INTERNET & WEB TECHNOLOGY

FINAL REPORT

anirudhnakra4@gmail.com
abhijeetvats01@gmail.com

Delhi Technological University

PROPOSED BY

ANIRUDH NAKRA 2K17/EC/22
ABHIJEET VATS 2K17/EC/03

ABSTRACT

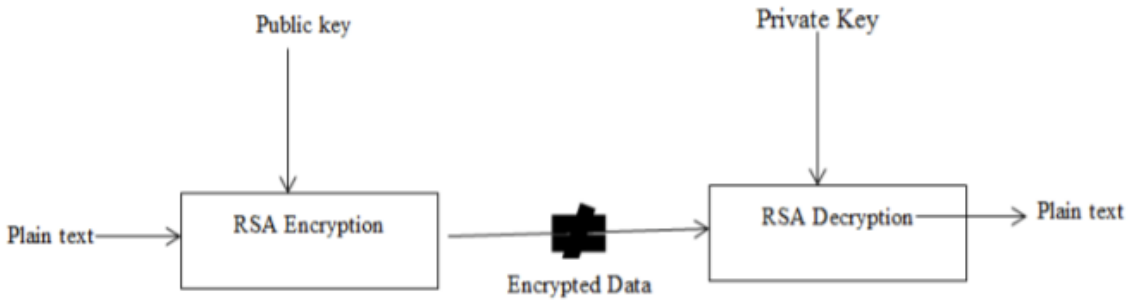
The success rate of various electronic mechanisms such as E-Governance, E-Learning, E-Shopping, E-Voting, etc. is absolutely dependent on the security, authenticity and the integrity of the information that is being transmitted between the users of sending end and the users of receiving end. Online trading is growing widely day by day, which makes safety the biggest concern while carrying out trading by electronic means. As many other operations can be done with digital environment and internet, operation that provides identity validation should also be added to the digital environment.

When data are transferred, the user should make sure that there are no changes in the original data while transferring them from sender to receiver. And it has also become necessary to authenticate the users often to ensure security and to avoid fraud. There are lot of different ways of online identification, in which digital signature is considered to be one of the powerful ways of authentication. So, the online user uses digital signature to authenticate the sender and to maintain the integrity of the document sent. Since digital signature schemes are basically various complex cryptographic algorithms which are embedded with the plain text message, the performance level of these E-services varies based on certain attributes like key size, block size, computational complexities, security parameters, application specific customizations, etc. In this paper, we review and compare some of these implementation methods to optimize signing procedure. We while comparing different cryptographic schemes also pursue the implementation of different ciphers and hashing algorithms.

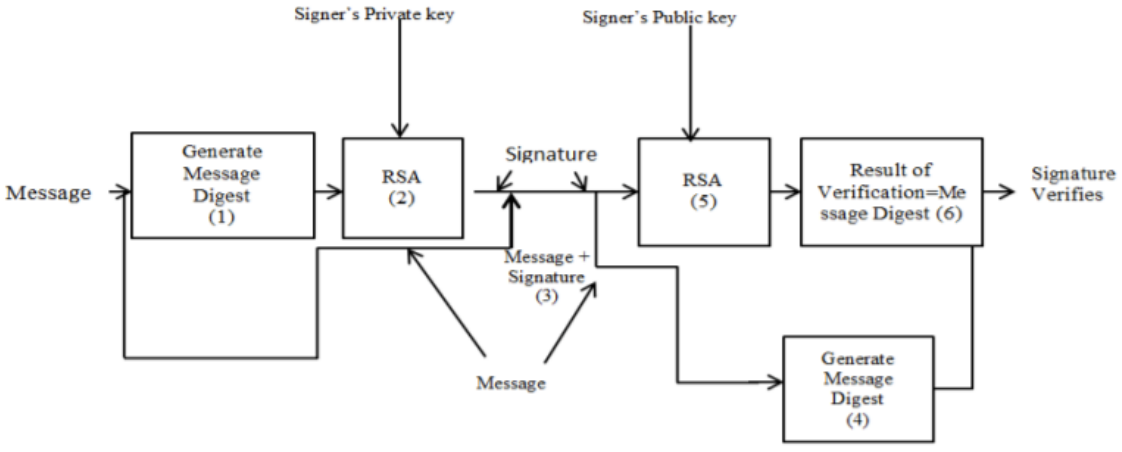
KEYWORDS: Digital signature, Cryptography, Authentication, Public Key Cryptography, Comparative study, Cipher and Hashing implementation

INTRODUCTION

In IS07498-2 standard, digital signature is defined as: "Some data affixed on data cells, or cryptographic transformation of data cells, these data and transformation allow the recipient of the data cells to confirm the data cell source and integrity of the data cell, in order to protect the data from being forged by somebody(for example, recipient) ". Specifically, digital signature is an alphabetic string obtained through processing the transmitted text by a Hash, with the purpose of verify the source of text and confirm whether the text is undergoing changes. Digital signature is generated and processed by cryptographic technology and its security depends on the security level of the cryptosystem, so a security level higher than the handwritten signature is achieved. In 1994, U.S. Government formally issued DDS, or Digital Signature Standard. In 1995, China's digital signature standard (GB15851-1995) was formulated. The digital signature is presented in many forms such as ordinary digital signature, arbitrated digital signature, undeniable signature, blind signature, group signature, threshold signature, etc. Digital signature is an electronized signature on an electronic document generated by cryptographic technology, but it's not the



digital image version of the handwritten signature. It is something like the handwritten signature or stamp, and it can be called electronic stamp. Upon signing on some important documents, we may confirm their validity. However, falsifying the traditional signature is not a difficult task, thus highlighting the important difference between the digital signature and traditional signature: without the private key that generates signature, it is technically impossible to falsify the signature that is generated by security code digital signature scheme.



Digital signatures rely on certain types of **encryption** to ensure authentication. Encryption is the process of taking all the data that one computer is sending to another and encoding it into a form that only the other computer will be able to decode. Authentication is the process of verifying that information is coming from a trusted source. These two processes work hand in hand for digital signatures.

There are several ways to authenticate a person or information on a computer:

Password - The use of a user name and password provide the most common form of authentication. You enter your name and password when prompted by the computer. It checks the pair against a secure file to confirm. If either the name or password do not match, then you are not allowed further access.

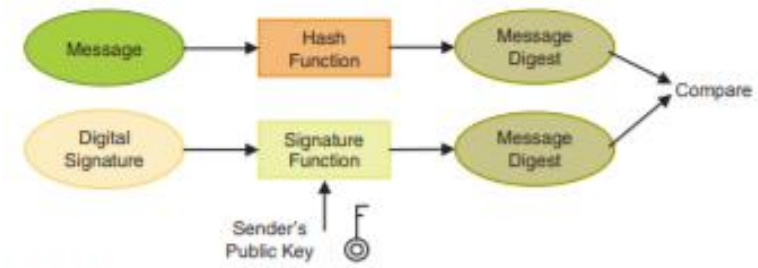


Fig. 2 Verifying a digital signature

Checksum - Probably one of the oldest methods of ensuring that data is correct, checksums also provide a form of authentication since an invalid checksum suggests that the data has been compromised in some fashion. A checksum is determined in one of two ways. Let's say the checksum of a packet is 1 byte long, which means it can have a maximum value of 255. If the sum of the other bytes in the packet is 255 or less, then the checksum contains that exact value. However, if the sum of the other bytes is more than 255, then the checksum is the remainder of the total value after it has been divided by 256. Look at this example:

- Byte 1 = 212
- Byte 2 = 232
- Byte 3 = 54
- Byte 4 = 135
- Byte 5 = 244
- Byte 6 = 15
- Byte 7 = 179
- Byte 8 = 80
- **Total = 1151**. 1151 divided by 256 equals 4.496 (round to 4). Multiply 4 X 256 which equals 1024. 1151 minus 1024 **equals checksum of 127**

CRC (Cyclic Redundancy Check) - CRCs are similar in concept to checksums but they use polynomial division to determine the value of the CRC, which is usually 16 or 32 bits in length. The good thing about CRC is that it is very accurate. If a single bit is incorrect, the CRC value will not match up. Both checksum and CRC are good for preventing random errors in transmission, but provide little protection from an intentional attack on your data. The encryption techniques below are much more secure.

Private key encryption -Private key means that each computer has a secret key (code) that it can use to encrypt a packet of information before it is sent over the network to the other computer. Private key requires that you know which computers will talk to each other and install the key on each one. Private key encryption is essentially the same as a secret code that the two computers must each know in order to decode the information. The code would provide the key to decoding the message. Think of it like this. You create a coded message to send to a friend where each letter is substituted by the letter that is second from it. So "A" becomes "C" and "B" becomes "D". You have already told a trusted friend that the code

is "Shift by 2". Your friend gets the message and decodes it. Anyone else who sees the message will only see nonsense.

Public key encryption - Public key encryption uses a combination of a private key and a public key. The private key is known only to your computer while the public key is given by your computer to any computer that wants to communicate securely with it. To decode an encrypted message, a computer must use the public key provided by the originating computer and it's own private key.

The key is based on a hash value. This is a value that is computed from a base input number using a hashing algorithm. The important thing about a hash value is that it is nearly impossible to derive the original input number without knowing the data used to create the hash value. Here's a simple example:

Input number 10667

Hashing Algorithm = Input # x 143

Hash Value = 1525381

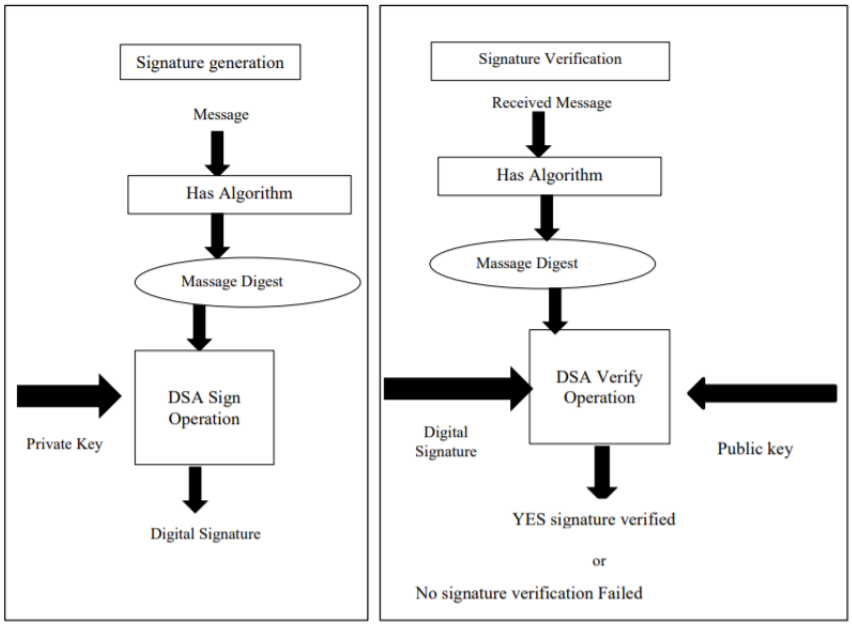


Figure 1 Digital signature generation and verification

You can see how hard it would be to determine that the value of 1525381 came from the multiplication of 10667 and 143. But if you knew that the multiplier was 143, then it would be very easy to calculate the value of 10667. Public key encryption is much more complex than this example but that is the basic idea. Public keys generally use complex algorithms and very large hash values for encrypting: 40-bit or even 128-bit numbers. A 128-bit number has a possible 2^{128} different combinations. That's as many combinations as there are water molecules in 2.7 million olympic size swimming pools. Even the tiniest water droplet you can image has billions and billions of water molecules in it!

Digital certificates - To implement public key encryption on a large scale, such as a secure Web server might need, requires a different approach. This is where digital certificates come in. A digital certificate is essentially a bit of information that says the Web server is trusted by an independent source known as a *Certificate Authority*. The Certificate Authority acts as the middleman that both computers trust. It confirms that each computer is in fact who they say they are and then provides the public keys of each computer to the other.

The *Digital Signature Standard (DSS)* is based on a type of public key encryption method that uses the *Digital Signature Algorithm (DSA)*. DSS is the format for digital signatures that has been endorsed by

the US government. The DSA algorithm consists of a private key that only the originator of the document (signer) knows and a public key. The public key has four parts, which you can learn more about at this page.

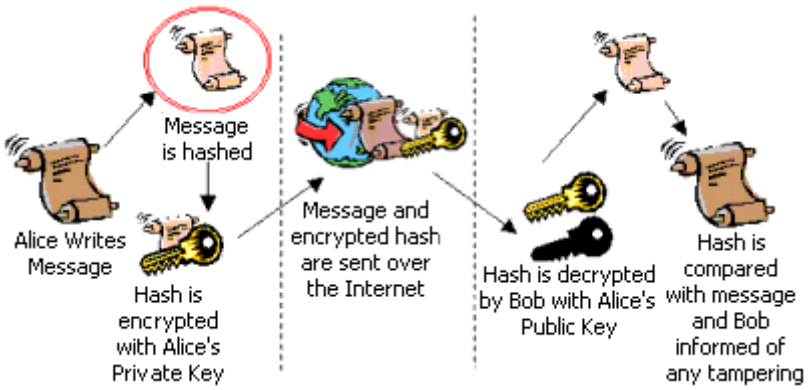
Electronic payment could become the future of currency. Click here to learn how digital signatures could help secure the future of electronic payment.

WORKING OF A DIGITAL SIGNATURE

The digital signature can be considered as a numerical value that is represented as a sequence of characters. The creation of a digital signature is a complex mathematical process that can only be created by a computer.

Consider a scenario where Alice has to digitally sign a file or an email and send it to Bob.

- Alice selects the file to be digitally signed or clicks on 'sign' in her email application
- The hash value of the file content or the message is calculated by Alice's computer
- This hash value is encrypted with Alice's Signing Key (which is a Private Key) to create the Digital Signature.
- Now, the original file or email message along with its Digital Signature are sent to Bob.
- After Bob receives the signed message, the associated application (such as email application) identifies that the message has been signed. Bob's computer then proceeds to:
    - Decrypt the Digital Signature using Alice's Public Key
    - Calculate the hash of the original message
    - Compare the (a) hash it has computed from the received message with the (b) decrypted hash received with Alice's message.
- Any difference in the hash values would reveal tampering of the message.



CREATION OF A DIGITAL SIGNATURE

You can obtain a digital signature from a reputable certificate authority such as Sectigo, or you can create it yourself. You need a digital certificate to digitally sign a document. However, if you create and use a self-signed certificate the recipients of your documents will not be able to verify the authenticity of your digital signature. They will have to manually trust your self-signed certificate.

If you want the recipients of your documents to be able to verify the authenticity of your digital signature then you must obtain a digital certificate from a reputable CA. After downloading and installing the

certificate - you will be able to use the 'Sign' and 'Encrypt' buttons on your mail client to encrypt and digitally sign your emails. This makes more sense in a business scenario, as it assures the recipient that it was genuinely sent by you and not by some impersonator.

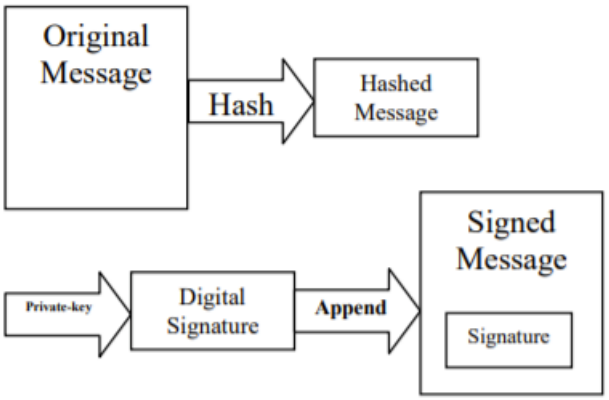


Fig 1 steps of generating Digital Signature

USES OF DIGITAL SIGNATURES

Sometimes you need proof that the document came from you and no one has tampered with it since you sent it. Digital Signature with your SSL Certificate fills the bill. On the other hand, sometimes you need to prove that a document came from someone else and has not been altered along the way. In legal matters, for example, you may need to prove that a contract has not been altered since someone sent it as an email. Because the computer tenaciously pairs the Digital Signature to one saved version of the document, it is nearly impossible to repudiate a digitally signed document. Or, if you are a developer distributing software online, you may need to reassure your customers that your executables really are from you. Put a Code Signing Certificate in your toolkit.

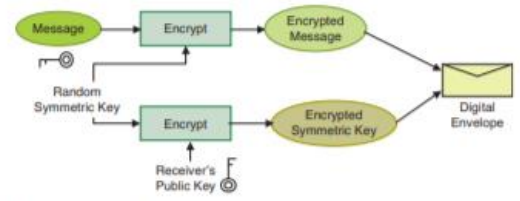


Fig. 3 Creating a digital envelope

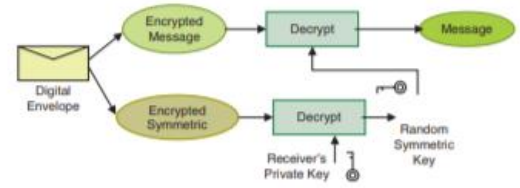


Fig. 4 Opening a digital envelope

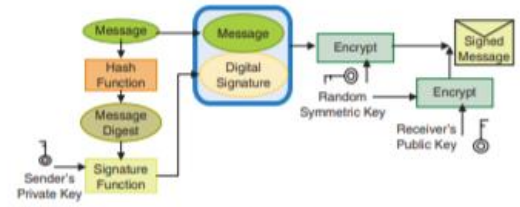


Fig. 5 Creating a digital envelope carrying a signed message

TYPES OF DIGITAL SIGNATURES

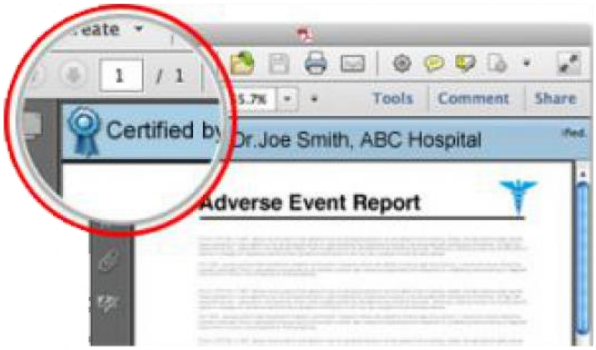
Different document processing platforms support and allow the creation of different types of digital signatures.

- Adobe supports - certified and approval digital signatures
- Microsoft Word supports - visible and non-visible digital signatures

Certified Signatures

Adding a certifying signature to a PDF document indicates that you are the author of the document and want to secure the document against tampering.

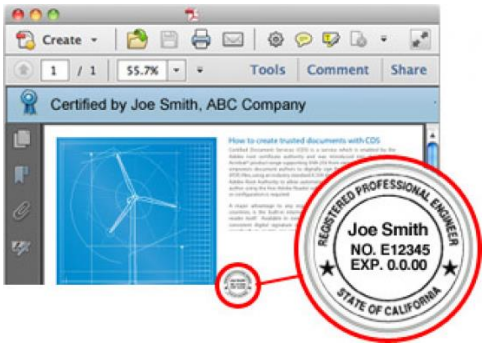
Certified PDF documents display a unique blue ribbon across the top of the document. It contains the name of the document signer and the certificate issuer to indicate the authorship and authenticity of the document.



Approval Signatures

Approval signatures on a document can be used in your organization's business workflow. They help optimize your organization's approval procedure. The process involves capturing approvals made by you and other individuals and embedding them within the PDF document.

Adobe allows signatures to include details such as an image of your physical signature, date, location, and official seal.



Visible Digital Signatures

These allow a single user or multiple users to digitally sign a single document. The signatures would appear on the document in the same way as signatures are applied on a physical document.

Invisible Digital Signatures

Documents with invisible digital signatures carry a visual indication of a blue ribbon in the task bar. You can use invisible digital signatures when you do not have to or do not want to display your signature, but you need to provide indications of the authenticity of the document, its integrity, and its origin.

ROLES OF DIGITAL SIGNATURE

One may deny that he applied signature on a document of a meeting. However, to deny a digital signature is far more difficult, to do this is to fundamentally prove that the security of private key is endangered before generation of digital signature. This is due to the fact that generation of digital signature may require private key, whilst the corresponding public key is used to certify the signature. Consequently, one key feature of digital signature is nonrepudiation. Some of existing schemes such as digital signature may bind together the identity of an entity (individual, organization or system) with a private key and a public key, making it difficult for a person to deny digital signature.

The digital signature, aside from handwritten signature, shall comply with the following requirements:

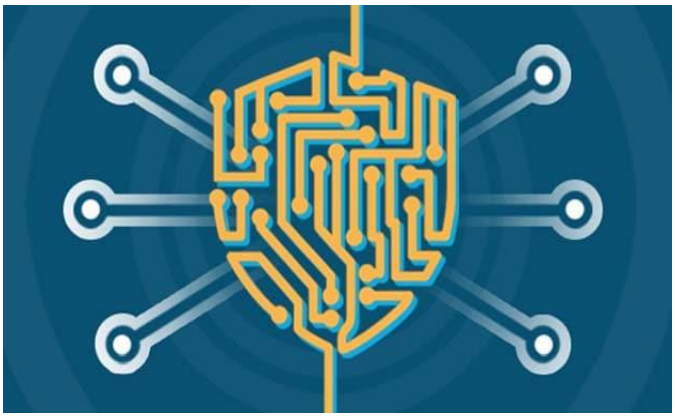
• The recipient may confirm or validate the signature of sender, but he can't falsify;

• When the sender sends the signature message to the recipient, he'll no longer be able to deny the sent message;

• When the sender/recipient have disputes on the content and source of message, they may provide the arbitrator with evidence that the sender has put a signature on the message that has already been sent. However, digital signature is different from handwritten signature, the handwritten signature is analogous and varies from one person to another, so simulation is possible no matter what language is used; digital signature is a string consists of digits 0 and 1 which varies by message and is not possible to simulate. the basis of digital signature is public key cryptography. Public key cryptography is the most fundamental invention and development of modem cryptography (according to general understanding, cryptography is to ensure the secrecy of information transfer, but this is one of many aspects of cryptography of modem age).

PROBLEMS AND COUNTERMEASURES IN APPLICATION OF DIGITAL SIGNATURE

Only in a sound information network system, can the security of on-line-transferred information be ensured and digital signature technology plays its due roles. The digital 499 signature technology doesn't resolve encryption problem of the message text itself, for this reason multiple encryption technologies shall be utilized to completely resolve a series of problems such as data security and identity verification, etc. China's information security technology started behind that of the foreign countries, encryption technology relatively lags behind, bringing higher security risks and technical risks to China's digital signature development. For this reason, we must make great efforts to develop advanced and proprietary



information technologies in order to build a complete information network security system. This system shall feature the comprehensive usage of computer system security technologies and multiple encryption technologies, to ensure confidentiality of e-commerce, data integrity, reliability and controllability.

FURTHER DEVELOPMENT OF DIGITAL SIGNATURE

Digital signature technology still requires further development. For example, digitally signed document may be repetitively used by the recipient. If the signed document is a check, it'll be easy for the recipient to repetitively encash that electronic check. For this reason, the sender shall affix the document with some specific marks of check. For example, the sender may leave time stamp on the electronic document to prevent this event from happening. Specifically, the sender encrypts the document that requires time stamp by Hash algorithm to generate summary, then send the summary to a dedicated service agency, the agency will add in the time information of the moment of receiving the document, then add in digital signature on that document, then send back to the user.

INFLUENCE OF DIGITAL SIGNATURES ON PRIVACY LAWS

Laws and codes in relation with digital signature need to be further improved to allow a more active role of digital signature. E-commerce is a complicated system engineering project, aside from technical research and application, it is also important to improve social and legal environment, including promotion and establishment of relevant laws, codes and Internet ethics. To this end, security of ecommerce is primarily based on establishment of laws, codes and relevant policies. A sound array of laws and policies will enable people to set up sound security management system by utilizing security transaction technologies and network security technologies. On August 28, 2004, the 11 th the Standing Committee Session of the 10th National People's Congress voted and passed Electronic Signature Law of the People's Republic of China, for the first time granting digital signature an legal effect equal to handwritten signature or stamp, and clarified the market access system of electronic certification service. The law is China's first e-commerce law in the true sense and a milestone of China's e-commerce development. Promulgation and implementation of the law will greatly improve the legal environment of China's ecommerce, promote the establishment of a safe and practical electronic transaction environment, and then greatly drive the development of China's e-commerce.

DIGITAL SIGNATURE'S APPLICATION SCHEMES IN NETWORK SECURITY

In on-line applications, suffering from limited PC speed, the information transfer is bottlenecked. Most of digital signature schemes nowadays has problems such as low efficiency and high cost, etc. For this reason, how to simplify the operation under the condition of security assurance is also a problem of digital signature technology that requires prompt resolution. The following paragraphs will present the digital signature scheme in on-line scenario, illustrated by the example of "A sends B a document". Given A is the information sender, whose public key/private key pair is (kaI, ka2), B is information recipient whose public key/private key pair is (kbI, kb2). A time stamp parameter P is applied to further enhance signature security. At the same time, considering the fact that the signed text varies in size, if putting signature directly on plain text, the counting operation duty may be heavy and the operation time may be long thus not applicable to on-line transfer. For this reason, first to use SHAI algorithm to generate a message summary of the plain text, then put signature on both P and summary to achieve dual certification, greatly improving the security level.

LITERATURE REVIEW

The German digital signature act came into force in the August of 1997 even before the EU‟s Directive; in fact, the directive was adjusted to conform to the German Digital signature Act. In the December of 1997 the European Union invited a proposal for its electronic signature directive with the purpose of making electronic signatures at least as binding as paper-based signatures in abide to facilitate "free movement of goods and service in the internal market". On the off chance that this yields the right message, then it is obvious that the message was surely scrambled by the private key of an, and consequently just A could have sent it. Herzberg in a private communication suggested signing the viewing program as well as the document, ensuring that the data displayed is viewed as it was intended. Austria fully implemented the directive in 1999; and concretized the malleability problem by specifying that only data formats may be used which have an "available specification" and which exclude "dynamic changes" or "invisibilities."

Ulrich Pordesch, a German researcher viewed it a risk to have other agencies verify and sign a document, "imbedding and using the schemes in application systems involves considerable risks, in particular, if the signer or the verifier uses an application environment which is maintained, used, or controlled, by other persons or organizations." He used personal signature for authentication purpose. His scheme involved having a Personal Digital Assistant to which documents could be transmitted for scrutinization. This scenario would behave like a safe room where the documents are inspected and verified. This strategy though avoids the problem of how secure the Personal Digital Assistant is and whether it has the capability to process dynamic content. Concurrent with the conference publication of the main results of this thesis, Audun Jφsang, a senior research scientist with the "Distributed Systems Technology Centre", published a very interesting paper based on the same area but with orthogonal results. Audun Jφsang's approach differed in terms of the depth and direction of attacks proposed. In his example of changing content based on browser type he used the differences in handling of HTML tags in Netscape Communicator and in Internet Explorer. Thus, showing that there are many ways to carry out attacks. In this paper he also considered various attacks on XML signatures. Sometimes the two XML documents may look same but may be used for two different applications.

In today's information age, it is impossible to imagine the world without internet. This modern era is dominated by paperless offices mail messages, cash transactions and virtual departmental stores. Large amounts of data are transferred between computers over the internet for professional as well as personal reasons. The computers are interconnected with each other, thus, the communication channels that are used by the computers is exposed to unauthorized access. Hence it has become necessary to secure such data. The obvious defence would be physical security (placing the machine protected behind physical walls). However, due to cost and efficiency issues, physical security is not always a feasible solution.

This led to a branch of developing virtual security methods for securing the data from adversaries called as Cryptography. The word cryptography was derived from the Greek word Kryptos, which is used to define anything that is hidden, obscure, secret or mysterious. Cryptography as defined by Yamen Akdeniz is "the science and study of secret writing" that concerns the different ways in which data and communication can be encoded to prevent their contents from being disclosed through various techniques like interception of message or eavesdropping. Different ways include using ciphers, codes, substitution, etc. so that only the authorized people can view and interpret the real message correctly.

Cryptography concerns itself with four main objectives, namely,

1) Confidentiality,

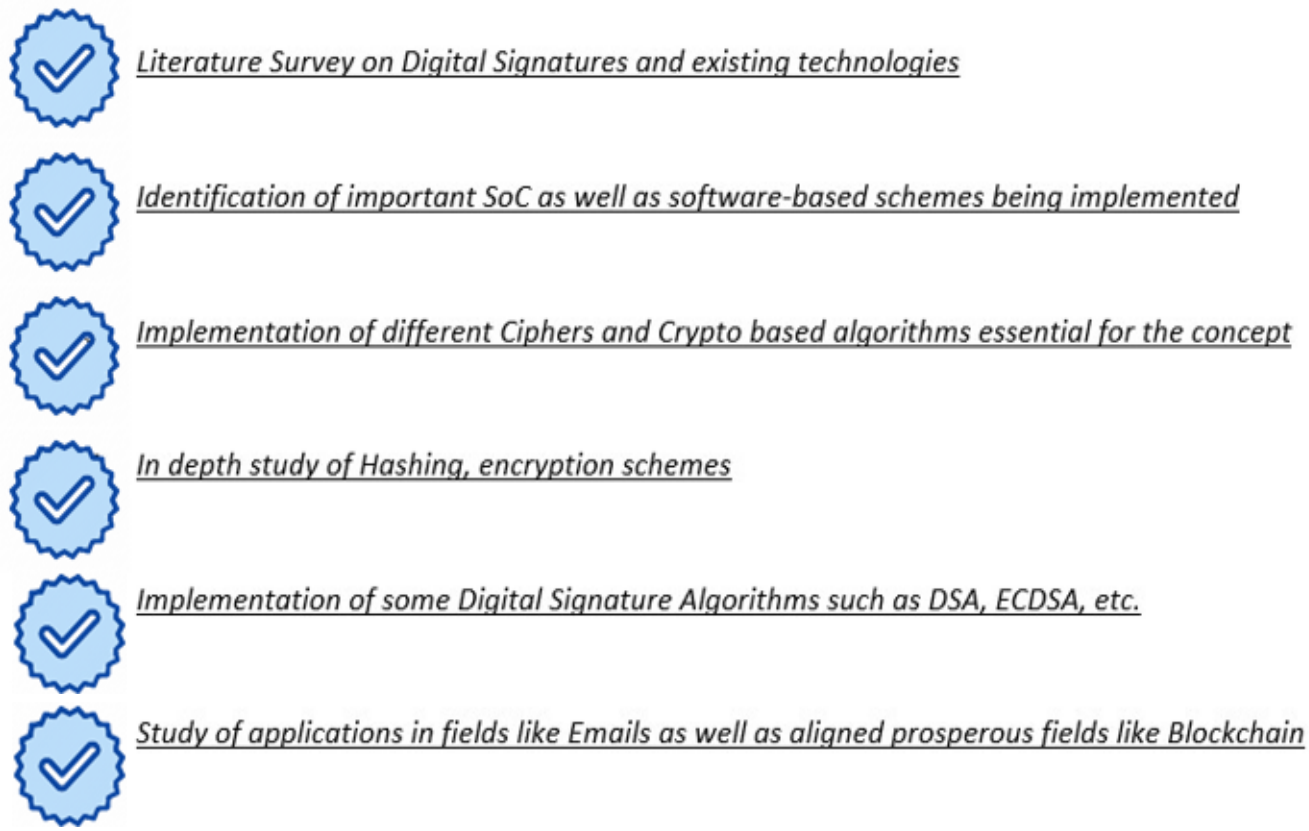
2) Integrity,

3) Non-repudiation and

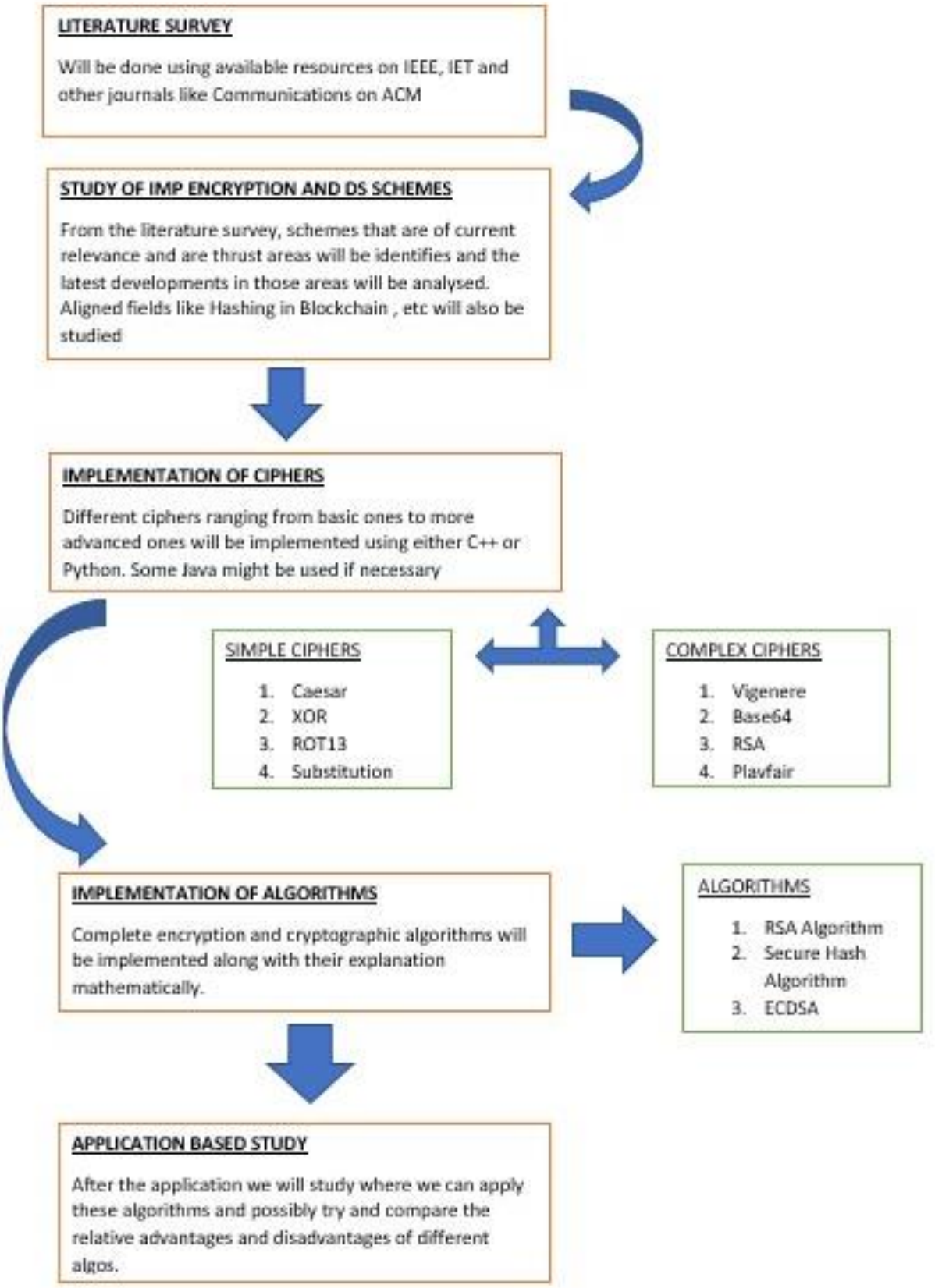
4) Authentication.

Cryptography is divided into two types, Symmetric key and Asymmetric key cryptography. In Symmetric key cryptography a single key is shared between sender and receiver. The sender uses the shared key and encryption algorithm to encrypt the message. The receiver uses the shared key and decryption algorithm to decrypt the message. In Asymmetric key cryptography each user is assigned a pair of keys, a public key and a private key. The public key is announced to all members while the private key is kept secret by the user. The sender uses the public key which was announced by the receiver to encrypt the message. The receiver uses his own private key to decrypt the message. What is a cipher? The method of encrypting any text in order to conceal its meaning and readability is called cipher. It is derived from the Arabic word sifr, meaning zero or empty. In addition to ciphers meaning in the context of cryptography, it also means a combination of symbolic letters as in letters for a monogram which is a symbol made by combining two or more letters. Cryptanalysis refers to the study of ciphers, cipher text, or cryptosystems with a view to finding weaknesses in them that will permit retrieval of the plaintext from the cipher text, without necessarily knowing the key or the algorithm. This is known as breaking the cipher, cipher text, or cryptosystem. The word breaking the cipher can be interchangeably used with the term weakening the cipher. That is to find a property or fault in the design of the encryption algorithm which would help the attacker to reduce the number of keys that he should try while performing brute force attack to break the code. For example, consider a symmetric key algorithm that uses a key of length 2^128 bits which implies that a brute force attack would require the attacker to try all 2^128 possible combinations to be certain of finding the correct key to convert the cipher text into plaintext, which is not possible since it will take thousands of years to try out each and every key. However, a cryptanalysis of the cipher text reveals a method that would allow the plaintext to be found in 2^20 rounds. While it is yet not broken, it is now much weaker and the plaintext can be found with comparatively smaller number of tries.

OBJECTIVES



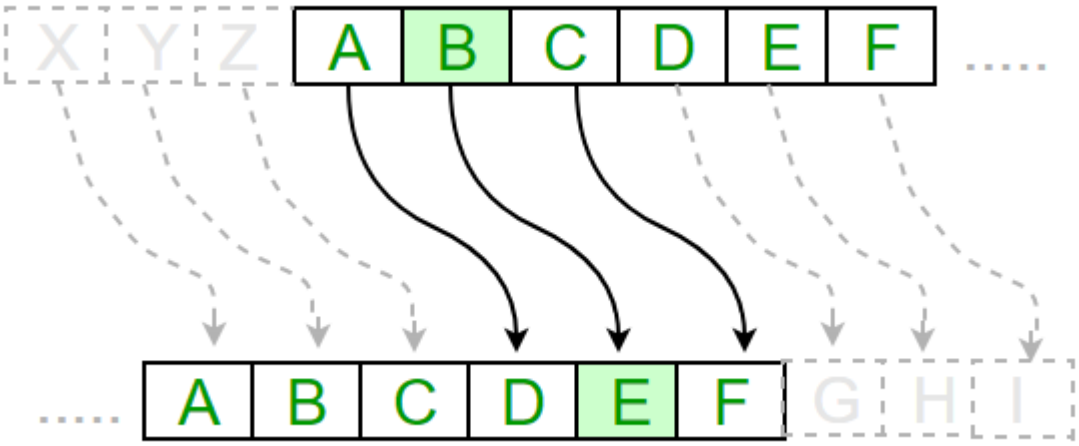
METHODOLOGY



CIPHERS AND ALGORITHMS PERUSED AND IMPLEMENTED

1. CAESAR CIPHER

The Caesar cipher is named after Julius Caesar, who, according to Suetonius, used shift cipher with a constant left shift of 3 to encrypt important military messages during the war. Hence it is also known as shift cipher, Caesar's cipher or Caesar shift. It uses a substitution method to evolve the encrypted text. The Caesar cipher is one of the earliest known and simplest ciphers. It is a type of substitution cipher in which each letter in the plaintext is 'shifted' a certain number of places down the alphabet. For example, with a shift of 1, A would be replaced by B, B would become C, and so on.



Consider an Example,

Plain text: ZYXWVUTSRQPONMLKJIHGFEDCBA

Cipher text: WVUTSRQPONMLKJIHGFEDCBAZYX

When encrypting, an individual looks up each letter of the text message in the "plain text" and writes down the corresponding letter in the "cipher text". Deciphering is done in exactly reverse manner, with a right shift of 3. This could also be represented using modular arithmetic by transforming the letters into numbers, as per the scheme, a→0, b→1, c→2 … x→23, y→24, z→25.

Now, if a letter (x) is to be encrypted, it is expressed as: En (x) = (x + n) mod 26. Decryption is performed similarly: Dn (x) = (x – n) mod 26. The replacement is same for entire text to be encrypted; thus, Caesar cipher is classified as monoalphabetic substitution. The major drawbacks of Caesar cipher is that it can easily be broken, even in cipher-text only scenario.

More complex encryption schemes such as the Vigenère cipher employ the Caesar cipher as one element of the encryption process. The widely known ROT13 'encryption' is simply a Caesar cipher with an offset of 13. The Caesar cipher offers essentially no communication security, and it will be shown that it can be easily broken even by hand.

Another Example:

To pass an encrypted message from one person to another, it is first necessary that both parties have the 'key' for the cipher, so that the sender may encrypt it and the receiver may decrypt it. For the caesar cipher, the key is the number of characters to shift the cipher alphabet.

Here is a quick example of the encryption and decryption steps involved with the caesar cipher. The text we will encrypt is 'defend the east wall of the castle', with a shift (key) of 1.

Plaintext: defend the east wall of the castle

Ciphertext: efgfoe uif fbtu xbmm pg uif dbtumf

It is easy to see how each character in the plaintext is shifted up the alphabet.

Decryption is just as easy, by using an offset of -1.

Plaintext: abcdefghijklmnopqrstuvwxyz

Ciphertext: bcdefghijklmnopqrstuvwxyza

Obviously, if a different key is used, the cipher alphabet will be shifted a different amount.

MATHEMATICAL MODELING OF CAESAR CIPHER

First we translate all of our characters to numbers, 'a'=0, 'b'=1, 'c'=2, ... , 'z'=25. We can now represent the caesar cipher encryption function, e(x), where x is the character we are encrypting, as:

$$e(x) = (x + k) \pmod{26}$$

Where k is the key (the shift) applied to each letter. After applying this function the result is a number which must then be translated back into a letter. The decryption function is :

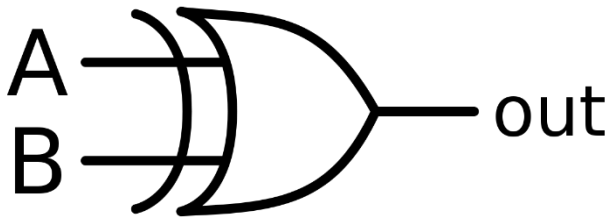
$$e(x) = (x - k) \pmod{26}$$

2. XOR ENCODER

Exclusive-OR encryption, while not a public-key system such as RSA, is almost unbreakable through brute force methods. It is susceptible to patterns, but this weakness can be avoided through first compressing the file (so as to remove patterns). Exclusive-or encryption requires that both encryptor and decryptor have access to the encryption key, but the encryption algorithm, while extremely simple, is nearly unbreakable.

Exclusive-OR encryption works by using the boolean algebra function exclusive-OR (XOR). XOR is a binary operator (meaning that it takes two arguments - similar to the addition sign, for example). By its name, exclusive-OR, it is easy to infer (correctly, no less) that it will return true if one, and only one, of the two operators is true. The truth table is as follows:

A	B	A XOR B
T	T	F
T	F	T
F	T	T
F	F	F



(A truth table works like a multiplication or addition table: the top row is one list of possible inputs; the side column is one list of possible inputs. The intersection of the rows and columns contains the result of the operation when done performed with the inputs from each row and column)

The idea behind exclusive-OR encryption is that it is impossible to reverse the operation without knowing the

initial value of one of the two arguments. For example, if you XOR two variables of unknown values, you cannot tell from the output what the values of those variables are. For example, if you take the operation A XOR B, and it returns TRUE, you cannot know whether A is FALSE and B is TRUE, or whether B is FALSE and A is TRUE. Furthermore, even if it returns FALSE, you cannot be certain if both were TRUE or if both were FALSE.

If, however, you know either A or B it is entirely reversible, unlike logical-AND and logical-OR. For exclusive-OR, if you perform the operation A XOR TRUE and it returns a value of TRUE you know A is FALSE, and if it returns FALSE, you know A is true. Exclusive-OR encryption works on the principle that if you have the encrypted string and the encryption key you can always decrypt correctly. If you don't have the key, it is impossible to decrypt it without making entirely random keys and attempting each one of them until the decryption program's output is something akin to readable text. The longer you make the encryption key, the more difficult it becomes to break it.

The actual way exclusive-OR encryption is used is to take the key and encrypt a file by repeatedly applying the key to successive segments of the file and storing the output. The output will be the equivalent of an entirely random program, as the key is generated randomly. Once a second person has access to the key, that person is able to decrypt the files, but without it, decryption is almost impossible. For every bit added to the length of the key, you double the number of tries it will take to break the encryption through brute force.

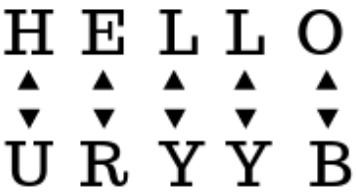
3. ROT13 CIPHER

Shift ciphers are also known as Caesar Ciphers. That's because the first recorded case of the shift cipher being used was by Julius Caesar. In his private correspondence, Julius Caesar would use a 3-letter shift to make his messages more difficult for prying eyes to read. ROT13 itself is much more modern. It rose to prominence in the 1980's when it was used in the Jokes newsgroup in the early days of the internet. It was used then to hide offensive jokes, letting a person choose if they wanted to view the content or not. It went on to become a simple way to hide the solution to online puzzle games and was often described as the encryption equivalent of turning a piece of paper upside down to read the answers. The treasure hunting website, Geocaching.com, uses encrypted hints to the locations of geocaches using ROT13. Although ROT13 is not a secure cipher, it has been used in some commercial applications. In 1999 it was discovered that Netscape Communicator was using the cipher to encrypt passwords. Two years later, the eBook vendor NPRG were revealed to be using ROT13 to encrypt their content. These aren't the only examples, though. Windows XP used ROT13 to encrypt registry entries. The UNIX fortune program also used the cipher to hide potentially offensive material.

What makes ROT13 unique is that it is its own inverse. Because the alphabet is 26 letters, and the shift is 13 letters, A translates to N and vice versa. However, it doesn't encode numbers or punctuation, which gives it some limitations.

ROT13 is easy to translate without any tools. If you think might be looking at a piece of ROT13 code, all you need to do is to write the letters A-M on a piece of paper, and the letters N to Z below them. You can then substitute the letters accordingly, so if the cipher text has a letter A, the plain text is N and vice versa.

For example, HELLO, would be translated to URYYB:



EXAMPLE

If you come across some code and you're not sure if you are looking at ROT13 or another, similar cipher you can use frequency analysis to identify the shift. Find the most commonly used character in the cipher text. The most commonly used letter in English is the letter E, so if your most frequently used character is R, you're probably looking at ROT13.

APPLICATIONS OF ROT13

Because there is no key needed to encrypt ROT13, it's not used for serious commercial purposes. In fact, ROT13 has become something of a joke in terms of its effectiveness. It's not uncommon to hear security experts deride insecure solutions by saying, 'That's about as much use as ROT13!'

On a similar note, you may hear techies talking about ROT26 or double ROT13/2ROT13. 'It's like ROT13, but twice as secure!' If you don't get why this is a joke, think about writing the letters A-Z on the top row, and again on the bottom. In other words, this sentence is already encrypted in ROT26. Congratulations on cracking the code.

ROT13 does still have its uses, however. It continues to be used to hide spoilers or NSFW content on forums and the like. It's also used to encode email addresses, to keep them from the grips of the spam bots.

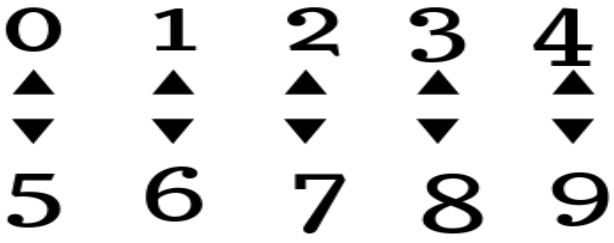
As a mental exercise, ROT13 can be used for letter games. You can look for words which, when encrypted, spell out other valid words. Examples of these are Gnat and Tang, and Abjurer/Nowhere.

Perhaps the most extreme version of this wordplay is Brian Westley's entry to the 1989 International Obfuscated C Code Contest. Brian wrote a piece of C code which would compile either as written, or when encrypted using ROT13. The icing on the cake is that the code itself encrypts and decrypts ROT13.

If you've ever been in a newgroup or internet forum and seen users exclaim, 'Fuurfu!' then you've been reading ROT13. This all started when someone claimed the word, 'Sheesh!' was being overused. This led to some wise cracker using ROT13 to encrypt the exclamation, and the idea caught on.

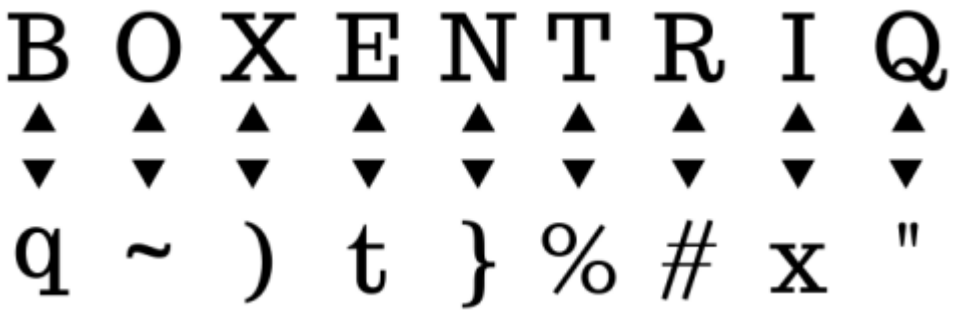
VARIATIONS OF ROT13

ROT5 is the numeric equivalent of ROT13. The numbers 0-4 are written on the top row, and 5-9 on the bottom. This allows numbers to be encrypted in the same manner as ROT13, by using their inverse. Using ROT13 and ROT5 together is sometimes called ROT18.



ROT5 TRANSLATION

Another alternative is ROT47, which uses a similar principle but encrypts all the visible ASCII characters allowing letters, numbers and punctuation to be encoded. It uses ASCII characters 33-126, which covers most of the characters available on a QWERTY keyboard.



4. SUBSTITUTION CIPHERS

Substitution ciphers encrypt the plaintext by swapping each letter or symbol in the plaintext by a different symbol as directed by the key. Perhaps the simplest substitution cipher is the Caesar cipher, named after the man who used it. To modern readers, the Caesar cipher is perhaps better known through the Captain Midnight Code-O-Graph and secret decoder rings that even came inside Kix cereal boxes. Technically speaking, the Caesar cipher may be differentiated from other, more complex substitution ciphers by terming it either a shift cipher or a mono-alphabetic cipher; both are correct.

Let's take a look at an example. Since case does not matter for the cipher, we can use the convention that plaintext is represented in lowercase letters, and ciphertext in uppercase. Spaces in the ciphertext are just added for readability; they would be removed in a real application of the cipher to make attacking the ciphertext more difficult.

Plaintext: speak, friend, and enter

Key: E

Ciphertext: WTIEOD JVMIRHD ERH IRXIV

This cipher's method of combining the plaintext and the key is actually addition. Each letter of the alphabet is assigned a number—that is, A is 0, B is 1, and so on, through Z at 25. The set of letters used can be more complex. This example also uses the comma character as the final character of the alphabet, 26. The spaces in the plaintext are ignored, for now. For each letter in the plaintext, it is converted to its number, then the value for the key is added, and the resulting number is converted back to a letter: S is 18 and E is 4. So, the result is 22, or W. This is repeated for each character in the plaintext. Decryption is simple—the inverse of addition is just subtraction, so the key is subtracted from the ciphertext to get the plaintext back. Of course, 22−4=18.

ISSUES AND DEVELOPMENTS

There are obviously lots of problems with this. To decrypt the message, one could quickly try all 26 keys. The number of possible keys is called the *key space*. If the key space is small enough that an adversary can try all possible keys in a "short" amount of time, then it doesn't matter what the algorithm is, it is essentially useless. This is known as the *sufficient key space principle*. "Short" is in quotes because the exact

length of time depends on the use of the key in the cryptosystem and the risk model that the defender has for how long the communication needs to be secret. However, if the adversary can try all of the keys in a day or a week, then the key space is generally too small for general commercial use. On modern computer systems, about 2^{80} keys can be tried in a "short" amount of time, so any algorithm employed by the defender to resist attack should have a key space at least this large. However, if the defender does not want to have to change the cipher relatively soon, we suggest a rather larger key space, and so does NIST (National Institute of Standards and Technology).

In this simple shift cipher, the key space is small. The best case for a mono-alphabetic cipher does not have a small key space, however. If A is randomly assigned to one of the 26 letters, B one of the remaining 25, C to one of the remaining 24, and so on, we create a table for the key that looks like this:

Plaintext character: 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

Key character: X F Q G A W Z S E D C V B N M L K J H G T Y U I O P

This is called a mono-alphabetic substitution cipher. For this cipher, there is no equivalent addition for encrypting the plaintext. The key is the whole table, and each letter is substituted by the key character. Decryption uses the same key, but you look up the ciphertext character on the bottom row and substitute the top-row character. The previous plaintext, "speak, friend, and enter," becomes HLAXCWJEANGXNGANGAJ, ignoring commas and spaces. The whole key space is quite large. There are 26×25×24×23×…×2×1 possible keys. This is written as 26!, read "twenty-six factorial." 26! is about equal to 2^{88}, which is large enough to resist brute-force attacks that try all the possible keys; that is, it satisfies the sufficient key space principle. But that does not mean the algorithm resists all attempts to subvert it.

The mono-alphabetic cipher is subject to frequency attacks or guessing. The ciphertext has just as many 'A' characters as there are 'e' characters in the plaintext. Anyone trying to attack the ciphertext could use a table of the frequency of letters in the English language to make some smart guesses about which ciphertext characters are which plaintext characters. This succeeds relatively easily. Humans can do it, rather slowly, once they have about 10 words, sometimes less. This is a relatively common puzzle in newspapers, so it should not be surprising it's easy to break. Computers can also do it reliably when they have at least 150 characters.

Frequency attacks are not limited to single letters. The problem applies to modern systems as well. If a bank begins every transaction with the same 10 characters, then an adversary would rightfully guess that that string is more frequent. Modern algorithms try to be robust against this in a variety of ways, which will be discussed later. However, sometimes the best course of action for the defender to resist such

frequency attacks is for the defender to modify the contents of the actual message, before encryption, to remove these regularities. If that is not possible, regularities in the plaintext should be minimized.

One method of frustrating frequency attacks on the underlying plaintext is to increase the block size of the cipher. The block size is how many units (in our example characters) are encrypted at once. Both the Caesar cipher and the mono-alphabetic substitution have a block size of one—only one character is encrypted at a time. A different defence is to use a key that changes per element of plaintext, whether or not the block size increases. The number of changes in the key per element of plaintext before the key repeats is called the *period of the key*; both preceding cipher examples have a key period of 1 as well as a block size of 1.

5. RSA CIPHER ALGORITHM

The RSA algorithm is an asymmetric cryptography algorithm; this means that it uses a *public* key and a *private* key (i.e two different, mathematically linked keys). As their names suggest, a public key is shared publicly, while a private key is secret and must not be shared with anyone.The RSA algorithm is named after those who invented it in 1978: Ron Rivest, Adi Shamir, and Leonard Adleman.

WORKING

The RSA algorithm ensures that the keys, in the above illustration, are as secure as possible. The following steps highlight how it works:

KEY GENERATION

1. Select two large prime numbers, x*x* and y*y*. The prime numbers need to be large so that they will be difficult for someone to figure out.
2. Calculate n = x * y*n*=*x*∗*y*.
3. Calculate the ***totient*** function; \phi(n) = (x-1)(y-1)$\phi(n)$=(*x*−1)(*y*−1).
4. Select an integer e*e*, such that e*e* is ***co-prime*** to \phi(n)$\phi(n)$ and 1 < e < \phi(n)1<*e*<$\phi(n)$. The pair of numbers (n,e)(*n*,*e*) makes up the public key.

Note: Two integers are co-prime if the only positive integer that divides them is 1.

5. Calculate d*d* such that e.d = 1*e*.*d*=1 mod*mod* \phi(n)$\phi(n)$.

d*d* can be found using the ***extended euclidean algorithm***. The pair (n,d)(*n*,*d*) makes up the private key.

ENCRYPTION

Given a plaintext P*P*, represented as a number, the ciphertext C*C* is calculated as:

C = P^{e}*C*=*Pe* mod*mod* n*n*.

DECRYPTION

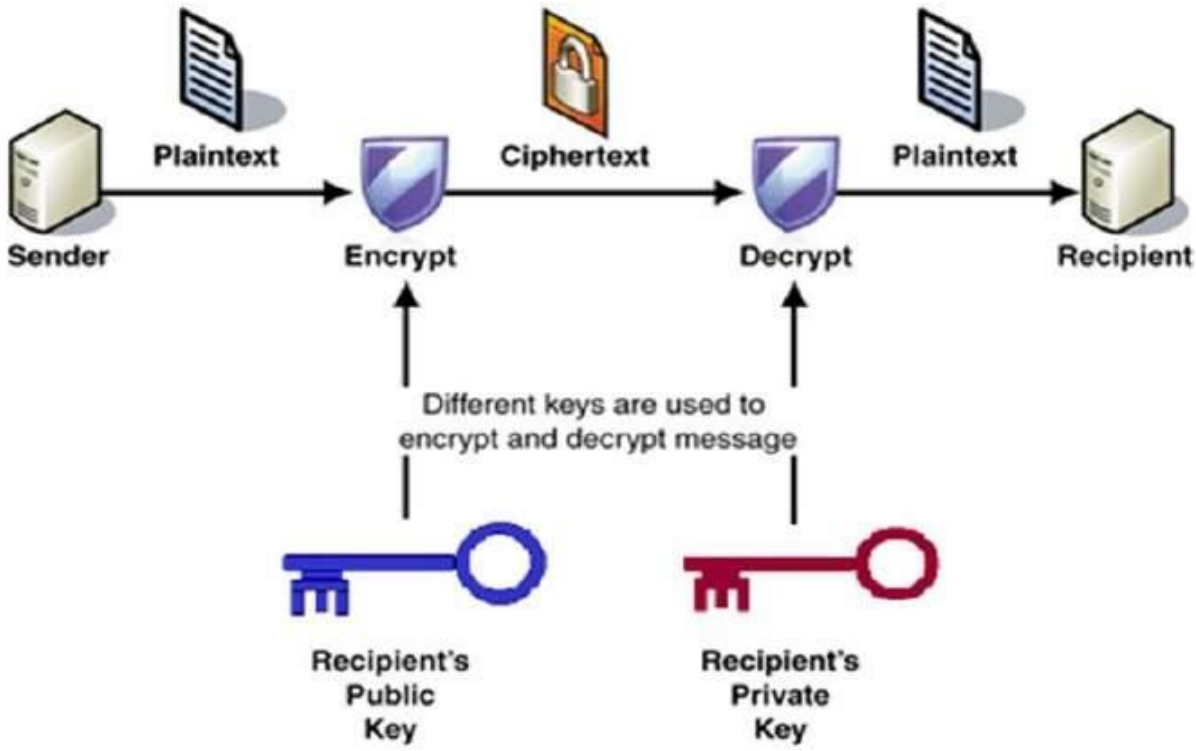
Using the private key (n,d)(*n*,*d*), the plaintext can be found using:

P = C^{d}*P*=*Cd* mod*mod* n*n*.

HISTORY OF THE ALGORITHM

The RSA algorithm is the basis of a cryptosystem -- a suite of cryptographic algorithms that are used for specific security services or purposes -- which enables public key encryption and is widely used to secure sensitive data, particularly when it is being sent over an insecure network such as the internet.

RSA was first publicly described in 1977 by Ron Rivest, Adi Shamir and Leonard Adleman of the Massachusetts Institute of Technology, though the 1973 creation of a public key algorithm by British mathematician Clifford Cocks was kept classified by the U.K.'s GCHQ until 1997.



Public key cryptography, also known as asymmetric cryptography, uses two different but mathematically linked keys -- one public and one private. The public key can be shared with everyone, whereas the private key must be kept secret.

In RSA cryptography, both the public and the private keys can encrypt a message; the opposite key from the one used to encrypt a message is used to decrypt it. This attribute is one reason why RSA has become the most widely used asymmetric algorithm: It provides a method to assure the confidentiality, integrity, authenticity, and non-repudiation of electronic communications and data storage.

Many protocols like secure shell, OpenPGP, S/MIME, and SSL/TLS rely on RSA for encryption and digital signature functions. It is also used in software programs -- browsers are an obvious example, as they need to establish a secure connection over an insecure network, like the internet, or validate a digital signature. RSA signature verification is one of the most commonly performed operations in network-connected systems.

USAGE OF RSA ALGORITHM

RSA derives its security from the difficulty of factoring large integers that are the product of two large prime numbers. Multiplying these two numbers is easy, but determining the original prime numbers from the total -- or factoring -- is considered infeasible due to the time it would take using even today's supercomputers.

The public and private key generation algorithm is the most complex part of RSA cryptography. Two large prime numbers, p and q, are generated using the Rabin-Miller primality test algorithm. A modulus, n, is calculated by multiplying p and q. This number is used by both the public and private keys and provides the link between them. Its length, usually expressed in bits, is called the key length.

The public key consists of the modulus n and a public exponent, e, which is normally set at 65537, as it's a prime number that is not too large. The e figure doesn't have to be a secretly selected prime number, as the public key is shared with everyone.

The private key consists of the modulus n and the private exponent d, which is calculated using the Extended Euclidean algorithm to find the multiplicative inverse with respect to the totient of n.

Read on or watch the video below for a more detailed explanation of how the RSA algorithm works.

AN EXAMPLE

Alice generates her RSA keys by selecting two primes: p=11 and q=13. The modulus is n=p×q=143. The totient is n $\phi(n)=(p-1)x(q-1)=120$. She chooses 7 for her RSA public key e and calculates her RSA private key using the Extended Euclidean algorithm, which gives her 103.

Bob wants to send Alice an encrypted message, M, so he obtains her RSA public key (n, e) which, in this example, is (143, 7). His plaintext message is just the number 9 and is encrypted into ciphertext, C, as follows:

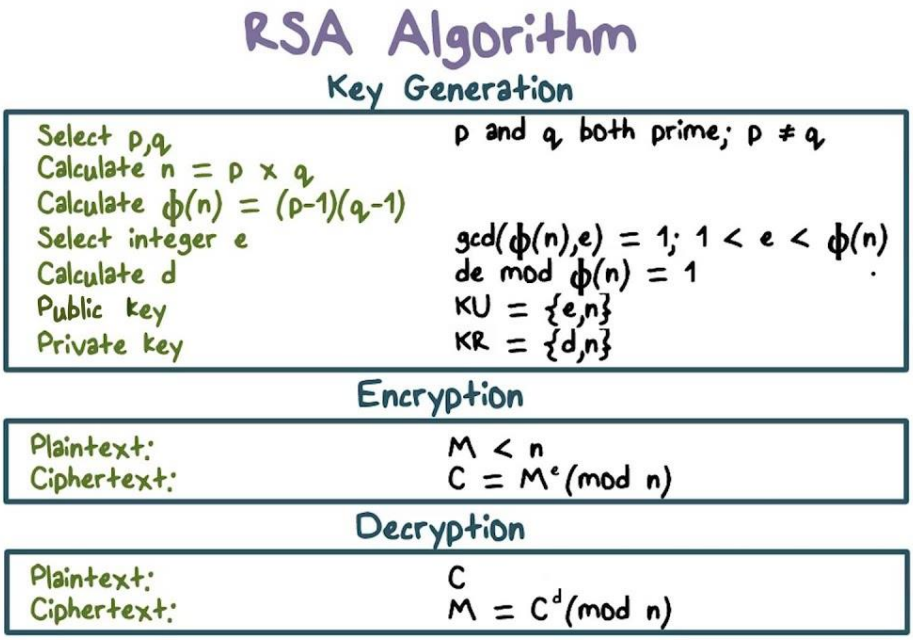
$M^e \bmod n = 9^7 \bmod 143 = 48 = C$

When Alice receives Bob's message, she decrypts it by using her RSA private key (d, n) as follows:

$C^d \bmod n = 48^{103} \bmod 143 = 9 = M$

To use RSA keys to digitally sign a message, Alice would need to create a hash -- a message digest of her message to Bob -- encrypt the hash value with her RSA private key, and add the key to the message. Bob can then verify that the message has been sent by Alice and has not been altered by decrypting the hash value with her public key. If this value matches the hash of the original message, then only Alice could have sent it -- authentication and non-repudiation -- and the message is exactly as she wrote it -- integrity.

Alice could, of course, encrypt her message with Bob's RSA public key -- confidentiality -- before sending it to Bob. A digital certificate contains information that identifies the certificate's owner and also contains the owner's public key. Certificates are signed by the certificate authority that issues them, and they can simplify the process of obtaining public keys and verifying the owner.



SECURITY OF RSA

RSA security relies on the computational difficulty of factoring large integers. As computing power increases and more efficient factoring algorithms are discovered, the ability to factor larger and larger numbers also increases.

6. PLAYFAIR CIPHER

In cryptosystems for manually encrypting units of plaintext made up of more than a single letter, only digraphs (pairs of letters) were ever used. By treating digraphs in the plaintext as units rather than as single letters, the extent to which the raw frequency distribution survives the encryption process can be lessened but not eliminated, as letter pairs are themselves highly correlated. The best-known digraph substitution cipher is the Playfair, invented in 1854 by Sir Charles Wheatstone but championed at the British Foreign Office by Lyon Playfair, the first Baron Playfair of St. Andrews. Below is an example of a Playfair cipher, solved by Lord Peter Wimsey in Dorothy L. Sayers's *Have His Carcase* (1932). Here, the mnemonic aid used to carry out the encryption is a 5 × 5-square matrix containing the letters of the alphabet (I and J are treated as the same letter). A key word, MONARCHY in this example, is filled in first, and the remaining unused letters of the alphabet are entered in their lexicographic order:

M	**O**	**N**	**A**	**R**
C	**H**	**Y**	B	D
E	F	G	I/J	K
L	P	Q	S	T
U	V	W	X	Z

Plaintext digraphs are encrypted with the matrix by first locating the two plaintext letters in the matrix. They are (1) in different rows and columns; (2) in the same row; (3) in the same column; or (4) alike. The corresponding encryption (replacement) rules are the following:

1. When the two letters are in different rows and columns, each is replaced by the letter that is in the same row but in the other column; i.e., to encrypt WE, W is replaced by U and E by G.
2. When A and R are in the same row, A is encrypted as R and R (reading the row cyclically) as M.
3. When I and S are in the same column, I is encrypted as S and S as X.
4. When a double letter occurs, a spurious symbol, say Q, is introduced so that the MM in SUMMER is encrypted as NL for MQ and CL for ME.
5. An X is appended to the end of the plaintext if necessary to give the plaintext an even number of letters.

Encrypting the familiar plaintext example using Sayers's Playfair array yields:

```
Plaintext: WE ARE DISCOVERED  SAVE YOURSELFX
Cipher:    UG RMK CSXHMUFMKB  TOXG CMVATLUIV
```

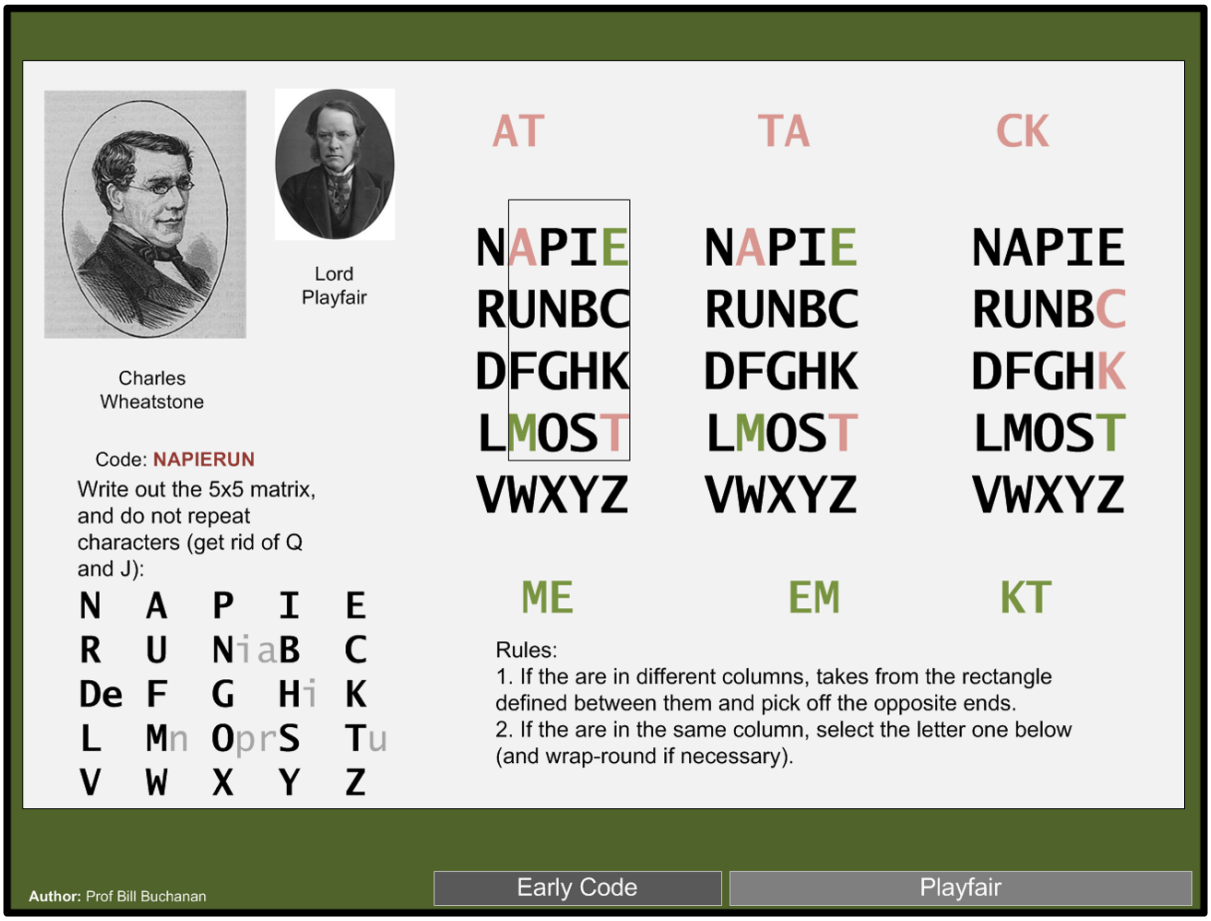
If the frequency distribution information were totally concealed in the encryption process, the ciphertext plot of letter frequencies in Playfair ciphers would be flat. It is not. The deviation from this ideal is a measure of the tendency of some letter pairs to occur more frequently than others and of the Playfair's row-and-column correlation of symbols in the ciphertext—the essential structure exploited by a

cryptanalyst in solving Playfair ciphers. The loss of a significant part of the plaintext frequency distribution, however, makes a Playfair cipher harder to cryptanalyze than a monoalphabetic cipher.

The Playfair Cipher was first described by Charles Wheatstone in 1854, and it was the first example of a **Digraph Substitution Cipher.** It is named after Lord Playfair, who heavily promoted the use of the cipher to the military.

When it was first put to the British Foreign Office as a cipher, it was rejected due to its perceived complexity. However, it was later adopted as a military cipher due to it being reasonably fast to use, and it requires no special equipment, whilst also providing a stronger cipher than a **Monoalphabetic Substitution Cipher**. It was used in the Second Boer War, and both World War I and World War II to different degrees. It is no longer used by military forces since the advent of powerful computers, but in its day it provided a relatively secure cipher which was easy to implement quite quickly.

ENCRYPTION OF PLAYFAIR CIPHER



In order to encrypt using the Playfair Cipher, we must first draw up a **Polybius Square** (but without the need for the number headings). This is usually done using a keyword, and either combining "i" and "j" or omitting "q" from the square.

We must now split the plaintext up into digraphs (that is pairs of letters). On each digraph we perform the following encryption steps:

1. If the digraph consists of the same letter twice (or there is only one letter left by itself at the end of the plaintext) then insert the letter "X" between the same letters (or at the end), and then continue with the rest of the steps.

2. If the two letters appear on the same row in the square, then replace each letter by the letter immediately to the right of it in the square (cycling round to the left-hand side if necessary).
3. If the two letters appear in the same column in the square, then replace each letter by the letter immediately below it in the square (cycling round to the top of the square if necessary).
4. Otherwise, form the rectangle for which the two plaintext letters are two opposite corners. Then replace each plaintext letter with the letter that forms the other corner of the rectangle that lies on the same ***row*** as that plaintext letter (being careful to maintain the order).

As an example, we shall encrypt the plaintext "hide the gold in the tree stump" using the key phrase *playfair example*. Firstly, we must generate the **Polybius Square** that we are going to use. We do this by setting out a 5x5 grid, and filling it with the alphabet, starting with the letters of the key phrase, and ignoring any letters we already have in the square. We are also going to combine "I" and "J" in the square.

We must now split the plaintext into digraphs. At this point it is a good idea to apply Rule 1, and split up any double letter digraphs by inserting an "x" between them. The first image below shows the initial digraph split of the plaintext, and the second image displays how we split up the "ee" into "ex" and "es". In this case, when we insert this extra "x", we no longer need to have one at the end of the plaintext.

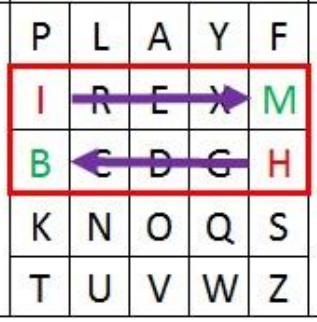
hi	de	th	eg	ol	di	nt	he	tr	ee	st	um	p

The initial split into digraphs.

hi	de	th	eg	ol	di	nt	he	tr	ex	es	tu	mp

The digraph split once we apply Rule 1, and remove any digraphs made from two of the same letter.

We now take each digraph in turn and apply rule 2, 3 or 4 as necessary. Each step is show below with a visual representation of what is done for each digraph.

Plaintext Digraph	Square	Rule	Ciphertext Digraph
hi		Rule 4: Rectangle	BM

We can now take each of the ciphertext digraphs that we produced and put them all together.

BM	OD	ZB	XD	NA	BE	KU	DM	UI	XM	MO	UV	IF

The ciphertext digraphs

We can now write out the ciphertext as a long string "BMODZBXDNABEKUDMUIXMMOUVIF" or split it into block of 5 "BMODZ BXDNA BEKUD MUIXM MOUVI F" or even give it the same layout as the original "BMOD ZBX DNAB EK UDM UIXMM OUVIF"

Decryption is nearly identical to the encryption process, except for rules 2 and 3 we must take the letters to the left and above respectively. Also, we remove any extra "X" in the decrypted text to reveal the final plaintext.

Split the ciphertext into digraphs. There is no need to add any "X" in the decryption process as these will be revealed as we decrypt.

UA	AR	BE	DE	XA	PO	PR	QN	XA	XA	NR

The ciphertext split into digraphs.

Now we apply the rules as needed to each digraph in the ciphertext.

Ciphertext Digraph	Square	Rule	Plaintext Digraph
UA	E A M P L B C D F G H I K N O Q R S T U W Y Z	Rule 4: Rectangle	we

We now combine all the digraphs together.

we	wi	lx	lm	ex	et	at	th	ex	ex	it

The plaintext digraphs.

So, we get the message "we wilxl mexet at thex exit". When we remove the unnecessary "x"s we get a final plaintext of "we will meet at the exit". Note that we cannot just remove all the "x"s as one is part of the word "exit".

IMPACT OF THE PLAYFAIR CIPHER

The Playfair Cipher was an ingenious new way to encipher messages. It was the first of its kind, and opened up the world of cryptography to a whole new type of cipher: the polygraphic cipher. Although not secure in terms of modern cryptography, it was a substantial improvement over Monoalphabetic Substitution Ciphers, and significantly easier to use in the field than Polyalphabetic Substitution Ciphers.

We can see in the decryption example above that there are three digraphs the same in the ciphertext, namely "XA", and we also see that all three decrypts to the same plaintext "ex". This shows us that Digraph Substitution Ciphers are still susceptible to Frequency Analysis, but it has to be done on pairs of letters.

Because it is done on pairs of letters, this Frequency Analysis is significantly harder to crack. Firstly, for a monoalphabetic cipher we have 26 possible letters to check. For a general Digraph Cipher, we have 26 x

26 = 676 possible pairings we need to check in our frequency analysis. In the instance of the Playfair Cipher, we cannot encrypt to a double letter, so we remove the 26 possibilities of double letters, giving us 650 possible digraphs we need to check. By hand this task in monumental, but with the help of a computer, it can be done in a matter of seconds.

Another useful weakness of the Playfair Cipher that can be exploited in cryptanalysis is the fact that the same pair of letters reversed will produce the same pair of letters reversed. For example, if the plaintext "er" encrypts to "HY", then the plaintext "re" will encrypt rto "YH". This is useful in some words in english such as "departed" which start and end with the same pair of letters in reverse.

7. VIGENERE CIPHER

One of the main problems with simple substitution ciphers is that they are so vulnerable to frequency analysis. Given a sufficiently large ciphertext, it can easily be broken by mapping the frequency of its letters to the know frequencies of, say, English text. Therefore, to make ciphers more secure, cryptographers have long been interested in developing enciphering techniques that are immune to frequency analysis. One of the most common approaches is to suppress the normal frequency data by using more than one alphabet to encrypt the message. A *polyalphabetic substitution cipher* involves the use of two or more cipher alphabets. Instead of there being a one-to-one relationship between each letter and its substitute, there is a one-to-many relationship between each letter and its substitutes.

THE TABLEAU

The *Vigenere Cipher ,* proposed by Blaise de Vigenere from the court of Henry III of France in the sixteenth century, is a polyalphabetic substitution based on the following *tableau:*

KEY \ -- PLAINTEXT --	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
A	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
B	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	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V

Note that each row of the table corresponds to a Caesar Cipher. The first row is a shift of 0; the second is a shift of 1; and the last is a shift of 25.

The Vigenere cipher uses this table together with a keyword to encipher a message. For example, suppose we wish to encipher the plaintext message:

TO BE OR NOT TO BE THAT IS THE QUESTION

using the keyword RELATIONS. We begin by writing the keyword, repeated as many times as necessary, above the plaintext message. To derive the ciphertext using the tableau, for each letter in the plaintext, one finds the intersection of the row given by the corresponding keyword letter and the column given by the plaintext letter itself to pick out the ciphertext letter.

Keyword:	RELAT IONSR ELATI ONSRE LATIO NSREL
Plaintext:	TOBEO RNOTT OBETH ATIST HEQUE STION
Ciphertext:	KSMEH ZBBLK SMEMP OGAJX SEJCS FLZSY

Decipherment of an encrypted message is equally straightforward. One writes the keyword repeatedly above the message:

Keyword:	RELAT IONSR ELATI ONSRE LATIO NSREL
Ciphertext:	KSMEH ZBBLK SMEMP OGAJX SEJCS FLZSY
Plaintext:	TOBEO RNOTT OBETH ATIST HEQUE STION

This time one uses the keyword letter to pick a column of the table and then traces down the column to the row containing the ciphertext letter. The index of that row is the plaintext letter.

The strength of the Vigenere cipher against frequency analysis can be seen by examining the above ciphertext. Note that there are 7 'T's in the plaintext message and that they have been encrypted by 'H,' 'L,' 'K,' 'M,' 'G,' 'X,' and 'L' respectively. This successfully masks the frequency characteristics of the English 'T.' One way of looking at this is to notice that each letter of our keyword RELATIONS picks out 1 of the 26 possible substitution alphabets given in the Vigenere tableau. Thus, any message encrypted by a Vigenere cipher is a collection of as many simple substitution ciphers as there are letters in the keyword.

Although the Vigenere cipher has all the features of a useful field cipher -- i.e., easily transportable key and tableau, requires no special apparatus, easy to apply, etc. -- it did not catch on its day. A variation of it, known as the Gronsfeld cipher, did catch on in Germany and was widely used in Central Europe. The Gronsfeld variant used the digits of a key number instead of a the letters of keyword, but remained unchanged in all other respects. So in fact the Gronsfeld is a weaker technique than Vigenere since it only uses 10 substitute alphabets (one per digit 0..9) instead of the 26 used by Vigenere.

DECRYPTION OF THE CIPHER

Vigenere-like substitution ciphers were regarded by many as practically unbreakable for 300 years. In 1863, a Prussian major named Kasiski proposed a method for breaking a Vigenere cipher that consisted of finding the length of the keyword and then dividing the message into that many simple substitution cryptograms. Frequency analysis could then be used to solve the resulting simple substitutions.

Kasiski's technique for finding the length of the keyword was based on measuring the distance between repeated bigrams in the ciphertext. Note that in the above cryptogram the plaintext bigram 'TO' occurs twice in the message at position 0 and 9 and in both cases it lines up perfectly with the first two letters of the keyword. Because of this it produces the same ciphertext bigram, 'KS.' The same can be said of plaintext 'BE' which occurs twice starting at positions 2 and 11, and also is encrypted with the same

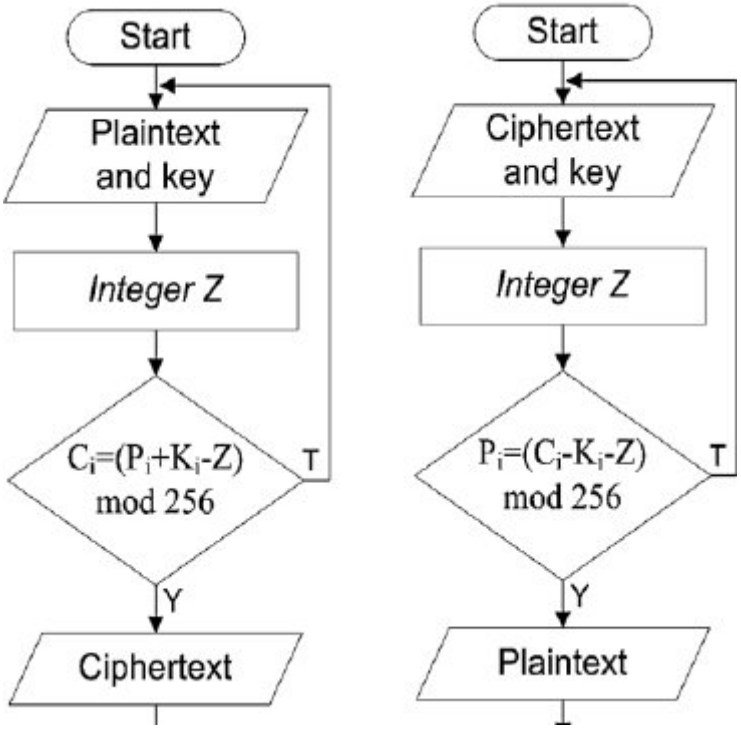
ciphertext bigram, 'ME.' In fact, any message encrypted with a Vigenere cipher will produce many such repeated bigrams. Although not every repeated bigram will be the result of the encryption of the same plaintext bigram, many will, and this provides the basis for breaking the cipher. By measuring and factoring the distances between recurring bigrams -- in this case the distance is 9 -- Kasiski was able to guess the length of the keyword. For this example,

Location	01234 56789 01234 56789 01234 56789
Keyword:	RELAT IONSR ELATI ONSRE LATIO NSREL
Plaintext:	TOBEO RNOTT OBETH ATIST HEQUE STION
Ciphertext:	KSMEH ZBBLK SMEMP OGAJX SEJCS FLZSY

the Kasiski method would create something like the following list:

Repeated Bigram	Location	Distance	Factors
KS	9	9	3, 9
SM	10	9	3, 9
ME	11	9	3, 9

Factoring the distances between repeated bigrams is a way of identifying possible keyword lengths, with those factors that occur most frequently being the best candidates for the length of the keyword. Note that in this example since 3 is also a factor of 9 (and any of its multiples) both 3 and 9 would be reasonable candidates for keyword length. Although in this example we don't have a clear favorite, we've narrowed down the possibilities to a very small list. Note also that if a longer ciphertext were encrypted with the same keyword ('RELATIONS'), we would expect to find repeated bigrams at multiples of 9 --



i.e., 18, 27, 81, etc. These would also have 3 as a factor. Kasiski's important contribution is to note this phenomenon of repeated bigrams and propose a method -- factoring of distances -- to analyze it.

Once the length of the keyword is known, the ciphertext can be broken up into that many simple substitution cryptograms. That is, for a keyword of length 9, every 9-th letter in the ciphertext was encrypted with the same keyword letter. Given the structure of the Vigenere tableau, this is equivalent to using 9 distinct simple substitution ciphers, each of which was derived from 1 of the 26 possible Caesar shifts given in the tableau. The pure Kasiski method proceeds by analyzing these simple substitution cryptograms using frequency analysis and the other standard techniques. A variant of this method, proposed by the French cryptographer Kerckhoff, is based on discovering the keyword itself and then using it to decipher the cryptogram. In Kerckhoff's method, after the message has been separated into several columns, corresponding to the simple substitution cryptograms, one tallies the frequencies in each column and then uses frequency and logical analysis to construct the key. For example, suppose the most frequent letter in the first column is 'K'. We would hypothesize that 'K' corresponds to the English 'E'. If we consult the Vigenere tableau at this point, we can see that if English 'E' were enciphered into 'K' then row G of the table must have been the alphabet used for the first letter of the keyword. This implies that the first letter of the keyword is 'G'.

The problem with this "manual" approach is that for short messages there are often several good candidates for English 'E' in each column. This requires the testing of multiple hypotheses, which can get quite tedious and involved. Therefore, we need a more sensitive test to discover the alphabet used by each letter of the keyword. Recalling that each row of the Vigenere tableau is one of the 26 Caesar shifts, we can use the chi-square test to determine which of the 26 possible shifts was used for each letter of the keyword. This modern-day version of the Kerckhoff method turns out to be very effective.

8. SECURE HASH ALGORITHM

Secure Hash Algorithms (SHA) are used for computing a condensed representation of electronic data (message). When a message of any length less than 264 bits (for SHA-224 and SHA-256) or less than 2128 bits (for SHA-384, SHA-512, SHA-512/224 and SHA-512/256) is input to a hash algorithm, the result is an output called a *message digest.* Common names for the output of a hash function include also hash value, hash, and digital fingerprint. The SHA-3 hash functions can be implemented as alternatives to the SHA-2 functions, or vice versa.

Algorithm	Message Size (bits)	Message Digest Size (bits)
SHA-1	$<2^{64}$	160
SHA-224	$<2^{64}$	224
SHA-256	$<2^{64}$	256
SHA-384	$<2^{128}$	384
SHA-512	$<2^{128}$	512
SHA-512/224	$<2^{128}$	224
SHA-512/256	$<2^{128}$	256

SHA-1 or Secure Hash Algorithm 1 is a cryptographic hash function which takes an input and produces a 160-bit (20-byte) hash value. This hash value is known as a message digest. This message digest is usually then rendered as a hexadecimal number which is 40 digits long. It is a U.S. Federal Information Processing Standard and was designed by the United States National Security Agency.

SHA-1 is now considered insecure since 2005. Major tech giants' browsers like Microsoft, Google, Apple and Mozilla have stopped accepting SHA-1 SSL certificates by 2017.

In simple words, SHA-256 (Secure Hash Algorithm, FIPS 182-2), is one of the cryptographic hash function which has digest length of 256 bits. It's a keyless hash function, means an MDC (Manipulation Detection Code). In other words, SHA (Secure Hash Algorithm) was developed by the National Institute of Standards & Technology, and further, they came with a new version called SHA-256 (the SHA-2 family), where the number is represented as the hash length in bits.

SHA-256 is one of the six variants of SHA family (SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256). These variants differ in size of their output, internal state size, block size, message size & rounds.

These days, all the major SSL Certificate issuers use SHA-256 which is more secure and trustworthy. Before SHA-1 was used, but it has been deprecated in January 2016, due to its security vulnerabilities it has become more susceptible to attacks and one of the reasons was a smaller bit size.

RESULTS

1. CAESAR CIPHER

```
Press 1: Encryption/Decryption; Press 2: quit: 1

Choose key value (choose a number between 1 to 26): 7

NOTE: Put LOWER CASE letters for encryption and
UPPER CASE letters for decryption

Enter cipher text (only alphabets) and press enter to continue: ILLUMINATI
beenfbgtmb

Press 1: Encryption/Decryption; Press 2: quit:
```

2. ROT13 CIPHER

```
71 }
72
                                                    input
XVATZNXRE
KINGMAKER


...Program finished with exit code 0
Press ENTER to exit console.
```

3. QWERTY SUBSTITUTION CIPHER

```
Enter text: abcdef
output: qwerty


...Program finished with exit code 0
Press ENTER to exit console.
```

4. VIGENERE CIPHER

```
THE PLAINTEXT: ANIRUDHNAKRA
THE KEY: GODGODGODGODGODGODGOD
THE CIPHERTEXT: GBLXIGNBDQFD
THE DECODED PLAINTEXT: ANIRUDHNAKRA


...Program finished with exit code 1
Press ENTER to exit console.
```

5. PLAYFAIR CIPHER

```
Put key value (put alphabets/words):
TOMMY
TOMMY
T       O       M       Y       A
B       C       D       E       F
G       H       I/J     K       L
N       P       Q       R       S
U       V       W       X       Z

Press 1: Encrypt | 2: Decrypt | 3: Quit
1
Put your text: ABHIJEET
TFIKiABY
Press 1: Encrypt | 2: Decrypt | 3: Quit
```

6. XOR ENCODING

```
Accepted cipher:
Anirudhisagod

Encrypted cipher:
S|{`gvz{asu}v

Decrypted cipher:
Anirudhisagod


...Program finished with exit code 0
Press ENTER to exit console.
```

7. RSA ALGORITHM

```
ENTER FIRST PRIME NUMBER
5

ENTER ANOTHER PRIME NUMBER
7

ENTER MESSAGE
hello

POSSIBLE VALUES OF e AND d ARE
11      11
13      13
17      17

THE ENCRYPTED MESSAGE IS
vjcco
THE DECRYPTED MESSAGE IS
hello

...Program finished with exit code 0
Press ENTER to exit console.
```

8. SHA-1 and SHA-256

```
{'sha512', 'sha256', 'sha1', 'sha224', 'md5', 'sha384'}
Object: <sha256 HASH object @ 0x7f180222a4e0>
Hexadecimal format: 20eab4fa057cc63161d37a5c6f5f646e33d9eadabae7d6cc63b93cc1f3bcce77


...Program finished with exit code 0
Press ENTER to exit console.
```

```
70c0a5e0ce5067cc40c67732b2de148ad263015d


...Program finished with exit code 0
Press ENTER to exit console.
```

CONCLUSION

We have studied the working and implemented different ciphers used for digital signature. Irrespective of the domain specific application of digital signatures, the primary focus is always over the implementation of authentication and integrity of data. The ever-increasing requirements of security will give rise to a new horizon for application of digital signatures using object-oriented modelling. This will lead to generation of more powerful and complex digital signature schemes which will be capable enough to fight against multiple types of attacks over the cryptosystem.

We have tried to analyze and study different digital signature algorithms such as DSA, RSA and ECDSA. First, we study these algorithms in theory, then implement them using c++ or python and finally, we draw our conclusions on a solution to the digital signature as to which algorithm is suitable for which example. For eg. RSA is recommended for signature verification on mobile devices, and ECDSA is recommended for generating keys and signatures on the device in mobile information systems. But there is still a great deal of research work on the digital signature to do.

To maintain the cost and computational efficiency of these cryptosystems with these increased complexities and real-world orientation, the application of the standard digital signature schemes like vigenere and elliptical curve schemes like ECDSA will become the primary focus of research in the near future.

REFERENCES

1. Sur C., Roy A., Banik S., A Study of the State of E-Governance in India, Proceedings of National Conference on Computing and Systems 2010 (NACCS 2010), January 29, 2010

2. Roy A., Sarkar S., Mukherjee J., Mukherjee A, Biometrics as an authentication technique in E-Governance security, Proceedings of UGC sponsored National Conference on "Research And Higher Education In Computer Science And Information Technology, RHECSIT-2012

3. Sarkar S., Roy A., A Study on Biometric based Authentication, Proceedings of Second National Conference on Computing and Systems - 2012 (NaCCS - 2012)

4. www2005.org/cdrom/docs/p412.pdf

5. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.59.5807

6. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5412852

7. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6022868

8. http://people.csail.mit.edu/rivest/Rsapaper.pdfDate of access -24th March (2012).

9. http://www.computer.org/portal/web/csdl/doi/10.1109/ICON.2000.875798

10. https://lirias.kuleuven.be/handle/123456789/60087

APPENDIX AND RELEVANT CODES

1. CAESAR CIPHER

```
#include <iostream>
#include <stdlib.h>
#include <string>
using namespace std;

char caesar( char c, int k )
{
    if( isalpha(c) && c != toupper(c))
    {
        c = toupper(c);
        c = (((c-65)+k) % 26) + 65;
    }
    else
    {
        c = ((((c-65)-k) + 26) % 26) + 65;
        c = tolower(c);
    }
    return c;
}

int main()
{
    string input, output;
    int choice = 0;

    while (choice != 2) {
        cout << endl << "Press 1: Encryption/Decryption; Press 2: quit: " ;

        try {
            cin >> choice;
```

```
            if (choice != 1 && choice != 2) throw "Incorrect Choice";
        }
        catch (const char* chc) {
            cerr << "INCORRECT CHOICE !!!!" << endl;
            return 1;
        }

        if (choice == 1) {
            int key;
            try {
                cout << endl << "Choose key value (choose a number between 1 to 26): ";
                cin >> key;
                cin.ignore();
                if (key < 1 || key > 26) throw "Incorrect key";
            }
            catch (const char* k) {
                cerr << "INCORRECT KEY VALUE CHOSEN !!!" << endl;
                return 1;
            }

            try {
                cout << endl << "NOTE: Put LOWER CASE letters for encryption and" << endl;
                cout << "UPPER CASE letters for decryption" << endl;
                cout << endl << "Enter cipher text (only alphabets) and press enter to continue: ";
                getline(cin, input);
```

```
out << endl << "NOTE: Put LOWER CASE letters for encryption and" << endl;
out << "UPPER CASE letters for decryption" << endl;
out << endl << "Enter cipher text (only alphabets) and press enter to continue: ";
etline(cin, input);

or (int i = 0; i < input.size(); i++) {
   if ((!(input[i] >= 'a' && input[i] <= 'z')) && (!(input[i] >= 'A' && input[i] <= 'Z'))) throw "Incorrect string";


 (const char* str) {
err << "YOUR STRING MAY HAVE DIGITS OR SPECIAL SYMBOLS !!!" << endl;
err << "PLEASE PUT ONLY ALPHABETS !!! " << endl;
eturn 1;


nsigned int x = 0; x < input.length(); x++) {
utput += caesar(input[x], key);


<< output << endl;
t.clear();
```

```
    try {
        cout << endl << "NOTE: Put LOWER CASE letters for encryption and" << endl;
        cout << "UPPER CASE letters for decryption" << endl;
        cout << endl << "Enter cipher text (only alphabets) and press enter to continue: ";
        getline(cin, input);

        for (int i = 0; i < input.size(); i++) {
            if ((!(input[i] >= 'a' && input[i] <= 'z')) && (!(input[i] >= 'A' && input[i] <= 'Z'))) throw
        }
    }
    catch (const char* str) {
        cerr << "YOUR STRING MAY HAVE DIGITS OR SPECIAL SYMBOLS !!!" << endl;
        cerr << "PLEASE PUT ONLY ALPHABETS !!! " << endl;
        return 1;
    }

    for(unsigned int x = 0; x < input.length(); x++) {
        output += caesar(input[x], key);
    }

    cout << output << endl;
    output.clear();
}
```

2. ROT13 CIPHER

main.cpp

```cpp

#include<bits/stdc++.h>
using namespace std;

map <char,int> dict1;

map <int,char> dict2;

void create_dict()
{
    for(int i = 1; i < 27; i++)
        dict1[char(64 + i)] = i;

    dict2[0] = 'Z';

    for(int i = 1; i < 26; i++)
        dict2[i] = char(64 + i);

    return;
}

string encrypt(string message, int shift)
{
    string cipher = "";
    for(int i = 0; i < message.size(); i++)
    {

        if(message[i] != ' ')
        {

            int num = (dict1[message[i]] + shift) % 26;


            cipher += dict2[num];
        }
        else
        {

            cipher += " ";
        }
    }
    return cipher;
}

string decrypt(string message, int shift)
{
    string decipher = "";
    for(int i = 0; i < message.size(); i++)
    {

        if(message[i] != ' ')
        {

```

```
59
60              int num = (dict1[message[i]] - shift + 26) % 26;
61
62              decipher += dict2[num];
63          }
64          else
65          {
66
67              decipher += " ";
68          }
69      }
70      return decipher;
71  }
72
73  int main()
74  {
75      create_dict();
76
77      string message = "KINGMAKER";
78      int shift = 13;
79
80      cout << encrypt(message, shift) << "\n";
81
82      message = "XVATZNXRE";
83      shift = 13;
84
85      cout << decrypt(message, shift) << "\n";
86
87      return 0;
88  }
89
```

3. QWERTY SUBSTITUTION CIPHER

```
1   #include <stdio.h>
2   #include <string.h>
3
4   int main() {
5       char* ciphertext = "qwertyuiopasdfghjklzxcvbnm";
6
7       char input[500];
8       printf("Enter text: ");
9       fgets(input, sizeof(input), stdin);
10      input[strlen(input) - 1] = 0;
11      int count = strlen(input);
12
13      char output[count];
14      for(int i = 0; i < count; i++) {
15          int index = ((int) input[i]) - 97;
16          if(index < 0) {
17              output[i] = ' ';
18          }
19          else {
20              output[i] = ciphertext[index];
21          }
22      }
23      output[count] = 0;
24
25      printf("output: %s\n", output);
26  }
27
```

4. VIGENERE CIPHER

```cpp
#include<iostream>
using namespace std;

string get_full_key(string pt, string k){
    if(k.size() >= pt.size())
        return k;
    else{
        int psize = pt.size()-k.size();
        int ksize = k.size();
        while(psize >= ksize){
            k += k;
            psize -= ksize;
        }

        k += k.substr(0, psize);
        return k;
    }
}
string get_encryption(string pt, string k){

    string ct = "";
    for(int i=0;i<pt.size();i++)
        ct += (char) (((int)pt[i]-'A' + (int)k[i]-'A') % 26) + 'A';

    return ct;
}

string get_decryption(string ct, string k){

    string pt = "";
    for(int i=0;i<ct.size();i++)
        pt += (char) ((((int)ct[i]- 'A' - (k[i] -'A')) + 26) % 26) + 'A';

    return pt;
}

int main(){

      string plaintext = "ANIRUDHNAKRA";


    string key = "GOD";

    key = get_full_key(plaintext, key);

    cout<<" THE PLAINTEXT: "<< plaintext <<endl;

    cout<<" THE KEY: "<< key <<endl;


    string ciphertext = get_encryption(plaintext, key);

    cout<<" THE CIPHERTEXT: "<< ciphertext <<endl;


    plaintext = get_decryption(ciphertext, key);

    cout<<" THE DECODED PLAINTEXT: "<< plaintext <<endl;

    return 1;
}
```

5. PLAYFAIR CIPHER

```
#include <iostream>
#include <stdio.h>
#include <string>
#include <vector>
using namespace std;

class PlayFair
{
  public:

    PlayFair ();
    ~PlayFair () {}

    void setKey (string k) { key = k; }
    string getKey () { return key; }

    string keyWithoutDuplicateAlphabet (string k);
    string encrypt (string str);
    string decrypt (string str);

    void setMatrix ();
    void showMatrix ();

    int findRow (char ch);
    int findCol (char ch);

    char findLetter (int x_val, int y_val);

  private:

    char matrix[5][5];
    string key;
};

PlayFair::PlayFair ()
{
      for (int i = 0; i < 5; i++) {
       for (int j = 0; j < 5; j++) {
           matrix[i][j] = 0;
       }
    }
}

string PlayFair::keyWithoutDuplicateAlphabet (string k)
{
    string str_wo_dup;

    for (string::iterator it = k.begin(); it != k.end(); it++) {
    bool alphabet_exist = false;
    for (string::iterator it1 = str_wo_dup.begin(); it1 != str_wo_dup.end(); it1++) {
        if (*it1 == *it) {
            alphabet_exist = true;
        }
    }

        if (!alphabet_exist) {
            str_wo_dup.push_back(*it);
        }
```

```
    }

    return str_wo_dup;
}

void PlayFair::setMatrix ()
{
    string kwda = keyWithoutDuplicateAlphabet(getKey());

    int i_val, j_val;

    int count = 0;

    for (int i = 0; i < 5; i++) {
      for (int j = 0; j < 5; j++) {
      if (count == kwda.length()) break;
      else {
        matrix[i][j] = toupper(kwda[(5 * i) + j]);
        ++count;
      }
      }
      if (count == kwda.length()) break;
    }

    for (int i = 0; i < 26; i++) {
    char ch = 65 + i;
    bool alphabet_exist = false;

    for (string::iterator it = kwda.begin(); it != kwda.end(); it++) {
        if (ch == toupper(*it)) {
        alphabet_exist = true;
        }
    }

    if (ch == 'J') alphabet_exist = true;

    bool exit = false;
    if (!alphabet_exist) {
        for (int i = 0; i < 5; i++) {
        for (int j = 0; j < 5; j++) {
            if (!isalpha(matrix[i][j])) {
            matrix[i][j] = toupper(ch);
            exit = true;
            }
            if (exit == true) break;
        }
        if (exit == true) break;
        }
    }
    }
}

void PlayFair::showMatrix()
{
    for (int i = 0; i < 5; i++) {
    for (int j = 0; j < 5; j++) {
        if (matrix[i][j] != 'I') cout << matrix[i][j] << "\t";
```

```
116             else cout << "I/J" << "\t";
117         }
118         cout << endl;
119         }
120         cout << endl;
121     }
122
123     int PlayFair::findRow (char ch)
124     {
125
126         if (ch == 'j') ch = 'i';
127         for (int i = 0; i < 5; i++) {
128         for (int j = 0; j < 5; j++) {
129             if (matrix[i][j] == toupper(ch)) { return i; }
130         }
131         }
132         return -1;
133     }
134
135     int PlayFair::findCol (char ch)
136     {
137
138         if (ch == 'j') ch = 'i';
139         for (int i = 0; i < 5; i++) {
140         for (int j = 0; j < 5; j++) {
141             if (matrix[i][j] == toupper(ch)) { return j; }
142         }
143         }
144         return -1;
145     }
146
147
148     string PlayFair::encrypt (string str)
149     {
150         string output;
151
152
153         for (int i = 1; i < str.length(); i = i + 2) {
154         if (str[i-1] == str[i]) {
155             string temp1, temp2;
156
157             for (int j = 0; j < i; j++) {
158             temp1.push_back(str[j]);
159             }
160
161             for (int j = i; j < str.length(); j++) {
162             temp2.push_back(str[j]);
163             }
164
165             str.clear();
166             str = temp1 + 'x' + temp2;
167         }
168         }
169
170         for (int i = 0; i < str.length(); i = i + 2) {
171
```

```
    if (findRow(str[i]) == findRow(str[i+1])) {
        output.push_back(matrix[findRow(str[i])][(findCol(str[i]) + 1) % 5]);
        output.push_back(matrix[findRow(str[i + 1])][(findCol(str[i + 1]) + 1) % 5]);
    }


    else if (findCol(str[i]) == findCol(str[i+1])) {
        output.push_back(matrix[(findRow(str[i]) + 1) % 5][findCol(str[i])]);
        output.push_back(matrix[(findRow(str[i + 1]) + 1) % 5][findCol(str[i + 1])]);
    }


    else {
        output.push_back(matrix[findRow(str[i])][findCol(str[i + 1])]);
        output.push_back(matrix[findRow(str[i + 1])][findCol(str[i])]);
    }
    }


    if ((str.length() % 2) != 0) {
    output[output.length() - 1] = toupper(str[str.length() - 1]);
    }

    return output;
}

```

```
string PlayFair::decrypt (string str)
{
    string output;

    for (int i = 0; i < str.length(); i = i + 2) {


    if (findRow(str[i]) == findRow(str[i+1])) {
        int y;
        if ((findCol(str[i]) - 1) >= 0) y = (findCol(str[i]) - 1);
        else y = 4;
        output.push_back(matrix[findRow(str[i])][y]);

        if ((findCol(str[i + 1]) - 1) >= 0) y = (findCol(str[i + 1]) - 1);
        else y = 4;
        output.push_back(matrix[findRow(str[i + 1])][y]);
    }


    else if (findCol(str[i]) == findCol(str[i+1])) {
        int x;
        if ((findRow(str[i]) - 1) >= 0) x = (findRow(str[i]) - 1);
        else x = 4;
        output.push_back(matrix[x][findCol(str[i])]);

        if ((findRow(str[i + 1]) - 1) >= 0) x = (findRow(str[i + 1]) - 1);
        else x = 4;
        output.push_back(matrix[x][findCol(str[i + 1])]);
```

```
    }


    else {
        output.push_back(matrix[findRow(str[i])][findCol(str[i + 1])]);
        output.push_back(matrix[findRow(str[i + 1])][findCol(str[i])]);
    }
    }


    for (int i = 0; i < output.length(); i++) {
    if (output[i] == 'X') {
        output.erase(output.begin() + i);
    }
    }

    return output;
}

int main () {
    PlayFair pf;
    string key, input;

    cout << "Put key value (put alphabets/words): " << endl;
    getline(cin,key);
     cout << key << endl;

     pf.setKey(key);
     pf.setMatrix();
     pf.showMatrix();

     cout << "Press 1: Encrypt | 2: Decrypt | 3: Quit" << endl;
     int choice;
     cin >> choice;

     while (choice != 3) {

         cout << "Put your text: ";
         fflush(stdin);
         getline(cin,input);

         if (choice == 1) cout << pf.encrypt(input) << endl;
         else cout << pf.decrypt(input) << endl;

         cout << "Press 1: Encrypt | 2: Decrypt | 3: Quit" << endl;
         cin >> choice;
     }
     return 0;
}
```

6. XOR ENCODER

```
#include<iostream>

using namespace std;

void encrypt(char *);
void decrypt(char *);

int main(void){
    char str[29]="Anirudhisagod";

    cout<<"Accepted cipher:\n";
    cout<<str;
    cout<<endl<<endl;

    encrypt(str);

    cout<<"Encrypted cipher:\n";
    cout<<str;
    cout<<endl<<endl;;

    cout<<"Decrypted cipher:\n";
    decrypt(str);
    cout<<str;
    cout<<endl;
}

void encrypt(char *str){
    while(*str!='\0'){
        *str^=18;
        str++;
```

```
    cout<<"Decrypted cipher:\n";
    decrypt(str);
    cout<<str;
    cout<<endl;
}

void encrypt(char *str){
    while(*str!='\0'){
        *str^=18;
        str++;
    }
}

void decrypt(char *str){
    while(*str!='\0'){
        *str^=18;
        str++;
    }
}
```

6. RSA ALGORITHM

```
#include<iostream>
#include<math.h>
#include<string.h>
#include<stdlib.h>

using namespace std;

long int p, q, n, t, flag, e[100], d[100], temp[100], j, m[100], en[100], i;
char msg[100];
int prime(long int);
void ce();
long int cd(long int);
void encrypt();
void decrypt();
int prime(long int pr)
{
    int i;
    j = sqrt(pr);
    for (i = 2; i <= j; i++)
    {
        if (pr % i == 0)
            return 0;
    }
    return 1;
}
int main()
{
    cout << "\nENTER FIRST PRIME NUMBER\n";
```

```
    cin >> p;
    flag = prime(p);
    if (flag == 0)
    {
        cout << "\nWRONG INPUT\n";
        exit(1);
    }
    cout << "\nENTER ANOTHER PRIME NUMBER\n";
    cin >> q;
    flag = prime(q);
    if (flag == 0 || p == q)
    {
        cout << "\nWRONG INPUT\n";
        exit(1);
    }
    cout << "\nENTER MESSAGE\n";
    fflush(stdin);
    cin >> msg;
    for (i = 0; msg[i] != '\0'; i++)
        m[i] = msg[i];
    n = p * q;
    t = (p - 1) * (q - 1);
    ce();
    cout << "\nPOSSIBLE VALUES OF e AND d ARE\n";
    for (i = 0; i < j - 1; i++)
        cout << e[i] << "\t" << d[i] << "\n";
    encrypt();
    decrypt();
    return 0;
```

```
}
void ce()
{
    int k;
    k = 0;
    for (i = 2; i < t; i++)
    {
        if (t % i == 0)
            continue;
        flag = prime(i);
        if (flag == 1 && i != p && i != q)
        {
            e[k] = i;
            flag = cd(e[k]);
            if (flag > 0)
            {
                d[k] = flag;
                k++;
            }
            if (k == 99)
                break;
        }
    }
}
long int cd(long int x)
{
    long int k = 1;
    while (1)
    {
        k = k + t;
        if (k % x == 0)
```

```
            return (k / x);
    }
}
void encrypt()
{
    long int pt, ct, key = e[0], k, len;
    i = 0;
    len = strlen(msg);
    while (i != len)
    {
        pt = m[i];
        pt = pt - 96;
        k = 1;
        for (j = 0; j < key; j++)
        {
            k = k * pt;
            k = k % n;
        }
        temp[i] = k;
        ct = k + 96;
        en[i] = ct;
        i++;
    }
    en[i] = -1;
    cout << "\nTHE ENCRYPTED MESSAGE IS\n";
    for (i = 0; en[i] != -1; i++)
        printf("%c", en[i]);
}
void decrypt()
```

```
}
void decrypt()
{
    long int pt, ct, key = d[0], k;
    i = 0;
    while (en[i] != -1)
    {
        ct = temp[i];
        k = 1;
        for (j = 0; j < key; j++)
        {
            k = k * ct;
            k = k % n;
        }
        pt = k + 96;
        m[i] = pt;
        i++;
    }
    m[i] = -1;
    cout << "\nTHE DECRYPTED MESSAGE IS\n";
    for (i = 0; m[i] != -1; i++)
        printf("%c", m[i]);
}
```

7. SHA1 and SHA256

```
import hashlib

print(hashlib.algorithms_guaranteed)

import hashlib

name = 'Anirudh'

encoded_name = name.encode()

hashed_name = hashlib.sha256(encoded_name)

print("Object:", hashed_name)
print("Hexadecimal format:", hashed_name.hexdigest())
```

```
import hashlib

str = "Anirudh"

encoded_str = str.encode()

hash_obj = hashlib.sha1(encoded_str)

hexa_value = hash_obj.hexdigest()

print("\n", hexa_value, "\n")
```