Spring 5-2000

A Brief History of Cryptography

William August Kotas
University of Tennessee - Knoxville

Appendix D - **UNIVERSITY HONORS PROGRAM**
SENIOR PROJECT - APPROVAL

Name: William August Kotas

College: Arts & Science Department: Computer Science

Faculty Mentor: Dr. Michael D. Vose

PROJECT TITLE: A Brief History of Cryptography

I have reviewed this completed senior honors thesis with this student and certify that it is a project commensurate with honors level undergraduate research in this field.

Signed: [illegible], Faculty Mentor

Date: 5/8/00

Comments (Optional):

A BRIEF HISTORY OF CRYPTOGRAPHY

Prepared by William A. Kotas

For

Honors Students at the University of Tennessee

May 5, 2000

ABSTRACT

This paper presents an abbreviated history of cryptography. The paper begins with an introduction that defines cryptography and establishes the context and purpose of the report. The second section focuses on early ciphers, such as the Caesar cipher and Vigenere cipher. This is followed by a discussion of ciphers used during the two World Wars, including Enigma and the Navajo code talkers. The last section examines computerized ciphers such as Lucifer and RSA. The report ends with concluding remarks.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

GLOSSARY

cipher – a process that scrambles the letters of a message.

code – a process that scrambles the words of a message.

ciphertext – the message that results after encryption.

cryptography – the science of rendering a message unintelligible to any but sender and intended recipient, according to a specific protocol.

cryptographer – one who specializes in enciphering messages.

cryptanalyst – one who specializes in breaking codes and ciphers.

decryption – undoing the scrambling process to render the ciphertext legible.

encryption – using a cipher to scramble the meaning of a message.

plaintext – the message to be encrypted

steganography – the process of concealing the existance of a message.

A BRIEF HISTORY OF CRYPTOGRAPHY

Prepared by William A. Kotas

For

Honors Students at the University of Tennessee

May 5, 2000

INTRODUCTION

The modern usage of the word cryptography evokes images of spies and espionage, of secret messages sent back and forth between agent and government, and of secret panels in briefcases to conceal stolen documents. For those who lurk in the dark corners of the Internet, the word "encrypted" represents an unbreakable wall that prevents tampering. The science of cryptography is all of these things, but few of those who banter the word about know the actual definition of cryptography. Cryptography is the science of rendering a message unintelligible to any but sender and intended recipient, according to a specific protocol. The process of concealing a secret message through cryptography is called encryption. This does not include the process of hiding of the existence of a message, which is known as steganography. The Greeks used steganography to transmit important information without their enemies knowing. The historian Herodotus wrote of an exiled Greek Demeratus who wrote a message on a pair of wooden tablets about the intention of the Persian King Xerxes to attack Greece. He covered these tablets with wax, so they would appear blank to any Persian guards who may encounter them. The Greeks received the tablets, scraped off the wax, and prepared to meet Xerxes at the Bay of Salamis, where they won a great victory. Cryptography and steganography have been used over the centuries to conceal secret messages, yet it is cryptography that offers the better security. If the secret message is intercepted, steganography does not prevent the finder from reading the message, while an encrypted message is still unreadable by the "enemy". While independent, encrypting and hiding secret messages does add more security (Singh 5 – 6).

The purpose of this report is to provide the reader with a brief history of cryptography, the more powerful branch of secret writing. This report will first discuss early ciphers such as the Caesar cipher, then it will discuss the advent of polyalphabetic ciphers, the mechanization of encryption with machines such as ENIGMA, the development of public key encryption, and a look at the future of cryptography.

CODES AND CIPHERS

The two main requirements of any cryptographic undertaking are the method and the key. The method is the general procedure of encryption, such as exchanging one letter for another, or scrambling the existing letters around. The key defines the specifics of the method. The security of a particular cryptographic method depends on the number of possible keys a method can generate. If the number of keys is large, it becomes impractical for an enemy to test all possible keys in an attempt to decrypt the message.

THE BRANCHES OF CRYPTOGRAPHY

Cryptography can be broken into two branches: transposition and substitution. Transposition involves a scrambling of the order of the plaintext. Substitution involves exchanging one set of symbols for another, rendering the resulting text unreadable. If a transposition or substitution scrambles words it is called a code, and if it scrambles letters it is called a cipher. Transposition ciphers change the position of a letter while retaining its identity. An example of a simple transposition cipher is the two-line "rail fence" cipher. Take a short message, such as "Meet in the courtyard at five". Then write each letter of the message on alternating lines, as shown below:

```
M e i t e o r y r a f v
 e t nh c u t a d t i e
```

Finally, the text is rewritten, reading each line from right to left, with the encrypted message being "Meiteoryrafv etnhcutadtie" (Gardner 11 – 12). Transposition ciphers seem to offer a high level of security. As the number of letters grows, the number of letter combinations also grows. With ten distinct letters, there are 3, 628, 800 different ways of arranging them. Most messages of importance will be longer than ten letters. The flaw in this is that the transposition must be done in a straightforward way so the receiver can quickly unscramble the letters, as can anyone else who knows the key. Anyone who figures out the key will be able to easily decipher the messages.

EARLY CIPHERS

One of the earliest ciphers to be used on a large scale was a transposition cipher known as the scytale cipher, used by the Spartans in the fifth century B.C. The scytale was a wooden staff, around which a strip of leather or parchment was wound. The message was written on the strip and the strip was unwound, scrambling the letters. To recover the message the strip would have to be wound around a scytale of the same diameter of the one used to encrypt the original message (Smith 16). Julius Caesar used ciphers extensively, but we only have detailed records of one of these, in which Caesar would write down the letter three places down from the actual letter of the message. This type of cipher alphabet became known as the Caesar shift cipher. Originally applied only to the actual cipher used by Julius Caesar, it now represents a family of ciphers where the cipher alphabet is shifted a fixed number of places down from the plain alphabet. As an example, the English alphabet is given, with a three-place Caesar shift alphabet underneath:

ABCDEFGHIJKLMNOPQRSTUVWXYZ
DEFGHIJKLMNOPQRSTUVWXYZABC

A Caesar shift can be between one and twenty-five, thus providing twenty-five different keys. This type of cipher does not offer very much security. However, if we allow any permutation of letters to be used as a cipher alphabet, the number rises to over 10^{25} different possible keys. The keys are also simple arrangements of the twenty-six letters of the alphabet. By sacrificing just a few of the possible keys, the generation of the keys could be made even easier by selecting a code word to form the first part of the cipher alphabet, with the remaining letters following the key phrase, in their normal order. Thus a cipher using the word MILK as the keyword would look like this:

ABCDEFGHIJKLMNOPQRSTUVWXYZ
MILKNOPQRSTUVWXYZABCDEFGHJ

This type of cipher is called a monoalphabetic substitution cipher. This cipher was considered unbreakable for a millennium.

THE ADVENT OF FREQUENCY ANALYSIS

While the European world wallowed in its Dark Ages, the Arab world was reaching a pinnacle of intellectual achievement. The fields of mathematics and linguistics were very advanced, mainly for the purpose of discovering the chronological order of the revelations in the Koran. As a side effect, Arab scholars discovered that the letters of the Arabic alphabet have different frequencies of usage. For example, the letter 'L' is used, on average, ten times as much as the letter 'J'. These frequencies can be exploited to determine the cipher alphabet used to encrypt a message by matching frequencies of letters in the cipher text with the frequencies of the letters in the alphabet of the language that the plain text was written. This procedure, known as frequency analysis, was first documented by the scholar Abu Yusuf Ya'qub ibn Is-haq ibn as-Sabbah ibn 'omran ibn Ismail al-Kindi in the ninth century AD. It is possible for a skilled cryptanalyst to decipher a message in minutes if the message is more than a line or two, and the language of the plain text is known. This technique does not automatically make ciphered messages simple exercises in letter replacement. Messages that have a high percentage of normally seldom used letters would be extremely difficult to decipher if the frequencies are applied mechanically (Singh 10 – 20). It still requires intuition and guile to be a cryptanalyst. This technique has been a primary tool of cryptanalysts for hundreds of years, but it would take a Renaissance to bring this technique to the Western world.

Ciphers in Europe

During the Dark Ages, cryptography was rarely practiced. Only monasteries encouraged the study of cryptography, mainly for the purpose of deciphering encrypted messages found in the Bible. These monks developed cryptography into a workable science, and gradually reintroduced it to European society. By the fifteenth century, it was a part of the daily politics of Europe, particularly in Italy. Italy during the early Renaissance was full of independent city-states. Diplomacy and a need for secure communication required that all ambassadors have cipher secretaries and every nation have a cipher office. The technique of frequency analysis made its way from the Middle East to Europe, and the once unbreakable codes began to be broken at an alarming rate. Those ignorant of the power of frequency analysis had faith in monoalphabetic substitution ciphers, while their secure communications were transparent to cryptanalysts (Singh 26 – 28). The Spanish cipher office was one such case. When they discovered the French cryptanalyst Francois Viete had been breaking their ciphers for years, they thought him in league with dark powers. King Philip II petitioned the Vatican to try him before a Cardinal's Court. The Pope rejected the petition (Smith 22). The search was on for a way to confound this new codebreaking tool.

A Better Cipher?

In an effort to increase the security of the monoalphabetic substitution cipher, a few simple improvements were added. The first of these was the introduction of nulls, or symbols in the cipher text that would have no meaning in the plain text. The intended recipient, who has the key, would know to ignore these symbols, but an enemy trying to decipher the message could not ignore them. Another technique used was to misspell the plain text before encryption, so that the message "this is a secret" would be written as "thys es uh sikret." It was hoped that these methods would change the frequencies of the symbols used in the cipher text, and defeat the frequency analysis. Nomenclatures, or a cipher alphabet with a few distinct code words, were also used to confound the cryptanalysts of the day. Unfortunately for the cryptographers most skilled cryptanalysts could get around these innovations and secret messages were not very secret (Singh 29 – 32).

VIGENERE'S CIPHER

In the mid-sixteenth century, a French diplomat turned scholar named Blaise de Vigenere came up with the first truly innovative new cipher since the advent of the Caesar cipher. By carefully examining earlier cryptographic works and making some educated leaps of intuition, Vigenere developed a powerful new means of encryption. Named for its creator, the Vigenere cipher makes use of twenty-six distinct cipher alphabets to encrypt a message. The cipher alphabets are all simple Caesar shifts of one to twenty-six arranged in a square, as shown in Figure 1. Each letter of the plain text message is encrypted using a different row of the Vigenere Square as the cipher alphabet. To choose which rows will be used, a keyword is chosen. This word is then written above each line to be encrypted, and repeated until all the letters of the line have a corresponding letter in the code word. Using the code word SMILE, the results look something like this:

```
smi  lesmil esmi  le  sm iles
The attack will   be at  dawn
```

Then each letter of the message is encrypted using the row of the Vigenere Square that begins with the letter of the keyword above the plaintext letter. With this key word, five cipher alphabets are used to encrypt a message. With longer key words and key phrases, more alphabets can be used. The Vigenere cipher completely confounds frequency analysis because each letter of the cipher text can represent several different letters in the plain text. The number of keys is also very large, because the number of possible key words and phrases is very large. Blaise de Vigenere published an account of this cipher in 1586, but the world of cryptography did not embrace this new cipher until almost two hundred years later (Singh 46 – 51).

Figure 1. A Vigenere Square.

ABCDEFGHIJKLMNOPQRSTUVWXYZ
BCDEFGHIJKLMNOPQRSTUVWXYZA
CDEFGHIJKLMNOPQRSTUVWXYZAB
DEFGHIJKLMNOPQRSTUVWXYZABC
EFGHIJKLMNOPQRSTUVWXYZABCD
FGHIJKLMNOPQRSTUVWXYZABCDE
GHIJKLMNOPQRSTUVWXYZABCDEF
HIJKLMNOPQRSTUVWXYZABCDEFG
IJKLMNOPQRSTUVWXYZABCDEFGH
JKLMNOPQRSTUVWXYZABCDEFGHI
KLMNOPQRSTUVWXYZABCDEFGHIJ
LMNOPQRSTUVWXYZABCDEFGHIJK
MNOPQRSTUVWXYZABCDEFGHIJKL
NOPQRSTUVWXYZABCDEFGHIJKLM
OPQRSTUVWXYZABCDEFGHIJKLMN
PQRSTUVWXYZABCDEFGHIJKLMNO
QRSTUVWXYZABCDEFGHIJKLMNOP
RSTUVWXYZABCDEFGHIJKLMNOPQ
STUVWXYZABCDEFGHIJKLMNOPQR
TUVWXYZABCDEFGHIJKLMNOPQRS
UVWXYZABCDEFGHIJKLMNOPQRST
VWXYZABCDEFGHIJKLMNOPQRSTU
WXYZABCDEFGHIJKLMNOPQRSTUV
XYZABCDEFGHIJKLMNOPQRSTUVW
YZABCDEFGHIJKLMNOPQRSTUVWX
ZABCDEFGHIJKLMNOPQRSTUVWXY

Intermediate Efforts

The Vigenere cipher was impregnable to conventional cryptanalysis because of its complexity. Ciphers that use multiple cipher alphabets are called polyalphabetic substitution ciphers. They are much more secure, and much more complex. This makes them difficult and time consuming to use. This discouraged many people from using the Vigenere cipher and others of its kind. The simpler monoalphabetic substitution ciphers are fast and easy to use, making them perfect for concealing one's diary from a spouse and other civilian applications. Military and political cryptographers needed something more secure, but also required something that was fast so that urgent communications could be encrypted with little delay. Thus the quest to improve the monoalphabetic cipher continued. One method was the homophone substitution cipher. This cipher used multiple symbols to represent the same letter, with the number of symbols used proportional to the frequency of the letter. In ciphers for English the letter 'e' would have the most symbols that could represent it, while the letter 'z' would have only one. Using this method the frequencies of the symbols would all be approximately the same, defeating frequency analysis. Other permutations of the monoalphabetic substitution cipher arose; the greatest of which was the Great Cipher of Louis XIV, developed by Antoine and Bonaventure Rossignol in the seventeenth century. This cipher was used to conceal all sorts of important French documents, and resisted the attempts of cryptanalysts to break it for two hundred years. Instead of using a homophone substitution to encrypt letters, the Great Cipher used it to encrypt syllables. These variations were much more secure than straightforward monoalphabetic substitution, and there was no need to adopt the bulky Vigenere cipher.

The Black Chambers

By the eighteenth century, these enhanced ciphers were beginning to fall prey to skilled teams of cryptanalysts who pooled their resources for the various governments of the times in secret cryptographic offices known as the Black Chambers. Letters that passed between embassies and their respective home countries were often first routed through these Black Chambers, where they were opened, copied, resealed, and sent on their way. The copies would then be turned over to the cryptanalysts, who would puzzle out the meanings of the messages, and each Chamber's respective government would use the information. The greatest of these was the Viennese Black Chamber, which sold much of the information it collected to other governments. These exploits horrified the cryptographers of the day, and eventually they were forced to adopt the Vigenere cipher to insure security. Another push to insure security of messages was the advent of the telegraph. To send a message by telegraph, one must provide the telegraph operator with a copy of the message. It was possible that a company's operators could be bribed to reveal the contents of these messages, which was a breach of security. The solution was to encrypt a message before giving it to the operator, who would be unable to read it but could transmit it across the telegraph lines. The Vigenere cipher offered perfect security in all respects, and was considered unbreakable (Singh 52 – 63).

The Fall of the Vigenere Cipher

The Vigenere cipher was the bastion of cryptography. The Black Chamber of London, feared as it was, could not crack it. It would take a genius to discover the hidden weakness of the Vigenere cipher, and that genius was Charles Babbage. Charles Babbage was a nineteenth century scientist and eccentric who is most famous for his 'difference engines', the forerunners of today's modern computers, but he was also well known in London circles as an exceptional cryptanalyst. In 1854 he became embroiled in an argument about a cipher developed by John Hall Brook Thwaites, a dentist who claimed his cipher was new, when in fact it was merely a repeat of the Vigenere cipher. When Babbage pointed this out to Thwaites in correspondence, Thwaites challenged Babbage to break his cipher. Babbage set to work to find a weakness in the Vigenere cipher. Babbage noticed that some sequences of letters would repeat themselves after a set number of letters. He hypothesized that perhaps these were common words encoded using the same sequence of cipher alphabets. This proved to be the key to cracking the Vigenere cipher. By studying the numbers of letters between repeated sequences of letters, and performing a little statistical magic, Babbage was able to divine the length of the keyword. Babbage then broke the cipher text down into several parts, each one corresponding to a message encrypted by a cipher alphabet defined by a letter of the keyword. These alphabets are Caesar shift alphabets and using frequency analysis to find the alphabets, Babbage could figure out the letters of the key word. Once the keyword was found, it was a simple matter to decode Thwaites' cipher. This would have been wonderful news for the cryptanalyitic community if Babbage had ever published his work. He did not, and Friedrich Wilhelm Kasiski, who published his results in 1863, independently discovered his technique. This technique has hereafter been known as the Kasiski Test, and Babbage's contribution was unknown until twentieth century scholars examined his personal notes (Singh 67 – 78). Even so, these developments proved that the unbreakable cipher was indeed breakable thanks to human ingenuity.

CIPHERS IN THE PUBLIC EYE

From the nineteenth century to the dawn of the twentieth, public interest in ciphers grew. The popularity of telegraph service showed the need to protect sensitive personal correspondence from the prying eyes of the telegraph operator. This cost more money, because it would take a telegraph operator longer to send jumbles of letters than words and phrases, but it was better than having gossip spread by nosy operators. The public embraced this way of hiding information and began to show off their skills. Victorian lovers would write long encrypted passages and publish them in the personal columns of the local newspapers. These columns would come to be known as 'agony columns'. Cryptographers would challenge their colleagues with blocks of cipher text in the newspapers. This familiarity with cryptography found its way into the literature of the time. Author Jules Verne's novel *Journey to the Center of the Earth* begins with the decipherment of a parchment. Edger Allen Poe wrote a story called *The Gold Bug* that contains an example of frequency analysis. Other cryptographic puzzles were devised during this period. In 1885 a small pamphlet was published that detailed the story of Thomas J. Beale. This pamphlet outlined a tale concerning one of Beale's trips out west, and his discovery of a gold mine. Upon his return he and his companions decided it would be better to conceal the gold so that they could go and increase their wealth. The second time they returned Beale entrusted some curious instructions with Robert Morris. Beale disappeared and never returned to Lynchburg, where Morris lived. After many years Morris opened the box and found an account of Beale's adventures. The box also contained three sheets detailing the location of the treasure. The problem is that the sheets are encrypted. The author of the pamphlet, a friend of Morris' had deciphered the second sheet of Beale's ciphered instructions and revealed the amount of treasure, which in today's money is worth about twenty million dollars. Beale's second page was encrypted with a book cipher. A book cipher uses a text as a key for the encryption. The words of the key text are numbered, and the first letter of each word can be associated with the number of the word. The result is similar to a homophone cipher, but with many, many more symbols for each letter. The key text for the second sheet was the Declaration of Independence. The other two sheets have never been deciphered (Singh 79 – 98).

The Vigenere cipher had been broken. With a few notable exceptions, it was up to the cryptographers to regain the upper hand in the never-ending struggle between those who conceal and those who reveal. The next big breakthrough for cryptographers would come during the twentieth century, and the fate of the world would rest upon defeating it.

CIPHERS AT WAR

During the First World War, cryptography was struggling to defeat the efforts of cryptanalysts on both sides. Many new ciphers were developed, but they were permutations or combinations of older ciphers that had already been broken. Their security could be measured in days, as cryptographers would break them almost as fast as they were developed. The main problem facing cryptographers was the volume of material to be decrypted. Radio had made communication fast, easy, and wireless. This ease of use dramatically increased the number of messages sent and intercepted each day. Several intelligence techniques were developed to provide information about messages without actually decrypting them. One of these was traffic analysis. By studying where messages come from and who sent them, it was possible to trace particular units of troops as they moved. This information could be used to guess their objectives. The Allied cryptographers were unparalleled during World War I. The Germans entered the war with little cryptographic expertise, and did not develop a cryptography department until two years into the war. The contribution of cryptographers during this time can best be shown by the story of the Zimmerman telegram.

ZIMMERMAN'S FOLLY

Arthur Zimmerman was appointed Germany's Foreign Minister in 1916. President Woodrow Wilson hoped this would bring about a peaceful resolution to the War since the

United States had taken a neutral stance. This was not to be the case. In January 1917 the Supreme High Command was trying to convince the Kaiser to forgo his promise that German U-boats would surface before attacking to insure that a civilian ship would not be sunk. The Command had determined that such an act would enable German U-boats to tighten their blockade of Britain and bring about swift German victory. The sinking of US civilian ships would occur, and draw the US into the war. The Kaiser decided that victory would come before the US could mobilize its forces, and approved the action.

Zimmerman devised a way to insure the US would not enter the war, even if victory did not come as rapidly as promised. He proposed an alliance with Mexico and to give the Mexican President encouragement to invade the US to reclaim Texas and other territories. He also suggested that Mexico encourage Japan to attack the US at the same time. The resulting multi-front invasion would distract American troops and guarantee no US involvement in the European arena. Zimmerman sent a telegram to the German ambassador in Washington outlining his plan with instructions to send a copy to the German ambassador in Mexico, who would in turn give it to the Mexican president. He encrypted this message, because he knew the Allies were intercepting all diplomatic communications. The British cryptographers were presented with the message the day it was sent, and by the next day had puzzled out its message. However, it was not presented to President Wilson until after its Mexican equivalent had also been intercepted. This was to fool the Germans into believing there was a break in security at the Mexican end of the transmission route, and not that their cipher had been broken. After reading the

Zimmerman telegram and Zimmerman's confession of its authenticity, President Wilson brought the United States into the War to End all Wars (Singh 104 – 115).

THE UNBREAKABLE CIPHER - AGAIN

Charles Babbage was able to break the Vigenere cipher because the key to the Vigenere cipher was made up of one word repeated over and over again. Longer keys as long as the message itself were also tried, and failed to provide additional security. Most of these message length keys were lists of things, or lyrics to songs. In each case, with a few deft leaps of intuition, a skilled cryptanalyst could piece together the key because he was able to find meaningful words that would suggest the next step. Major Joseph Mauborgne introduced the concept of the random key for use in ciphering messages using a Vigenere square. Mauborgne's system involved producing two pads of randomly generated keys, one for the sender and one for the receiver. The message is encrypted with the first key in the pad. The receiver uses that same key to decrypt the message and both sender and receiver destroy their keys. This cipher is known as the one-time pad cipher. It is absolutely secure. Babbage's method of decryption relies on repetition within the key to crack the cipher. With the one-time pad cipher, there is no repetition in the key. The keys are random strings of letters and contain no meaningful words to guide a shrewd cryptanalyst in the right directions to solve the key. The number of keys is enormous as well. These factors combine to make a truly unbreakable cipher.

Theoretically, the one-time pad cipher was perfect. In practice, there are more problems associated with encryption than the security of the cipher. So much so that the one-time pad cipher is hardly ever used. Generating truly random sequences of letters is a difficult undertaking. Unless one uses a truly random event as a basis for generation, random sequences will not be truly random sequences. Humans who try to generate random sequences will invariably leave patterns in their sequences that cryptanalysts can exploit for decryption. Harnessing natural processes for generating random numbers is slow compared to the number of random sequences a radio operator might need as keys during a war. Distribution of all these random keys is also a large problem. Everyone has to be using the same key for particular messages or encrypted communications become meaningless. Enemy interception of a cipher pad would render all further communications transparent. With all these problems, Mauborgne's one-time pad cipher was impractical for modern battlefield use (Singh 120 –123). Only the most vital communications sent by people with enormous resources could be encrypted properly in this fashion. The search continued for a better cipher.

ENIGMA

Mechanical aids to encryption have been around since Sparta's heroes used the scytale. In the fifteenth century, Leon Alberti, who helped develop the Vigenere cipher, placed two disks with the alphabet carved on them together. The inner disk could be rotated so that the letters of the inner and outer alphabets could be matched up in various ways, generating the different Caesar ciphers. By changing the alphabet for each letter, one encrypts a message using the Vigenere cipher. This is a cipher disc, and it has been used in various forms for hundreds of years. These devices merely made the encryption easier to accomplish, but the methods were still the standard ciphers of the day. In 1918, a machine was developed than not only aided encryption, but also was the basis for a whole series of secure ciphers.

Frustration and Success

Arthur Scherbius was a German inventor and businessman looking for new business opportunities. A personal project of his was to correct the inadequacies of World War One's cryptographic methods. In 1918, Scherbius took out a patent on what he called his Enigma machine. There are several parts to the Enigma machine that make up its complex cipher. The first portion is the scrambler. The scrambler was a rubber disc with wires embedded inside. This scrambler was connected to a keyboard and a series of lamps that would light up in response to a letter pressed. The scrambler took an electric signal from the keyboard and transferred the signal to the lampboard, causing a letter to light up. By placing the wires along different paths, the scrambler created a monoalphabetic cipher. In the Enigma plans, this scrambler rotated one twenty-sixth of a revolution between each keystroke. The scrambler and rotation created a polyalphabetic cipher with twenty-six cipher alphabets. Scherbius used three standard scramblers in his Enigma machine, with the convention that the second scrambler would rotate one twenty-sixth of a full turn after the first scrambler completed one full rotation. The third scrambler would only rotate one twenty-sixth after the second scrambler completed one full rotation. This brought the number of cipher alphabets to 17, 576. Deciding that his machine needed more security, Scherbius added a reflector that would send a signal through the reflectors a second time and made the three scramblers interchangeable. He also added a plugboard. The plugboard would switch the signals of two letters if a special cable connected the two plugs on the plugboard. By itself, this only produces a simple monosubstitution cipher, but combined with the other security measures, the total number of keys rises to over 10^{15}. The Enigma machine cipher was resistant to all the techniques used to crack ciphers at this time, and it was easy to use and generated encrypted messages very quickly. Scherbius thought his creation would be impossible to crack, and began to market his machine to both military and business alike. Absolute security did not come cheap. Each Enigma unit would cost thousands of dollars in today's money. Many businesses shied away from this hefty price tag. The German military was also not interested, confident that their cryptographers were doing their job well. Other inventors were developing similar machines and ran into similar frustrations.

Revelation and Revolution

In 1923, two documents were published that would change the course of Scherbius' Enigma business from failing to thriving. Winston Churchill's *The World Crisis* and the British Royal Navy's official history of the First World War both stated that Allied cryptanalysis had cracked German codes and provided valuable information to the Allied war effort. The German military sought to avoid these mistakes a second time, and in 1925 the Enigma was in mass production, to enter service in 1926. By the start of the

Second World War, 30,000 Enigma machines were in use by the Nazi forces, providing them with absolute security for their communications. The race was on to crack the Enigma cipher, a race to be won by an unexpected participant.

At the end of the First World War, Poland was in a terrible predicament. Sandwiched between an ambitious Russia and a desperate Germany, an independent Poland had to be vigilant if it wanted to retain its freedom. The Biuro Szyfrow was the most successful cipher bureau during the time between World War One and World War Two. Its baptism of fire came during the Russo-Polish War of 1919-1920, and it continued to monitor German communications as well. In 1926, the Biuro Szyfrow began receiving Enigma messages, and German communications became indecipherable. The Biuro Szyfrow worked tirelessly to find a way to break Enigma's cipher. The French Secret Service provided much needed information regarding the internal workings of the military version of Enigma by providing documents about the operation of the machine to the Biuro. The French cipher bureau had given up breaking the Enigma cipher, and cheerfully handed over all their collected information. Using these documents it was possible to construct a replica of the Enigma machine. An unexpected bonus that came with this information was how the Germans provided a key for Enigma's use. German Enigma operators would receive a new codebook every month that listed a key for every day. These keys were made up of plugboard arrangements, scrambler orientations, and scrambler settings. These day keys would be used to encode a message key that was the same as the day key except for the scrambler settings. Then the text would be enciphered using the message key. Deciphering these keys seemed next to impossible, but the Biuro was undaunted. They recruited several mathematicians to become cryptanalysts in the hopes that a scientific mind would be more apt at cracking a mechanical cipher. The star of the group was Marian Rejewski.

A Chink in the Armor

The Germans followed a strict procedure to encrypt their messages. First the operator would pick a random scrambler arrangement to serve as the message key. This key would be encrypted twice using the day key, and then the message would be encrypted using the message key. This repetition of the message key was to prevent miscommunication by radio interference or human error. It would prove to be the weak spot in Enigma's security.

Marion Rejewski was given the task of cracking Enigma. Having studied mathematics for a career in insurance, his logical mind attacked the problem by focusing on the repetition of the message key. With some intuitive leaps, Rejewski discovered that he could form a table of letter relationships for a day key, similar to a cipher alphabet. Each day key had a different table of relationships. Rejewski began to look for patterns in the tables, and finally settled on letter chains. Rejewski formed these chains by picking a letter and then alternating between the lines of a relationship table until he returned to the starting letter, as in Figure 2. With another leap of intuition, Rejewski concluded that the number of links in the chains was only determined by the scrambler settings. The total number of day keys for Enigma is roughly 10^{15}, but the number of scrambler settings is only 105, 456. This is a considerably simpler problem to solve. Rejewski and his team

Figure 2. Chains of Letters

Source: Singh 151 - 152

Table of Relationships

1st letter:	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
4th letter:	F	Q	H	P	L	W	O	G	B	M	V	R	X	U	Y	C	Z	I	T	N	J	E	A	S	D	K

This table reveals the following letter chains:

$A \Rightarrow F \Rightarrow W \Rightarrow A$	3 links
$B \Rightarrow Q \Rightarrow Z \Rightarrow K \Rightarrow V \Rightarrow E \Rightarrow L \Rightarrow R \Rightarrow I \Rightarrow B$	9 links
$C \Rightarrow H \Rightarrow G \Rightarrow O \Rightarrow Y \Rightarrow D \Rightarrow P \Rightarrow C$	7 links
$J \Rightarrow M \Rightarrow X \Rightarrow S \Rightarrow T \Rightarrow N \Rightarrow U \Rightarrow J$	7 links

proceeded to check each scrambler setting and catalogue the chain lengths determined by each one. After a year their catalogue was complete. Rejewski could now take a series of messages, find the patterns of chain length, and then look up a matching pattern in his catalogue to find the day key's scrambler settings. Now all that remained was to find the plugboard settings. This was accomplished by removing all plugs from the Enigma machine replica, setting the scramblers according to the deciphered settings, and typing in some cipher text. Most of what was returned was gibberish, but recognizable phrases would appear occasionally, with just a couple of letters mismatched. These mismatches would reveal a particular plugboard setting, and with enough phrases the entirety of the plugboard settings would be revealed. Rejewski would then have the complete day key, and have no trouble deciphering messages. The Enigma cipher was broken, but Rejewski did not sit idle after his triumph. When the Germans modified their message transmission protocols, Rejewski automated his cataloguing process into a machine he called a bombe. This machine would rapidly check the 17,576 settings for a particular scrambler arrangement. With six of them working together, the bombes were able to reveal a day key in approximately two hours.

At the end of 1938, the Germans introduced some new procedures for their Enigma operators. Two new scramblers were added to the original three, and the number of plugboard settings was raised from six to ten. These new additions to the day key increased the number of possible keys to 1.59×10^{20}. Worse, the original catalogues and bombes were also rendered useless. The Biuro knew the Enigma cipher could be broken, but the task would require sixty bombes instead of six, even if the internal wiring of the new scramblers were known. The Biuro had neither the resources nor the time to crack this new Enigma cipher. Hitler had introduced the concept of blitzkrieg, and the German military was gearing up for the invasion of Poland. The Biuro was determined that the Allies know about their work, and in the weeks before the invasion transferred replicas of the Enigma machine and blueprints for the bombes, along with an explanation of Rejewski's technique.

Turing's Marvelous Machine

The British cipher bureau, Room 40, had given up any hope that the Enigma cipher could be broken. With the revelation of the Polish breakthrough, Room 40 regained hope. Armed with a new resolve and new techniques, the cryptanalysts of Room 40 embarked on a struggle that would play a vital role in the Allied war effort. To begin, they gathered up scientists and mathematicians to add to the compliment of linguists, and moved to Bletchley Park to form the Government Code and Cipher School (GC&CS). This new organization mastered Rejewski's techniques and went on to discover other shortcuts to finding Enigma keys (Singh 124 – 162). The Germans frequently updated their machine, so new shortcuts were always needed. The Bletchley Park members were dreading the day that the Germans changed their transmission of the message key from twice to once before encrypting the message, because all of their techniques exploited this concept. The task of figuring out how to solve this problem was given over to Alan Turing.

Alan Turing was a mathematician at Cambridge before being recruited to work at Bletchley Park. Though only twenty-six in 1937, Turing published a paper called 'On Computable Numbers' which would form the basis for all modern computing theory. In 1939 he began work on a method to crack Enigma without relying on the repeated message key. Turing looked for structure in the vast library of decrypted messages at his disposal. He began to see that sometimes the contents of an enciphered message could be predicted, based on the time of interception and the source of the message. These bits of plaintext that can be associated with ciphertext are called cribs. These cribs could be used to find patterns in the relationships between letters similar to the letter chains Rejewski used in his work. Many of the breakthroughs following this discovery are beyond the scope of this paper, but these achievements allowed Turing to construct a machine that could find an Enigma key in under an hour under perfect conditions. These were also called bombes, in honor of the achievements of the Biuro (Singh 165 – 177). Using these bombes, and their top-notch team of cryptanalysts, the GC&CS provided valuable wartime intelligence to the Allies. They had cracked another 'unbreakable cipher'. The GC&CS also decrypted Japanese and Italian messages, providing information for the African and Pacific efforts. Unfortunately, the marvelous breakthroughs in cryptanalysis made at Bletchley Park would not be revealed until 1974, when Captain F. W. Winterbotham published *The Ultra Secret*, a book about the successes of the GC&CS (Singh 188). It should be noted that the true weakness of the mechanical cipher is not the machine, but in the way the machine is used. If Enigma operators had not used repetition in their message keys, sufficiently randomized the day keys, and avoided messages that resulted in easily identifiable cribs, their ciphers would have been much more difficult to break, maybe even impossible. There is one important drawback to mechanical ciphers, and that flaw is that they take time to use properly. It takes a fair amount of time to type each letter of a message into the cipher machine and write down the enciphered letter before the message can be given to a radio operator to transmit. Deciphering a message requires a like amount of time. Under heavy combat conditions, this type of encryption is too slow to be practical. The American forces of the allies were faced with just such a problem in the Pacific Theater of World War Two. Their solution created a code that remains unbroken to this day.

NAVAJO CODE TALKERS

Before 1942, U.S. forces in the Pacific were letting valuable tactical information fall into Japanese hands because they had no rapid means of encryption and decryption for their communications during combat conditions. In such cases, radio operators resorted to using slang English, sprinkled with as many expletives as possible. Many soldiers have resorted to this during battle, as related by William E. Kotas, a squad leader during the Vietnam War. Sgt. Kotas states that "We were given code books every month or so, but some months the codes didn't come, and we resorted to our own hack code, called the motherfuck code. A radio operator would say 'Motherfuck, motherfuck' to indicate he was using the code, and then say the message using that code. It changed every now and then, and was easy to break, but it gave us some security during a firefight (Kotas)."

In 1942, a man named Philip Johnston did not meet the age limit for military service but still wanted to contribute to the war effort. He began to develop an encryption system that could be used rapidly while in the field, and he was inspired by his youth spent on a Native American reservation, specifically with the Navajo tribe. The Navajo language is incomprehensible to almost anyone who is not closely affiliated with the tribe. Johnston thought if each radio unit used Navajo tribesmen to translate messages into the Navajo language, it would be an unbreakable code and provide secure communications. Johnston took his idea to Lieutenant Colonel James E. Jones, and after a small demonstration to senior marine officers, began the task of recruiting and training Navajo men and boys. The tribal council fully supported these efforts, and tribe members were so eager to join that they lied about their ages and gorged to meet military requirements. Ultimately, four hundred and twenty Navajos would join the ranks of the U.S. Armed Forces as code talkers. There were some problems with Johnston's code that had to be overcome before the Navajos could be used in real combat. Many of these came from the fact that the Navajo language did not have terms for most technical and military jargon. Johnston and other Navajos developed a code that assigned common Navajo words and phrases to the most common of these terms. For example, military planes were given the names of birds. Names and places were spelled out, using Navajo words in place of letters. The Navajo recruits memorized this code so that codebooks would not be needed. During the first months of real combat, the Navajos were pitted against the portable encryption machines the military used. The machines could encrypt and decrypt in an hour or two, while the Navajos could do the same task in less than five minutes. Demand for these specialists increased. The Navajo code was improved, adding more common terms and alternate words for each letter. This new code removed the few flaws in the original, and the Japanese intelligence was never able to break it, giving the U.S. forces the upper hand in the war of communication (Singh 193 – 201). The Navajo code is one of the few that remains unbroken today.

After the Second World War, cryptographers and cryptanalysts began to use a new device to aid them in their ongoing struggle to make and break ciphers. This tool was the computer. The first computers were room-sized machines that were costly to build and maintain, so that only government agencies would have the resources to keep one of them around. As computers became smaller, faster, and less expensive, they became more common in the workplace. They would also give rise to some potent new ideas in cryptography.

CIPHERS AND COMPUTERS

Computer ciphers are essentially the same as the mechanical ciphers embodied by Enigma, with three major differences. First, a computer is not limited by what can be practically built, and can emulate a mechanical cipher machine that would be far to expensive and complicated to actually construct. Second, a computer can encrypt and decrypt long messages very quickly, much faster than an Enigma machine. Third, a computer only encrypts binary numbers. Using conversion codes such as the ASCII, text messages are converted into binary numbers before encryption, and converted back to text after decryption. Despite these differences, computers still use forms of transposition and substitution to perform encryption and decryption.

LUCIFER

In the 1960's, when computer technology was becoming more powerful and less expensive, businesses began to use computerized encryption to protect important messages, such as money transfers. As computer encryption spread, problems with standardization began to arise. In May of 1973, America's National Bureau of Standards began accepting proposals for a standard encryption system. A prime candidate for the standard was an IBM product known as Lucifer. Horst Feistel, a German immigrant, developed the Lucifer cipher at IBM's Thomas J. Watson Laboratory in the early 1970's (Singh 245 – 249). The details of the Lucifer cipher are as follows:

> "First, the message is translated into a long string of binary digits. Second, the string is split into blocks of 64 digits, and encryption is performed separately on each of the blocks. Third, focusing on just one block, the 64 digits are shuffled, and then split into two half-blocks of 32, labeled $Left^0$ and $Right^0$. The digits in $Right^0$ are then put through a 'mangler function', which changes the digits according to a complex substitution. The mangled $Right^0$ is then added to $Left^0$ to create a new half-block of 32 digits called $Right^1$. The original $Right^0$ is relabeled $Left^1$. This set of operations is called a 'round'. … This process is repeated until there have been 16 rounds in total. … The exact details of the mangler function can change, and are determined by a key… (Singh 249)"

Lucifer was lauded as one of the strongest commercial ciphers available. So strong that the National Security Agency thought there was a chance they would not be able to break the encryption. Thus, the NSA lobbied to have Lucifer weakened by reducing the number of possible keys to approximately 10^{16}. This is known as 56-bit encryption because 56 is the number of bits required to represent a number as large as 10^{16}. This limitation made Lucifer impossible to break by civilian businesses and their limited access to powerful computers, but still allow the NSA to crack it with their tremendous amounts of computing power. In November of 1976, the 56-bit Lucifer cipher was adopted as the United States' official encryption standard and renamed Data Encryption Standard (DES). DES is still the official standard (Singh 250).

With DES, the problem of standardization was solved, but there was still one major hurdle to overcome for businesses: key distribution. DES users must still agree on a key for messages and all recipients must have a copy of that key. For international business communications, this can be very troublesome because the only truly secure way to deliver a key is in person. Couriers were employed by many businesses to deliver keys to their networks of contacts, but as the number of contacts grew, the overhead associated with key distribution grew exponentially. The problem of how to get around distributing keys was deemed impossible to solve, and indeed had remained unsolved since ciphers were originally developed. The answer to the solution would be the next great cryptographic step.

ALICE, BOB, AND EVE

The reason key distribution is such a difficult constraint to resolve is that the sender and receiver must agree on a key before any encrypted messages can be sent back and forth. The key is what allows the recipient to reverse the encryption algorithm and retrieve the text message. In September 1974, three mathematicians, Whitfield Diffie, Martin Hellman, and Ralph Merkle began to explore ways to get around the key exchange problem. After several failed ideas, they began to explore one-way functions. One-way functions are easy to do, but very difficult to reverse. Modular arithmetic is an area of mathematics that has many one-way functions, so Diffie, Hellman, and Merkle concentrated their efforts in that area. Modular arithmetic has many of the same operations as normal arithmetic (addition, subtraction, etc.), but adds an extra step to find the answer. First, one picks a number. This number is the modulus of the operation. Then, one performs the operation as normal on two arguments. Finally, you divide the answer of the operation by the modulus and note the remainder. The remainder is the answer of the modulus function. For example, suppose we choose the modulus to be 7. The operation we want to perform is addition (mod 7), and the arguments are 4 and 13. First, we perform the addition: $4 + 13 = 18$. Then we perform the modulus: $18 / 7 = 2$ remainder 4. Thus, $4 + 13 = 4 \pmod 7$. Two years after starting the project, Hellman had a flash of insight. He envisioned the following hypothetical situation, involving Alice, Bob, and Eve; the imaginary people referred to when discussing cryptographic puzzles. Alice and Bob wish to exchange secret information. They agree on an encryption method that uses numbers as a key while talking over the phone. Eve, an 'enemy' of Alice and Bob, is listening in on their phone conversation. How is it possible for Alice and Bob to agree on a key using the phone if Eve is listening in? Hellman focused on a function of the form M^x mod N. Using his idea, Alice and Bob agree on values for M and N, so that M is less than N. Then Alice and Bob pick secret numbers, called A and B respectively. For this example, Alice and Bob have picked $M = 7$ and $N = 11$. Eve knows these numbers because they were transmitted over the phone. Alice's secret number A = 3, and Bob's secret number B = 6. The next step has Alice and Bob run their secret numbers through the one-way function. Alice ends up with the following results:

$$7^3 \bmod 11 = 343 \bmod 11 = 2 \text{ or } \alpha$$

Bob ends up with these results:

$$7^6 \bmod 11 = 117{,}649 \bmod 11 = 4 \text{ or } \beta$$

Alice and Bob exchange these numbers via phone, and Eve also knows them. Alice calculates the equation $\beta^A \bmod N$ and gets the following:

$$4^3 \bmod 11 = 64 \bmod 11 = 9$$

Bob calculates the equation $\alpha^B \bmod N$ and receives this result:

$$2^6 \bmod 11 = 64 \bmod 11 = 9$$

At this point, both Alice and Bob have the same number, and that number can now be used as a key. Eve, who has listened to every exchange knows the one-way function, α and β. Unfortunately, she does not know A and B. While it is possible to work out A and B from the information Eve knows, it is very difficult because Alice and Bob are using a one-way function. If the numbers are very large, the task of reversing the function becomes even more difficult. Thus, a key was agreed upon and Alice and Bob did not have to meet in person to exchange it. Hellman explained his discovery to his two partners, and in June of 1976 the Diffie-Hellman-Merkle key exchange scheme was demonstrated at the National Computer Conference (Singh 255 – 267). The problem of key distribution had been solved.

RSA

The Diffie-Hellman-Merkle key exchange scheme is a solution to the key distribution problem, but not the best solution. Alice and Bob must still communicate to establish a key before encryption can begin. This can be a hassle if Alice and Bob live on opposite sides of the world. One person is going to lose sleep waiting to agree on a key, or the key exchange will take place through a series of e-mails that arrive every 12 hours. E-mail is instantaneous, and these restrictions slow e-mail communications to a crawl, if security is desired. A more efficient solution was needed, one that would allow communications to occur at any time. Whitfield Diffie approached the key distribution problem from a different angle than Hellman. Though Diffie still concentrated on one-way functions, he began to look at the fact that all encryption techniques developed so far were symmetric, or that the same operation was used for both encryption and decryption, so the same key is used. With a flash of insight, Diffie came up with the idea of the asymmetric cipher. This cipher would require two keys, one for encryption and one for decryption. Using an asymmetric cipher would make things much easier for Alice and Bob. Alice could generate two keys, one for encryption and one for decryption. Alice keeps the key for decryption secret. This is also known as Alice's private key. She then publishes the encryption key publicly, and it is known as Alice's public key. If Bob wants to send a secure message to Alice, he does not need to contact her to do so. Bob just looks up Alice's public key, encrypts a message with it, and sends it to Alice. When Alice receives the message, she decrypts it using her private key and reads it. This is much more efficient than the Diffie-Hellman-Merkle key exchange scheme. Unfortunately, while Diffie had conceptually created a better solution to the key distribution problem, he did not have a specific asymmetric cipher algorithm in mind. In 1975, Diffie published an outline of this idea, and other researchers joined in the search for a practical, asymmetric function.

At the Massachusetts Institute of Technology's Laboratory for Computer Science, a trio of researchers began to explore ideas for an asymmetric function that met the requirements of Diffie's idea. Ron Rivest, Leonard Adleman, and Adi Shamir spent a year developing functions and discarding them when flaws emerged, until April 1977, when Rivest had a breakthrough. He worked through an entire night writing a paper describing his new function, and credited all three men with the discovery. Adleman tried to find a flaw in Rivest's work, but his only complaint was the listing of authors. After some deliberation, the authors were listed as Rivest, Shamir, and Adleman. This paper described the cipher that would eventually be named the RSA cipher (Singh 269 – 273).

The RSA cipher is not very difficult to use. Suppose Alice wants to use the RSA cipher to encrypt her personal messages. She would begin by choosing two prime numbers p and q. Then Alice would calculate $n = p \times q$ and $\phi(n) = (p - 1) \times (q - 1)$. She chooses a prime number d such that $d > p$, $d > q$, and the greatest common denominator of d and $\phi(n)$ is 1. Alice can now calculate the number $e = d^{-1} \bmod \phi(n)$. Alice's public key is made up of the numbers e and n, which she publishes. Bob now wants to send Alice a message. Bob finds Alice's public key. After converting the text message into a number, he calculates the cipher text $C = M^e \bmod n$. When Alice receives the message, she then converts the cipher text into plain text $M = C^d \bmod n$ (Vose). In practice, the message is broken up into blocks of binary numbers and each one encrypted or decrypted separately. For strong security p and q should be very large prime numbers, because it is those values which ultimately determine the public and secret keys. If Eve is attempting to find out Alice's secret key, she must factor the number n in Alice's public key. Finding prime factors is a very difficult problem, so for sufficiently large prime numbers, the problem would be impractical to solve. This is the strength of the RSA cipher.

Alternative History

Rivest, Shamir, and Adleman have been hailed as some of this century's greatest cryptographers for their development of the RSA cipher, but recent events have revealed that public-key encryption was developed earlier, by researchers working at the Government Communications Headquarters (GCHQ), the top secret British agency that succeeded Bletchley Park. James Ellis was working for GCHQ under an oath of secrecy when he was given the task of creating a solution for the key distribution problem in the early months of 1969. By the end of the year, Ellis had developed ideas very like those Diffie, Hellman, and Merkle would discover independently years later, and was stymied by the same problem. Ellis knew public-key encryption was feasible, but he could not find a function that would satisfy the requirements. Ellis was not a mathematician by trade, so he showed his discoveries to his bosses. Three years later, GCHQ's top minds were hard at work trying to discover a function that would make Ellis' ideas practical. In

September of 1973, Clifford Cocks, a specialist in number theory, was recruited to work for GCHQ and assigned a mentor named Nick Patterson to "show him the ropes." During the course of Cocks' introduction to the routine at GCHQ Patterson outlined Ellis' theory on public-key encryption. Not realizing the significance of this theory, Cocks' began to search for a solution and echoed many of the same insights that led Rivest, Shamir, and Adleman to their cipher (Singh 279 – 285). In his own words, Cocks' states: "From start to finish, it took me no longer than half an hour. I was quite pleased with myself. I thought 'Ooh that's nice. I've been given a problem and I've solved it.' (Singh 285)" Cocks reported this to his mentor, and Patterson reported it to his supervisors. Cocks showed his discovery to his close friend and fellow GCHQ employee Malcolm Williamson the following year. Williamson was skeptical, and spent the next day trying to find the flaw in Cocks' idea. He failed, but he came up with another solution, which would later be called the Diffie-Hellman-Merkle key exchange scheme. Ellis, Cocks, and Williamson were ahead of their time, but these discoveries would go unnoticed because their oaths of secrecy regarding their work. It was not until 1997, when Cocks was allowed to present a paper at the Institute of Mathematics and its Applications Conference that began with a history of his work at GCHQ, that Ellis, Cocks, and Williamson would be recognized for their achievements (Singh 285 – 292).

Another Zimmermann's Folly

Phil Zimmermann is a former anti-nuclear activist who has turned his attentions toward encouraging the use of encryption to secure privacy. Zimmermann believes that the proliferation of electronic communications has made it much easier for those communications to be monitored. Zimmermann states, "Unlike paper mail, e-mail messages are just too easy to intercept and scan for interesting keywords. This can be done easily, routinely, automatically, and undetectably on a grand scale. (Singh 295)" RSA seemed to be the perfect solution to this problem in 1977, but RSA encryption and decryption requires much more computing power than DES or other symmetric encryption methods. Thus only government, large businesses and the military had the computing power to use RSA encryption. Zimmermann decided everyone should have this level of security for their communications, and set about to create a product that would offer an easy to use encryption interface with the security of RSA for the personal computer. He called his project Pretty Good Privacy (PGP), and began working on it during the 1980's (Singh 298).

In 1991, PGP was almost ready for market. PGP encrypts a message using the following procedure. First, it encrypts a message using the symmetric IDEA cipher, which is similar to DES but provides more security. Second, the key used to encrypt the message is then encrypted with RSA using the intended recipient's public key. Finally, a digital signature is added to the message, and the message is sent to the recipient, who uses PGP to undo the encryption process and verify the digital signature. This procedure would be complicated, but the PGP program does all the work automatically. By only using the processor intensive RSA cipher to encrypt a small message (the IDEA key), and using the IDEA cipher to encrypt the actual message, Zimmermann was able to make PGP secure, but without the computing power costs associated with full RSA encryption

(Zimmermann 76 – 81). However, some concerns remained for Zimmermann before he could market PGP. Zimmermann would need a license to use RSA, because RSA is a patented product. Also, an anti-crime bill that would, among other things, require electronic communication equipment manufacturers to allow law enforcement officials to obtain plain text copies of all electronic communications was almost passed in the U.S. Senate. Zimmermann felt the public had a right to PGP before it was banned, and he felt that such a ban was only a matter of time. In June of 1991, Zimmermann had PGP posted to a Usenet newsgroup. PGP began to spread across the world, but now Zimmermann had other worries. In 1993, RSA Data Security, Inc. sued for patent infringement, and the FBI began prosecuting him for illegal weapons exportation (Singh 301 – 303). With PGP available on the Internet, everyone in the world has access to strong encryption. This includes organized crime, terrorist groups, drug dealers, and other criminal elements. For decades, law enforcement has used wiretaps and other surveillance techniques to monitor electronic communications. If these communications are encrypted strongly they are impossible to decipher, so intercepting them would be useless and make wiretaps ineffective. The Internet is also a worldwide communications network. Since PGP was released on the Internet, anyone in the world is able to download the program. PGP is a strong encryption program. The U. S. Government classifies all strong encryption software as munitions (Dam 415). In 1996, the FBI dropped all charges, and Zimmermann received his license after an out of court settlement, but his story touched off a heated debate about the privacy of the individual over the ability of law enforcement agencies to perform effective surveillance on suspected criminals (Singh 315). This debate is still heatedly discussed on web pages, newsgroups, and in Congress.

CONCLUSION

From Julius Caesar to Phil Zimmermann, cryptographers have struggled to keep messages secret and cryptanalysts have struggled to read secret messages. With the advent of public-key encryption and the RSA cipher, it would seem that the cryptographers have won the ancient war of words, but future discoveries may render the RSA cipher obsolete. The Vigenere cipher was thought to be unbreakable until Babbage discovered its weakness. The same is true for the Enigma cipher. With new advances in the field of quantum computing, it may be that RSA's days of absolute security are numbered. While modern computers operate on the classical laws of physics, quantum computers operate using quantum laws of physics that allow them to do billions of calculations simultaneously. This would make factoring large prime numbers trivial, thus making it easy to break the RSA cipher. Currently, researchers are unable to build a quantum computer, and may never be able to build one. Only time will reveal the true winner in the secret war of cryptography.

WORKS CITED

Dam, Kenneth W. and Herbert S. Lin, ed. Cryptography's Role in Securing the Information Society. Washington, D.C.: National Academy Press, 1996

Gardner, Martin. Codes, Ciphers, and Secret Writing. New York: Dover Publications, Inc., 1972.

Kotas, William E. Personal interview conducted by William A. Kotas. March 23, 2000.

Smith, Laurence Dwight. cryptography: the science of secret writing. New York: Dover Publications, Inc., 1943.

Singh, Simon. The Code Book. New York: Doubleday, 1999

Vose, Michael D. Lecture for Computer Science 494 at University of Tennessee, Knoxville. Sept. 28, 1999.

Zimmermann, Philip. The Official PGP User's Guide. Cambridge, MA: The MIT Press, 1995.

APPENDIX 1: README AND SOURCE CODE FOR A MODIFIED RSA IMPLEMENTATION

```
Program: rsa2.c
Description: This program will implement an encryption system based on RSA
           WARNING! This program is for demonstration purposes only, so it
           utilizes 32-bit keys.  For true security, longer keys
           should be chosen.

Notes: The random number generation and bigint codes were written by
      Dr. Michael Vose.  Dr. Vose also assisted with the input/output
      portion of the code.

Usage:
      Compile with: gcc -g -o mycrypt rsa2.c
      Run with:
              mycrypt pubkey seckey c /*Creates public and secret keys */
              mycrypt <file> pubkey e /*Encrypts <file> to cfile*/
              mycrypt <file> seckey d /*Decrypts <file> to output*/
```

```
#include <stdio.h>
#include <malloc.h>

typedef long          Int;
typedef unsigned long Unsigned;
typedef unsigned char uchar;
typedef Unsigned      digit;
typedef digit         *digits;

typedef struct tag {
  int    s;
  int    l;
  int    u;
  digits d;
} bigintStruct;

typedef bigintStruct *bigint;

bigint ZERO, ONE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT, NINE, TEN,
TENk, BILLION;
bigint Tmp, Tmq, Tmr;
bigint Tmp0, Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6, Tmp7, Tmp8, Tmp9, Tmp10
, Tmp11, Tmp12;
bigint I, F, R, convert[256];
Int    Equal, atoi[256], m13[256], p13[30030];
char   itoa[10], str[1233], ptr[1233];
int sqcheck;

#define S(x)       ((x)<<16)
#define M          (S(1)-1)
#define H(x)       ((x)>>16)
#define L(x)       ((x)&M)
#define Abs(n)     (((n)<0)? (-(n)): (n))
#define Max(x,y)   (((x)>(y))?  (x): (y))

char *print(bigint, char *);
#define show(s,x) printf("%s%s\n",s,print(x,str))

/**************************** arithmetic ******************************
*/

char *print(bigint,char *);

void error(char *m)
{
  printf("\n%s\n",m);
  exit(0);
}

bigint getVar(int n)
{
  int i;  bigint new = (bigint)malloc(sizeof(struct tag));

  new->s = 0;
  new->l = n;
  new->u = 0;
  new->d = (digits)malloc((n+1)*sizeof(digit));

  for(i = 0; i< n; new->d[i++] = 0);
  return new;
}
```

```
  bigint getDigit(int n)
{
  bigint new = (bigint)malloc(sizeof(struct tag));

  new->s    = (n>0)? 1: ((n<0)? -1: 0);
  new->l    = 1;
  new->u    = Abs(new->s);
  *(new->d = (digits)malloc(2*sizeof(digit))) = Abs(n);

  return new;
}

void init(Unsigned s)
{
  Unsigned h,i,j,k;

  initrand(s);

  convert[itoa[atoi['0'] = 0] = '0'] = ZERO  = getDigit(0);
  convert[itoa[atoi['1'] = 1] = '1'] = ONE   = getDigit(1);
  convert[itoa[atoi['2'] = 2] = '2'] = TWO   = getDigit(2);
  convert[itoa[atoi['3'] = 3] = '3'] = THREE = getDigit(3);
  convert[itoa[atoi['4'] = 4] = '4'] = FOUR  = getDigit(4);
  convert[itoa[atoi['5'] = 5] = '5'] = FIVE  = getDigit(5);
  convert[itoa[atoi['6'] = 6] = '6'] = SIX   = getDigit(6);
  convert[itoa[atoi['7'] = 7] = '7'] = SEVEN = getDigit(7);
  convert[itoa[atoi['8'] = 8] = '8'] = EIGHT = getDigit(8);
  convert[itoa[atoi['9'] = 9] = '9'] = NINE  = getDigit(9);

  TEN = getDigit(10);
  (TENk = getDigit(1))->d[0] = L(10000);
  (BILLION = getDigit(1))->d[0] = L(1000000000),
    BILLION->d[1] = H(1000000000); BILLION->u = 2;

  F        = getVar(256);
  R        = getVar(256);
  I        = getVar(256);
  Tmp      = getVar(256);
  Tmq      = getVar(256);
  Tmr      = getVar(256);
  Tmp0     = getVar(256);
  Tmp1     = getVar(256);
  Tmp2     = getVar(256);
  Tmp3     = getVar(256);
  Tmp4     = getVar(256);
  Tmp5     = getVar(256);
  Tmp6     = getVar(256);
  Tmp7     = getVar(256);
  Tmp8     = getVar(256);
  Tmp9     = getVar(256);
  Tmp10    = getVar(256);
  Tmp11    = getVar(256);
  Tmp12    = getVar(256);

  for (j = 1, i = 0; i < 256; m13[i++] = j = j%30030, j <<= 16);
  for (i = 0; i < 30030; i++){
    p13[i]  = i&1;
    p13[i] &= (i%3  > 0);
    p13[i] &= (i%5  > 0);
    p13[i] &= (i%7  > 0);
    p13[i] &= (i%11 > 0);
    p13[i] &= (i%13 > 0);
  }
  for (i = j = k = 0; i < 30030; i++){
    if ((h = p13[i] &= p13[(1+(i<<1))%30030])&&!k) k = i;
    j += h;
  }
  for (j = 30030/j, i = h = 0; i < 30030; i++)
    if (!p13[i]){
      p13[i] = S((30030+k-i)%30030);
      if (++h > j)
        for (h = 0; !L(p13[++k]); );
    }
  for (i = 0; i < 30030; p13[i] >>= 16, i++);
}

void release(bigint x)
{
  free (x->d);
  free (x);
}

bigint ass(bigint x, bigint y)
{
  int i;

  for(y->s = x->s, y->u = x->u, i = 0; i< x->u; i++) y->d[i] = x->d[i];
  return y;
}

int gtr(bigint x, bigint y)  /* assumes no garbage */
{
  int i;

  if (y->s > x->s) return Equal = 0;
  if (x->s > y->s) { Equal = 0; return 1; }
  if (x->u > y->u) { Equal = 0; return ((x->s > 0)? 1: 0); }
  if (y->u > x->u) { Equal = 0; return ((x->s > 0)? 0: 1); }

  for(Equal = 1, i = x->u; i--; ){
    if (x->d[i] > y->d[i]) { Equal = 0; return ((x->s > 0)? 1: 0); }
    if (y->d[i] > x->d[i]) { Equal = 0; return ((x->s > 0)? 0: 1); }
  }
  return 0;
}

#define les(x,y) (1^(gtr((x),(y))|Equal))
#define equ(x,y) ((gtr((x),(y))|Equal)&Equal)

bigint neg(bigint x)
{
  x->s *= -1;
  return x;
}

bigint mul(bigint x, bigint y, bigint z)
{
  int i,j,k = 0;  Unsigned q,r;

  if (z->s = (x->s)*(y->s)){
    for (k = x->u+y->u, i = 0; i <= k; z->d[i++] = 0);
    for(*(z->d) = i = 0; i < x->u; i++)
      for(j = 0; j < y->u; j++)
        z->d[j+i]     = L(r = z->d[j+i] + L(q = x->d[i]*y->d[j])),
          z->d[j+i+1] += H(q) + H(r);

    while (!z->d[k]) k--;
  }
  z->u = k+1;
  return z;
}

bigint sub(bigint,bigint,bigint);

bigint add(bigint x, bigint y, bigint z)
{
  int i,k = Max(x->u,y->u);  Unsigned q,r;

  if (!(x->s))      return ass(y,z);
  if (!(y->s))      return ass(x,z);
  if (y->s > x->s) { x->s *= -1; sub(y,x,z); x->s *= -1; return z; }
  if (x->s > y->s) { y->s *= -1; sub(x,y,z); y->s *= -1; return z; }

  for(*(z->d) = i = 0; i < k; i++)
    q = ((i<x->u)? x->d[i]: 0) + ((i<y->u)? y->d[i]: 0),
      z->d[i] = L(r = z->d[i] + L(q)), z->d[i+1] = H(q) + H(r);

  while (!z->d[k]) k--;
  z->u = k+1, z->s = x->s;
  return z;
}

bigint sub(bigint x, bigint y, bigint z)
{
  int a,b,i,k;  Int q,r;  bigint p;

  if (!(y->s))          return ass(x,z);
  if (!(x->s))        { ass(y,z); z->s *= -1; return z; }
  if (y->s != x->s) { y->s *= -1; add(x,y,z); y->s *= -1; return z; }

  a = x->s, b = y->s, x->s = y->s = 1;
  if (gtr(y,x))
    x->s = a, y->s = b, p = y, y = x, x = p, a = -a;
  else
    x->s = a, y->s = b;

  if (Equal) return ass(ZERO,z);

  for(*(z->d) = 1, i = 0; i < x->u; i++)
    z->d[i]     = L(r = z->d[i] + L(q = x->d[i]+((i<y->u)? y->d[i]^M: M))
),
      z->d[i+1] = H(q) + H(r);

  for (z->d[k = x->u] = 0; !z->d[--k]; );
  z->u = k+1, z->s = a;
  return z;
}

bigint div(bigint x, bigint y, bigint z) /* assumes no garbage */
{
  int a,b,i,k,q;  bigint t;
  if (!(b = y->s))    error("zero divide");
  if (!(a = x->s)) { ass(ZERO,R); return ass(ZERO,z); }

  z->s = a*b, x->s = y->s = 1; ass(x,R);

  if (gtr(y,x)) { R->s = x->s = a, y->s = b; return ass(ZERO,z); }

  Tmp0->s = Tmp0->u = 1, k = x->u - y->u, i = k+1, R->d += i, R->u -= i;
  while  (i){
    z->d[--i] = 0, R->d--, R->u++, R->s |= (*R->d > 0);
    while (!gtr(y,R)){
      if (Equal)
        q = 1;
      else if (R->u > y->u)
        q = (S(R->d[R->u-1])|R->d[R->u-2])/((y->u>1) + y->d[y->u-1]);
      else if (R->u > 1)
        q = (S(R->d[R->u-1])|R->d[R->u-2])/((y->u>2) + (S(y->d[y->u-1])|
y->d[y->u-2]));
      else
        q = R->d[R->u-1]/y->d[y->u-1];

      z->d[i] += *(Tmp0->d) = Max(q,1);
      sub(R,mul(Tmp0,y,Tmp1),Tmp2);
      ass(Tmp2,R);
    }
  }
  while (!z->d[k]) k--;
  z->u = k+1, R->s *= x->s = a, y->s = b;
  return z;
}

bigint mod(bigint x, bigint y, bigint z)
{
  div(x,y,z);
  if (R->s < 0) return add(y,R,z);
  return ass(R,z);
}

bigint pwr(bigint w, bigint x, bigint y, bigint z) /* w^x mod y */
{
  Int i,j,k;

  if (x->s){
    ass(ONE,z);
    ass(w,Tmp4);
    for (i = 0; i < x->u; i++)
      for (j = x->d[i], k = 0; k++ < 16; j >>= 1){
        if (j&1){
          mul(z,Tmp4,Tmp3);
          mod(Tmp3,y,z);
        }
        mul(Tmp4,Tmp4,Tmp3);
        mod(Tmp3,y,Tmp4);
      }
  } else ass(ONE,z);
  return z;
}

bigint gcd(bigint x, bigint y, bigint z)
{
  bigint p;

  ass(x,Tmp3);
  ass(y,R);
  ass(ONE,I);
  ass(ZERO,Tmp6);
  do {
    ass(R,z);
    div(Tmp3,z,Tmp5);
    ass(z,Tmp3);
    mul(Tmp6,Tmp5,Tmp4);
    sub(I,Tmp4,Tmp5);
    ass(Tmp6,I);
    ass(Tmp5,Tmp6);
  } while (R->s);

  return z;
}

bigint inv(bigint x, bigint y, bigint z) /* x^-1 mod y */
{
  if (equ(gcd(x,y,z),ONE)) return mod(I,y,z);
  return ass(ZERO,z);
}

/****************************** input / output ************************
*****/

void binary(FILE *f)
{
  Int c,i;

while ((c=getc(f)) != EOF) for (i = 0; i < 8; i++, c >>= 1)
putchar(itoa[c&1]);
}

bigint scan(char *s, bigint z)
{
  int i,j,k;  Unsigned t;

  if (*s == '-') s++, k = -1; else k = (*s == '0')? 0: 1;

  ass(ZERO,z);
  for (Tmp1->s = Tmp1->u = 1, t = i = 0; *s; i++){
    if (i > 3){
      Tmp1->d[0] = t, i = t = 0;
      mul(TENk,z,Tmp0);
      add(Tmp0,Tmp1,z);
    }
    t = 10*t + atoi[*(s++)];
  }
  for (Tmp2->s = Tmp2->u = j = 1; i--; j *= 10);
  Tmp1->d[0] = t, Tmp2->d[0] = j;
  mul(Tmp2,z,Tmp0);
  add(Tmp0,Tmp1,z);
  z->s = k;
  return z;
}

char *print(bigint x, char *s)  /* assumes no garbage */
{
  int h,i,j = 0, k = x->s;  Unsigned t;  char *r = s;

  if (!k) { s[0] = '0', s[1] = '\0'; return s; }

  ass(x,Tmp); ass(R,Tmr);
  for (Tmp->s = 1; gtr(Tmp,ZERO);){
    if (Tmp->u > 2){
      div(Tmp,BILLION,Tmq);
      t = (R->s)? ((R->u > 1)? S(R->d[1])|*R->d: *R->d): 0;
      for (i = 9; i--; ptr[j++] = itoa[t%10], t /= 10);
    }
    else if (Tmp->u > 1){
      div(Tmp,TENk,Tmq);
      t = (R->s)? *R->d: 0;
      for (i = 4; i--; ptr[j++] = itoa[t%10], t /= 10);
    }
    else {
      div(Tmp,TEN,Tmq);
      ptr[j++] = (R->s)? itoa[*R->d]: '0';
    }
    ass(Tmq,Tmp);
  }
  if (k < 0) *(s++) = '-';
  while (j--) *(s++) = ptr[j]; *s = '\0';
  ass(Tmr,R);
  return r;
}

/****************************** random numbers **************************
*/

Unsigned rtab[55];
Int      rndx;

#define TWO_32 (4294967296.0)
#define rndm() ((++rndx>54)?rtab[rndx=nrndm()]:rtab[rndx])
#define U01()  (rndm()/TWO_32)
#define rnd(n) ((int)(U01()*(n)))

nrndm()
{
  int i;

  for (i =  0; i < 24; i++) rtab[i] -= rtab[i+31];
  for (i = 24; i < 55; i++) rtab[i] -= rtab[i-24];
  return 0;
}

initrand(unsigned s)
{
  int h,i;  Unsigned j = s|1, k = 1;

  rtab[54] = s;
  for (i = 1; i < 55; h = (21*i++)%55, rtab[--h] = k, k = j-k, j = rtab[
h]);
  nrndm(); nrndm(); nrndm(); nrndm(); rndx = 0;
}

bigint prandom(int u, bigint y)
{
  Int i,j;

  for (Tmp8->u = u, Tmp8->s = 1, j = i = 0; i < u; i++, j = j%30030)
    j += (((Tmp8->d[i] = rnd(M))*m13[i])%30030);
  while (i && !Tmp8->d[--i]) Tmp8->u--;

  if (!p13[j]) return ass(Tmp8,y);
  Tmp1->s = 1, Tmp1->u = 2, *Tmp1->d = L(p13[j]), Tmp1->d[1] = H(p13[j])
;
  return add(Tmp8,Tmp1,y);
}

bigint random(bigint x, bigint y) /* y is sort of random and < x  */
{
  int i;

  if (!x->s) return ass(ZERO,y);
  if (1 == (i = x->u))
    *y->d = rnd(*x->d);
  else
    for (i--, y->d[i] = rnd(x->d[i]); i--; y->d[i] = rnd(S(1)));

  for (i = x->u; i-- && !y->d[i]; ); y->s = ((y->u = i+1)>0);
  return y;
}

/********************** finding / testing "primes"
*************************/

int miller(bigint x, bigint b)
{
  int i,j,k,l;

  ass(sub(x,ONE,Tmp5),Tmp7);

  for (Tmp6->s = 1, Tmp6->u = Tmp5->u, i = k = 0; !Tmp5->d[i]; i++, Tmp6
->u--);
  while (!(Tmp5->d[i]&1)) k++, Tmp5->d[i] >>= 1; *Tmp6->d = Tmp5->d[i];
  for (j = (i<<4)+k, l = 0; ++i < x->u; )
    Tmp6->d[l++] |= L(S(Tmp5->d[i])>>k), Tmp6->d[l] = Tmp5->d[i]>>k;

  if (equ(pwr(b,Tmp6,x,Tmp5),ONE)) return 1;
  while (j--){
    if (equ(Tmp5,Tmp7)) return 1;
    mul(Tmp5,Tmp5,Tmp6);
    mod(Tmp6,x,Tmp5);
  }
  return 0;
}

int rabin(int i, bigint x)
{
  while (i--) if (gtr(random(x,Tmp8),ONE) && !miller(x,Tmp8)) return 0;
  return 1;
}

bigint prime(int u, bigint y) /* y = 2q+1 and q are probably prime */
{
 top:
  do random(prandom(u,Tmp9),Tmp8); while (!miller(Tmp9,Tmp8));
  add(ONE,mul(TWO,Tmp9,Tmp0),y);
  if (!equ(pwr(FOUR,Tmp9,y,Tmp8),ONE)) goto top;
  if (!rabin(31,Tmp9)) goto top;
  return y;
}

/********************** simple minded factoring
*************************/

int factor(bigint x) /* might not return in your lifetime */
{
  Unsigned i = 0;

  if (!rabin(32,x)){
    ass(ONE,Tmp7);
    ass(TWO,Tmp8);
    while (++i){
      add(ONE,Tmp7,Tmp6);
      ass(Tmp6,Tmp7);
      pwr(Tmp8,Tmp7,x,Tmp6);
      ass(Tmp6,Tmp8);
      if (!(i&1023)){
      sub(Tmp8,ONE,Tmp9);
      gcd(Tmp9,x,F);
      if (gtr(F,ONE)&&les(F,x)) return 1;
      else printf("not yet...\n");
      }
    }
  }
  return 0;
}

bigint proot(bigint p, bigint r)
{
  bigint temp0, temp1;

  temp0 = getVar(256);
  temp1 = getVar(256);

  /* Assume p is a prime such that p = */
  /* 2q+1 where q is prime             */

  sub(p, TWO, temp0);
  mul(temp0, temp0, temp1);
  sub(p, temp1, temp0);
  mod(temp0, p, r);

  return r;
}

void rsa(bigint n, bigint d, bigint e)
{
  bigint temp0, temp1, phi_n, a, b;
  bigint test;

  temp0 = getVar(256);
  temp1 = getVar(256);
  phi_n = getVar(256);
  a = getVar(256);
  b = getVar(256);
  test = getVar(256);
  prime(2,a);
  prime(2,b);

  mul(a, b, n);

  sub(a, ONE, temp0);
  sub(b, ONE, temp1);
  mul(temp0, temp1, phi_n);

  do prandom(2, d); while (equ(ZERO,inv(d, phi_n, e)));
}

void rsa2(bigint n, bigint d, bigint e, bigint b)
{
  bigint temp0, temp1, p1, p2;
  bigint test;

  temp0 = getVar(256);
  temp1 = getVar(256);
  p1 = getVar(256);
  p2 = getVar(256);
  test = getVar(256);

  ass(ZERO, test);
  while(!equ(test, THREE))
    {
      prime(2, p1);
      mod(p1, FOUR, test);
    }

  ass(ZERO, test);
  while(!equ(test, THREE))
    {
      prime(2, p2);
      mod(p2, FOUR, test);
    }

  ass(p1, d);
  ass(p2, e);
  mul(p1, p2, n);
  prime(2, test);
  mod(test, n, b);

}

void getkey(bigint n, bigint key, char *newkey)
{
  FILE *keyfile;
  char skey[1000], sn[1000];
  int i;

  keyfile = fopen(newkey, "r");

  fscanf(keyfile, "%s", sn);
  fscanf(keyfile, "%s", skey);

  scan(skey, key);
  scan(sn, n);

  fclose(keyfile);
}
```

```
void squareroot(bigint x, bigint n, bigint z)
{
  bigint n_plus_one, temp1;
  char s4[1000], sn1[1000], sx[1000], sz[1000];

  n_plus_one = getVar(256);
  temp1 = getVar(256);

  add(n, ONE, n_plus_one);

  div(n_plus_one, FOUR, temp1);
  pwr(x, temp1, n, z);

  /*print(n_plus_one, sn1);
    print(x, sx);
    print(z, sz);

    printf("n+1 = %s\n", sn1);
    printf("x = %s\n", sx);
    printf("z = %s\n", sz);*/

}

void encrypt(bigint m, bigint b, bigint n, bigint c)
{
  bigint temp1, temp2;
  char s[1000];

  temp1 = getVar(256);
  temp2 = getVar(256);

  add(m, b, temp1);
  mul(m, temp1, temp2);
  mod(temp2, n, c);

  /*print(m, s);
    printf("m = %s\n", s);
    print(b, s);
    printf("b = %s\n", s);
    print(n, s);
    printf("n = %s\n", s);
    print(temp1, s);
    printf("temp1 = %s\n", s);
    print(temp2, s);
    printf("temp2 = %s\n", s);
    print(c, s);
    printf("c = %s\n", s);*/
}

void decrypt(bigint c, bigint p1, bigint p2, bigint m)
{
  bigint b, b_p1, b_p2, c_p1, c_p2, temp1, x_p1, x_p2, two_inv;
  bigint four_inv, temp2, n;
  char s[1000];

  b = getVar(256);
  n = getVar(256);
  b_p1 = getVar(256);
  b_p2 = getVar(256);
  temp1 = getVar(256);
  temp2 = getVar(256);
  c_p1 = getVar(256);
  c_p2 = getVar(256);
  x_p1 = getVar(256);
  x_p2 = getVar(256);
  two_inv = getVar(256);
  four_inv = getVar(256);
  getkey(n, b, "pubkey");
  inv(TWO, n, two_inv);
  inv(FOUR, n, four_inv);
  mod(b, p1, b_p1);
  mod(c, p1, c_p1);
  mod(b, p2, b_p2);
  mod(c, p2, c_p2);
  release(b);

  /* (x + B/2)^2 = C + (B^2/4) mod p */
  mul(b_p1, b_p1, temp1);
  mul(temp1, four_inv, temp2);
  add(temp2, c_p1, x_p1);
  squareroot(x_p1, p1, temp1);
  if (sqcheck&1) /* negative */
    {
      sub(p1, temp1, x_p1);
      ass(x_p1, temp1);
    }
  mul(b_p1, two_inv, temp2);
  sub(temp1, temp2, x_p1);
  mod(x_p1, p1, temp1);
  ass(temp1, x_p1);

  mul(b_p2, b_p2, temp1);
  mul(temp1, four_inv, temp2);
  add(temp2, c_p2, x_p2);
  squareroot(x_p2, p2, temp1);
  if (sqcheck&2) /* negative */
    {
      sub(p2, temp1, x_p2);
      ass(x_p2, temp1);
    }
  mul(b_p2, two_inv, temp2);
  sub(temp1, temp2, x_p2);
  mod(x_p2, p2, temp1);
  ass(temp1, x_p2);

  inv(p1, p2, b_p1);
  inv(p2, p1, b_p2);

  mul(b_p2, p2, temp1);
  mul(temp1, x_p1, temp2);
  mod(temp2, n, x_p1);

  mul(b_p1, p1, temp1);
  mul(temp1, x_p2, temp2);
  mod(temp2, n, x_p2);

  add(x_p1, x_p2, temp1);
  mod(temp1, n, m);
}

#if 0
```

```
void checksq(FILE *fp, bigint key, bigint n, bigint p1, bigint p2)
{
  bigint block, check, seventeen;
  int i, setsq = -1;
  unsigned char ch;
  char s[1000];

  block = getVar(256);
  check = getVar(256);
  seventeen = getVar(256);

  add(TEN, SEVEN, seventeen);

  for(i = 0; (i < (n->u<<1)) && (EOF != (ch = getc(fp))); i++)
    if (i&1) block->d[i>>1] |= ch<<8; else block->d[i>>1]= ch;
  for (i = n->u; i-- && !block->d[i]; ); block->s = ((block->u =
i+1)>0);

  for(i = 0; i < 4; i++)
    {
      sqcheck = i;
      decrypt(block, p1, p2, check);
      if(equ(seventeen, check))
      setsq = i;
    }
  print(block, s);
  printf("block = %s\n", s);
  printf("setsq = %d\n", setsq);
  sqcheck = setsq;
}
#endif

main(int argc, char **argv)
{
  char sn[1000], skey[1000], sblock[1000], sout[1000];
  bigint n, key, block, output, p1, p2, seventeen;
  int i, j, ndigit, ch;
  FILE *filein, *fileout;

  /*
    e - encryption.  Provide the infile and keyfile.
    d - decryption.  Provide the cypher file and keyfile.
    c - creation. Create two files, named in the <infile> and <keyfile>
    arguments, that will have the public and secret keys, as well as n.
  */
  if(argc > 4)
    {
      printf("Error: Too many arguments.\n");
      printf("mycrypt: mycrypt [inputfile] [keyfile] [action]\n");
    }
  else if(argc < 4)
    {
      printf("Error: Not enough arguments.\n");
      printf("mycrypt: mycrypt [inputfile] [keyfile] [action]\n");
    }
  else
    {
      init(5);
      n = getVar(256);
      key = getVar(256);
      p1 = getVar(256);
      p2 = getVar(256);
      seventeen = getVar(256);
      add(TEN, SEVEN, seventeen);

      if(argv[3][0] == 'c')
      {
        block = getVar(256);
        output = getVar(256);
        rsa2(n, key, block, output);
        filein = fopen(argv[1], "w");
        fileout = fopen(argv[2], "w");
        print(n, sn);
        print(key, skey);
        print(block, sblock);
        print(output, sout);
        fprintf(filein, "%s\n%s", sn, sout); /*write public key n, b*/
        fprintf(fileout, "%s\n%s", skey, sblock);/*secret key p1 p2*/
        fclose(filein);
        fclose(fileout);
      }
      else if(argv[3][0] == 't')
      {
        block = getVar(256);
        output = getVar(256);
        add(TWO, TEN, block);
        getkey(n, key, "pubkey");
        encrypt(block, key, n, output);
        getkey(p1, p2, "seckey");
        decrypt(output, p1, p2, block);
        print(block, sblock);
        printf("block = %s\n", sblock);
      }
      else if(argv[3][0] == 'e')
      {
        getkey(n, key, argv[2]);
        ndigit = n->u - 1;
        block  = getVar(ndigit);
        output = getVar(ndigit);
        filein  = fopen(argv[1], "r");
        fileout = fopen("cfile", "w");

        /*begin encrypting file*/
        do
          {
            for( i = 0; (i < (ndigit<<1)) && (EOF != (ch =
getc(filein))); i++)
            if (i&1) block->d[i>>1] |= ch<<8; else block->d[i>>1] = ch;

            while (i < (ndigit<<1)){
            if (i&1) block->d[i>>1] |= 0<<8; else block->d[i>>1] = 0;
            i++;
            }
            for (i = ndigit; i-- && !block->d[i]; ); block->s = ((block-
>u = i+1)>0);
            encrypt(block, key, n, output);
            for(i = 0; i< output->u; i++){
            fputc(output->d[i], fileout);
            fputc(output->d[i]>>8, fileout);
            }
            while (i++ < n->u) { fputc(0, fileout); fputc(0, fileout); }
          } while(!feof(filein));
        fclose(filein);
        fclose(fileout);
      }
      else if(argv[3][0] == 'd')
      {
        // sscanf(argv[4],"%d",&sqcheck);
        getkey(n, key,"pubkey");
        getkey(p1, p2, argv[2]);
        ndigit = n->u - 1;
        block  = getVar(ndigit);
        output = getVar(ndigit);

        filein  = fopen(argv[1], "r");
        fileout = fopen("output", "w");

        do
          {
            for( i = 0; (i < (n->u<<1)) && (EOF != (ch = getc(filein)));
i++)
              if (i&1) block->d[i>>1] |= ch<<8; else block->d[i>>1] =
ch;

            for (i = n->u; i-- && !block->d[i]; ); block->s = ((block->u
= i+1)>0);

            for (sqcheck = 0; sqcheck < 4; sqcheck++){
              decrypt(block, p1, p2, output);
#if 0
            for(i = 0; i< output->u; i++){
              putchar(output->d[i]);
              putchar(output->d[i]>>8);
            }
            fflush(stdout);
#endif
            for (j = 1, i = 0; i< output->u; i++)
              if ((128|(128<<8))&output->d[i]) { j = 0; break; }
            if (j)
              break;
            }

            for(i = 0; i< output->u; i++){
            fputc(output->d[i], fileout);
            fputc(output->d[i]>>8, fileout);
            }
            while (i++ < ndigit) { fputc(0, fileout); fputc(0, fileout);
}

          } while(!feof(filein));
        fclose(filein);
        fclose(fileout);
      }
    }
}
/*
gcc -g -o rsa2.c

*/
```