given the summation (over the valid key space) of the probabilities of the key values and the decrypted ciphertext (i.e., the plaintext):

$$\boldsymbol{P(C = c) = \sum_{\{k: c \in C(k)\}} P(K = k) \cdot P(M = d_k(c))} \qquad (1)$$

- Now, for $c \in C \; and \; m \in M$, we can compute the probability that **c** is the **ciphertext** given that **m** is the **plaintext** as follows:

$$\boldsymbol{P(C = c|M = m) = \sum_{\{k:m=d_k(c)\}} P(K = k)} \qquad (2)$$

- To break a cipher we need to know the probabilities of the plaintext given a certain ciphertext. We can compute the probability of m being the plaintext, given c is the ciphertext as follows:

$$P(M = m|C = c) = \frac{P(C = c|M = m) \cdot P(M = m)}{P(C = c)}$$

Substituting the above relationships we finally have (3):

$$P(M = m|C = c) = \frac{\sum_{\{k:m=d_k(c)\}} P(K = k) \cdot P(M = m)}{\sum_{\{k:c\in C(k)\}} P(K = k) \cdot P(M = d_k(c))} \qquad (3)$$

- This result can be used to compute the probability of **m** being the **plaintext**, given **c** is the **ciphertext**, as long as the **probability distributions** of K, M and the **encryption function** are known.

Example

Consider a system that has a set of two plaintext messages with the following probabilities:

$P(M=a) = 1/4$; $P(M=b) = 3/4$

There are three possible keys with the following probabilities:

$P(K=k_1) = 1/2$; $P(K=k_2) = 1/4$; $P(K=k_3) = 1/4$

The ciphertext code space is C = {1, 2, 3, 4}, defined by the following function table:

$e_k(m)$	a	b
k_1	1	2
k_2	2	3
k_3	3	4

Given the probability distributions for M and K we can compute the probability distribution of C (using 2) as follows:

P(C=1) = 1/8
P(C=2) = 7/16
P(C=3) = 1/4
P(C=4) = 3/16

Then, using the relationship for conditional probability derived above (3), we compute the conditional probability for the plaintext $m \in M$ given that a specific ciphertext was observed:

P(M=a|C=1) = 1
P(M=a|C=2) = 1/7
P(M=a|C=3) = 1/4
P(M=a|C=4) = 0

P(M=b|C=1) = 0
P(M=b|C=2) = 6/7
P(M=b|C=3) = 3/4
P(M=b|C=4) = 1

Thus we observe that if:

the **ciphertext = 1**, we **know** (certainty) that the **plaintext** is "**a"**
the **ciphertext = 2**, we **guess** (probably) that the **plaintext** is "**b"**
the **ciphertext = 3**, we **guess** (probably) that the **plaintext** is "**b"**
the **ciphertext = 4**, we **know** (certainty) that the **plaintext** is "**b"**

- The example makes it evident that given the above cryptosystem, one can determine the plaintext information by just observing the ciphertext, with a very high probability.

- This also suggests a serious weakness in the cipher implementation. So in the previous example the ciphertext reveals a lot of information about the plaintext.

- A strong cipher system must not reveal any plaintext information from the ciphertext. A system with this property is said to be **perfectly secure**.

- A cryptosystem has perfect secrecy iff:

  P(M=m|C=c) = P(M=m)

  for all $m \in M$ and $c \in C$

- That is, the probability that the plaintext is **m** given that the ciphertext **c** is observed is the same as the probability that the plaintext is **m** without seeing **c**. In other words **knowing c reveals no information about m**.

- Shannon's Theorem for perfect security can be summarized as follows:

  - Let (**M, C, K, E, D**) be an encryption scheme with: **|M| = |C| = |K | < ∞**
  - P(M=m) > 0 for all plaintext units m
  - The encryption scheme is perfect secret iff the probability distribution on K is the equal distribution (i.e., each key is chosen with probability 1/|K |).
  - For each plaintext unit **m** and each ciphertext unit **c** there is one and only one encryption key **k** with $\mathbf{E_k(m) = c}$.

- Shannon theorized that perfect secrecy is only possible if the number of possible keys is at least as large as the number of possible messages.

- Stated another way, the key must be at least as large as the message itself, and no key can be used twice (hence the reason for one-time pads providing perfect secrecy).

Entropy and Uncertainty

- Shannon defined a mathematical quantity called entropy which measures the **amount of information** in a message, if the message is one of a collection of possible messages, each being equally likely.

- For example the day-of-the-week field in a database contains 3 bits of information, encoded as follows:

  Sunday - 000
  Monday - 001
  Tuesday - 010
  Wednesday - 011
  Thursday - 100
  Friday - 101
  Saturday - 111

- Note that this information can also be represented using 7-bit ASCII characters, but it would simply take up more memory space and would not contain any more information than the 3-bit encoding.

- This **entropy** is the **average number of bits** needed to represent each possible message, using the best possible encoding of the message. If there are **n** messages **M = {m_1, ... , m_n}**, with probabilities of occurrence:

  $P(m_1)$, ... , $P(m_n)$ (with sum equal 1), then the **entropy H(M)** of this set of messages is:

  $H(M) = P(m_1) \log_2(1/P(m_1)) + ... + P(m_n) \log_2(1/P(m_n)) = \sum_{i=1}^{n} P(m_i) \log_2(\frac{1}{P(m_i)})$

- Intuitively, the entropy is just the **weighted average of the number of bits** required to represent each message, where the weights are the probabilities that each message might occur.

- For example, if we have two messages **M = {male, female}**, each having probability **1/2**, then the entropy is

  $H(M) = (1/2)(\log_2(1/(1/2)) + (1/2)(\log_2(1/(1/2)) = 1/2 + 1/2 = 1.$

- Thus in this case, as we would intuitively expect, there is one bit of information in such a message.

- Suppose **$P(m_1) = 1$** and the remaining probabilities are zero. In this case the entropy works out to be **0** as one would expect, since one is only going to receive the message **m_1**, so there is no information and no surprise in receiving this message (it is completely predictable).

- The actual message **m_1** might be complex, with many bits representing it, but its probability is **1**, so only this message can occur, with no information or "surprise" on its receipt, even if it is complex.

- As another example, suppose **n = 3** and **$P(m_1) = 1/2$, $P(m_2) = 1/4$,** and **$P(m_3) = 1/4$**. Then the entropy works out to be **1.5.** It is possible to encode these messages as follows:

  m_1: 0, m_2: 10, and m_3: 11

- In this case the average code length is the same as the entropy.

- Finally, suppose there are **1000** equally probably messages. Then the entropy is:

  $$H(M) = (1/1000)(\log_2(1/(1/1000)) + \ldots + (1/1000)(\log_2(1/(1/1000))$$
  $$= (1/1000)\log_2(1000) + \ldots + (1/1000)\log_2(1000)$$
  $$= 1000\ (1/1000)\ \log_2(1000)$$
  $$= \log_2(1000) = 9.965784285.$$

- Thus the entropy value of these messages means that there are nearly **10** bits of information in each message. Similarly, if there are **n** equally likely messages, then the entropy of a message is **$\log_2 n$**.

- A random message has the most information (the greatest entropy). Conversely, **loss of entropy is gain of information**.

- The entropy of a cryptosystem is a measure of the size of the keyspace, **K**. It is approximated using the following relationship (**K is number of keys**):

  $$H(K) = \log_2 K$$

- Entropy is a measure of the randomness or unpredictability of a string of bits in the ciphertext. Generally speaking, the greater the entropy, the more difficult it is to break the cipher.

- To better understand the implications of entropy, let us go back to our earlier example where we had the following probability distributions:

  P(M=a) = 1/4; P(M=b) = 3/4
  P(K=k_1) = 1/2; P(K=k_2) = 1/4; P(K=k_3) = ¼
  C = {1, 2, 3, 4}:

    P(C=1) = 1/8
    P(C=2) = 7/16
    P(C=3) = 1/4
    P(C=4) = 3/16

- Then, using the relationship for computing entropy:

  H(X) = $\sum_{i=1}^{n} P(x_i)\, log_2(\frac{1}{P(x_i)})$ **; where n is the maximum number of messages**

- We obtain the following:

  H(M) = 0.81; H(K) = 1.5; H(C) = 1.85

- Observe that the uncertainty or entropy **H(C)** of the **ciphertext** is smaller than the sum of the entropies of the plaintext **H(M)** and the key **H(K).**

- The significance of this is that the difference is the remaining **uncertainty** about the key given the ciphertext.

- This is the number of plaintext bits needed to be recovered when the message is scrambled in the ciphertext in order to learn the plaintext. See the elaboration below.

- Let **M**, **K**, and **C** be the set of possible messages, keys and ciphertext with associated random variables P,K,C. Then the following conditions are valid:

- If H(M|K, C) = 0:
  - Given the ciphertext and the key, you know the plaintext since it is the decryption of the given ciphertext under the given key.

- If H(C|M, K) = 0
  - Given the plaintext and the key, you know the ciphertext since it is the encryption of the given plaintext under the given key.

- The conditional entropy **H(K|C)** is called the **key equivocation** and measures the average uncertainty remaining about the key when a ciphertext has been observed.

- Suppose that a cryptanalyst wants to determine the key of a non-perfect cipher. The smaller **H(K|C)** is, the easier it will be to recover the key.

- The information revealed about the key by the ciphertext is the loss of uncertainty about the key when a ciphertext has been observed, defined as follows:

  H(K|C) = H(K) − H(K|C)

- For a cryptosystem (M, C, K, e_k (m), d_k (c)) we have:

  H(K|C) = H(K) + H(M) − H(C)

- The above expression simply states that the remaining uncertainty about the key when a ciphertext has been observed is equal to the sum of the uncertainties about the key and the plaintext minus the uncertainty about the ciphertext.

- Combining the two equations above, we obtain and expression which gives us the information revealed about the key by the ciphertext:

  H(K) − H(K|C) = H(K) + H(M) − H(C)

  And,

  H(K|C) = H(C) − H(M)

- **Remember, the smaller H(K|C) is, the easier it will be to recover the key.**

- Returning back to our example, we have:

  H(M) = 0.81; H(K) = 1.5; H(C) = 1.85

- We can then determine:

  H(K|C) = H(K) + H(M) − H(C) = 1.5 + 0.81 – 1.85 = 0.46

- Thus we conclude that the remaining uncertainty about the key is less than half a bit.

- Also, the information revealed about the key by the ciphertext is determined as:

  H(K|C) = H(C) − H(M) = 1.85 − 0.81 = 1.04.

- Thus we conclude that the ciphertext leaks more than 1 bit of information about the key. Stated another way, the cryptanalyst has to learn only one well-chosen bit to recover the message.

Rate of a Language

- For any given language, the rate (or entropy) of the language is determined as follows:

  r = H(M)/N

  where, N is the length of the message, M.

- More precisely, for a language, L, it is defined as:

$$H_L = \lim_{n\to\infty}\left(\frac{H(M^n)}{n}\right)$$

- This is difficult to compute exactly but it can be approximated experimentally to yield the following empirical result:

$$1 \leq H_L \leq 1.5$$

- Thus the rate (or entropy) of normal English language is estimated between **1 bit/letter** to **1.5 bits/letter.**

- Shannon's work shows that this entropy depends on the length of the text. Specifically, for **8-letter blocks is 2.3 bits/letter**, and for **16-letter blocks it is 1.3 -1.5 bits/letter**. Generally we use 1.3 bits/letter.

- The absolute rate of a language is the maximum number of bits that can be coded in each character, assuming equal probability for each character.

- This is also referred to as the **maximum entropy** of the individual characters. For an L-character language, this is given as:

  $R = \log_2 L$

- For the English language this is:

  R = log2 (26) = 4.7 or 5 bits/character

- Stated another way, we will need 5 bits to represent each character. If on the other hand we use a compression scheme such as Huffman for example, we can reduce that to **1.5 bits/character**.

Redundancy

- For any language **L** with entropy $\mathbf{H_L}$ and alphabet **M**, we need about ($\mathbf{n log_2 \#M}$) (#M is the total number of messages or characters in the language) bits to represent a string of length **n**.

- However a compact encoding only needs about $\mathbf{nH_L}$ bits. The redundancy $\mathbf{R_L}$ of a language is defined as the relative difference between both encodings:

$$R_L = \left(\frac{nlog_2\#M - nH_L}{nlog_2\#M}\right) = \left(1 - \frac{H_L}{log_2\#M}\right)$$

- If we assume $H_L = 1.25$ then the redundancy of English is:

$$R_L = \left(1 - \frac{1.25}{log_2 26}\right) = 0.73$$

- This means that we can compress an English text file of 10 MB down to 2.7 MB.

- Cryptanalysts use the natural redundancy of a language to reduce the number of possible plaintexts. The more redundant the language, the easier it is to cryptanalyze.

- This is the main reason that many cryptographic implementations use a compression algorithm to reduce the size of the plaintext before encrypting it. This also reduces the computing effort required for the encryption and decryption processes.

Unicity Distance

- The unicity distance $\mathbf{n_o}$ of a cryptosystem is the value of **n** at which the expected number of keys becomes zero.

- Alternatively, we can say that it is the average amount of ciphertext required for an adversary to be able to uniquely determine the key, given enough computing resources.

- Corollary: for a perfectly secure cipher, $n_o = \infty$

- Unicity is approximated (it is a probabilistic measure) as follows:

$$n_o \approx \frac{log_2\#K}{R_L log_2\#M}$$

- As an example, consider a simple substitution cipher (Caesar cipher) which has:

  #M = 26
  #K = 26!
  R_L = 0.73 for English (see previous calculation for R_L)

  Thus,

$$n_o \approx \frac{log_2 26!}{0.73 log_2 26} = 25.7$$

- So we require on average only 25 ciphertext characters before we can break the substitution cipher.

Complexity Theory

- Complexity theory provides a methodology for analyzing the computational complexity of various cryptographic techniques and algorithms.

- The **"Big O"** notation is a way of expressing the computational difficulty of a certain computation or algorithm. It is not necessarily a measure of how much time the calculation would take, but rather how efficient it is in terms of the number of calculations that will be required.

- Computational complexity is usually measured using two variables, **T** (**time complexity)**, and **S** (**space** or **memory** requirements).

- Big O Notation is a measure of how well the performance of an algorithm scales as the amount of data to be processed increases.

- Big O notation generally contains two numbers, **n**, which is the **size of the dataset**, and another symbol such as an exponent, log, or coefficient multiplied with n. In this way, the result of the O[n] expression is how many calculations must be performed based on the input data size n.

- It is simply the term of a complexity function that grows the fastest as n (number of operations) gets larger, and all lower terms are ignored.

- For example, if the time complexity of a function is: $4n^2 + 7n + 12$, then the complexity is on the order of n^2, and expressed as O[n^2].

- An algorithm is constant if its complexity is independent of n: O[1]. A linear algorithm is O[n].

- The following are examples of Big O Notation:

  O[1]
  O[n]
  O[n^2]
  O[log(n)]
  O[n log(n)]
  O[n!]

- The following table illustrates the run times for different algorithm classes in which **n = 10^6**:

Class	Complexity	# of Operations for n = 10^6	Time at 10^6 O/S
Constant	O[1]	1	$1\mu sec$
Linear	O[n]	10^6	1 sec
Quadratic	O[n^2]	10^{12}	11.6 days
Cubic	O[n^3]	10^{18}	32,000 years
Exponential	O[n^n]	10^{301030}	10^{301006} times the age of the universe

- The above assumes that the unit of “time” for our computer is a microsecond (or the period, T, is 1 usec).

- As an example consider **quadratic** case:

$$Time = \frac{10^{12}/10^{6}}{86.4^{3} sec/day} = 11.6\ days$$

Modular Arithmetic

- All modern cryptography is based on number theory and modular arithmetic. Modular arithmetic is also known as “clock arithmetic”.

- For example, 1300 hrs means 2:00 pm; this is modulo 12 arithmetic (i.e., the modulus is 12). The notation is as follows:

  14 mod 12 = 2

- To do modular arithmetic, we simply do normal arithmetic, divide the result by the chosen modulus (2, 12, 360 etc.) and take the remainder.

- Other examples are when we do modulo-12 arithmetic when converting time from 24 hr format, and modulo-360 arithmetic when calculating angles.

- Alternatively we can do normal arithmetic and, if necessary, keep subtracting or adding the modulus until the result is positive and is less than the modulus.

- For example, in modulo-12 arithmetic, 2300 hrs, we have 23 mod 12 equals 11.

- Another way of writing this is to say that 23 and 11 are equivalent, mod 12:

  $23 \equiv 11 \pmod{12}$

- A very good (and ancient) example of modular arithmetic in encryption is the **Caesar Cipher**, in which each letter of a message is encrypted by replacing it with the letter three places further along in the alphabet.

- So “foo” would be encrypted as “irr”. In order make this process more conducive to software implementation, we could represent each letter by a number between 0 and 25 (note that starting from 0 is very helpful when it comes to modular arithmetic).

- The encryption key would be denoted +3, and the operation of encrypting a letter is then to add the key to the letter’s corresponding number.

- For example, to encrypt the letter f:

  E(M) = C = M + 3 = 5 + 3 = 8 => i

- Now, consider what happens when encoding **x, y and z**:

  C = 23 + 3 = 26 mod 26 = 0 => a
  C = 24 + 3 = 27 mod 26 = 1 => b
  C = 25 + 3 = 28 mod 26 = 2 => c

- Decryption in the Caesar cipher is performed by subtracting 3, and if the answer is negative, we add 26 to the remainder to arrive at the plaintext symbol.

- Thus, for the previous example the decryption process is as follows (**D(M) = C – 3**):

  M = 8 – 3 = 5 => f
  M = 0 – 3 = -3 + 26 = 23 => x
  M = 1 – 3 = -2 + 26 = 24 => y
  M = 2 – 3 = -1 + 26 = 25 => z

Prime Numbers

- Prime number theory is a singularly important component in securing information in the modern world.

- The mathematics of prime numbers has consumed and challenged mathematicians for centuries, with no idea that their work would one day have practical applications, such as securing information in a public domain such as the Internet.

- A positive integer that is greater than 1 is considered prime if its only divisors are 1 and itself. In other words, no other number evenly divides it. For example the following are all prime numbers:

  $$2, 73, 2521, 2365347734339, 2^{756839}-1$$

- It is a daunting task to count prime numbers, and to determine if any given large number is a prime, and to generate prime numbers. Nobody as yet has a simple formula that will generate all the prime numbers.

- In fact, Euclid may have been the first to prove that there are an infinite number of primes.

- Primes are the building blocks of all numbers. Every number greater than 1 can be written as the product of prime numbers, and the Fundamental Theorem of Arithmetic tells us that this prime factorization of a given number is unique, meaning there is not another set of primes that can be multiplied together to get the same result.

- Consider the highest common factor (hcf), or greatest common divisor, of two numbers. For example, the hcf of 8 and 12 is 4; no higher number will divide exactly into the two numbers.

- Euclid's algorithm enables us to calculate the hcf quite easily - you simply take remainders when dividing one number by the other, retaining the two smallest numbers each time, and stopping just before you get to 0. For example:

  12 / 8; remainder = 4
  8 / 4; remainder = 0 => answer is 4

- Two numbers are relatively prime if their hcf is 1 - so they have no prime factors in common.

- A cornerstone of modern public-key encryption is the supposition that it is enormously difficult to find the prime factors of very large numbers, meaning that the computing power required is not available to us today.

- Note that it is not absolute size or difficulty that matters, but how fast the difficulty increases as the size of the numbers increases.

- Some calculations can be performed in polynomial time. This means that the computation time can be expressed as a polynomial function (a sum of finite powers) of the "size" of the problem (typically the number of digits in the input).

- Equations that show a polynomial time complexity have variables in the bases of their terms. Examples: $n^3 + 2n^2 + 1$. Notice n is in the base, not the exponent.

- Other very difficult calculations are thought not to be solvable in polynomial time, even if the polynomial involves huge powers. These are exponential-time problems.

- In exponential equations, the variable is in the exponent. For example: 2^n.
  As mentioned earlier, exponential time grows much faster. If n is equal to 1000 (a reasonable input for an algorithm), then 1000^3 is 1 billion (polynomial time), and 2^{1000} (exponential time) is huge!

- Consider the fact that there are about 2^{80} hydrogen atoms in the sun, much more than 1 billion.

- For some sizes of numbers, they may be easier than some polynomial-time problems, but eventually as the size of the problem increases (the numbers become "very large") the exponential-time problem will become slower than the polynomial-time problem.

- To put a practical context to all this, consider how information such as your credit card number can be sent over the Internet, which is an inherently insecure medium. In the old days, to send a secret message, you first had to send a key - secretly - to the recipient, either in person, or through a trusted third party.

- Moving to modern times, most of us understand the fact that we cannot use the internet to send the key, then our financial information.

- This problem was solved in 1978 by Rivest, Shamir, and Adelman. Their method, now known as RSA, depends on some wonderful (and vast) properties of prime numbers.

- One of these is that it is rather easy to generate large prime numbers, but much harder to factor large numbers into primes.

- Another is Fermat's Little Theorem, which was used to implement the RSA algorithm.

- RSA uses a lookup method for translating strings of letters and other symbols into blocks of numbers and back again. The following table can be used as an example:

	0	1	2	3	4	5	6	7	8	9
0	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX
1	SP	A	B	C	D	E	F	G	H	I
2	J	K	L	M	N	O	P	Q	R	S
3	T	U	V	W	X	Y	Z	a	b	c
4	d	e	f	g	h	i	j	k	l	m
5	n	o	p	q	r	s	t	u	v	w
6	x	y	z	.	,	:	;	'	"	`
7	!	@	#	$	%	^	&	*	-	+
8	(	)	[	]	{	}	?	/	<	>
9	0	1	2	3	4	5	6	7	8	9

- We can do this using a table like the one on the right, where **A** corresponds to 11, **B** to 12, etc. Thus the string **foo&bar!** becomes **42427612115470**. Thus, the process of encrypting and decrypting becomes a matter of computing with large integers.

- Now, let **p** and **q** be large prime numbers and let **N = pq**.

- Let **e** be a positive integer which has no factor in common with **(p-1)(q-1)**.

- Let **d** be a positive integer such that **ed - 1** is divisible by **(p-1)(q-1)**.

- Let,

  $E(x) = x^e \bmod N$; where E(x) is the encryption process
  $D(x) = x^d \bmod N$; where D(x) is the decryption process

- Now let us consider makes RSA so hard to break. In keeping with tradition we will use Alice and Bob as examples.

- Alice, the person who designs the code, first generates the large primes **p** and **q**, then she chooses **e**. Finally she solves an equation to find **d**:

  de + (p-1)(q-1)y = 1, where all the letters are integers. (**y = E(x)**)

- Alice makes **e** and **N** public. This is all anyone needs to send secret messages to her.

- Now let us assume that Bob wants to decrypt messages sent to Alice. In order to do so, first he must factor **N** to find **p** and **q** so as to set up the equation.

- Then he solves the equation to find **d**. At this point he can decrypt Alice's messages.

- The problem is that the factoring problem takes huge amounts of computer time - for large enough values of **p** and **q** it would take literally millions of years, given current mathematical theory and current computer technology, to crack the code.

- As a further illustration of the magnitude of the problem, consider the message

```
2436895187740522149300895060 33
9985963357828798391070516253 60
7140448055114932771201027350325,
3239156661331877771746337433 07
6657414951585135873876216674 42
8451506590312184584172482223 6676
```

- This was encrypted using the following key :

```
e = 5,
N = 519208104502047440191322024032
    461128846299254256408973265508
    515449982559682356973314555442 57
```

- The number **N** is the product of two primes, just as **184033** is the product of the primes **73** and **2521**. If you could discover the primes, you could decrypt the message. But this is not so easy to do, and hence one of the reasons why RSA works.