**Hybrid Cryptosystem**

*Final report*

Steve McQueen, Matthew McQuaig, Tyler Rutland

Department of Computer Science and Information Technology

Armstrong State University

03-May-2016

Table of Contents

[1 Executive summary 3](#_Toc450243499)

[2 Project background 4](#_Toc450243500)

[2.1 Needs statement 4](#_Toc450243501)

[2.2 Goal and objectives 5](#_Toc450243502)

[2.3 Design constraints and feasibility 6](#_Toc450243503)

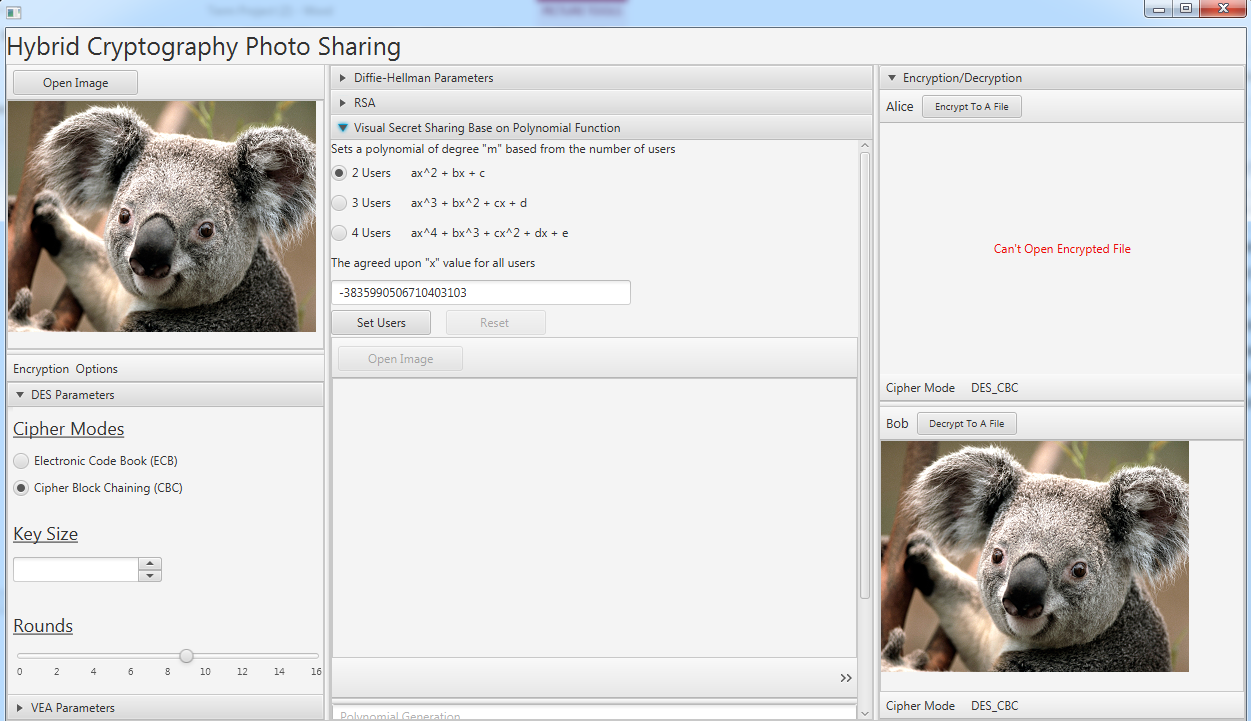
[2.4 Literature and technical survey 7](#_Toc450243504)

[3 Final design 9](#_Toc450243505)

[3.1 System description 9](#_Toc450243506)

[3.2 Approach for design validation 11](#_Toc450243507)

[4 Implementation notes 11](#_Toc450243508)

[ 14](#_Toc450243509)

[5 Experimental results 15](#_Toc450243510)

[6 User’s Manuals 17](#_Toc450243511)

# Executive summary

The threat, how to authenticate, forms of cryptography, and individual device and network performance, were considered before development. The threat is any unauthorized individual or system who succeeds in viewing at least part of the plaintext of the message. The goal was to develop a program to send messages to a single users or multiple users securely. The ability of an unauthorized user to view the plaintext message needed to be reduced by using methods stated in the project requirements. To do so, objectives were set to include a complex authentication procedure used to encrypt and decrypt message and verify the sender and receiver was developed. These procedures consisted of procedures where a sender would choose an image for a message, choose an encryption protocol and the options for that protocol, the receivers, and values based on visual secret sharing.

The final design implementation was completed in JavaFXML FXML in a single-page / multi-section format. As required, the user chooses between DES or VEA for secure image transmission, the DES/VEA number of rounds, an image file to be used during the secret sharing algorithm, the image to be encrypted or decrypted, Diffie-Hellman keying parameters, RSA key parameters, and key size. A secret sharing protocol allows users to randomly select shares from a shared image file. A symmetric polynomial is used to determine points based on the chosen image pixel. This image is shared between users. The chosen pixel is then used to determine a large and specific number to be added to the polynomial dependent on m users. When the message is received, the receiver can decrypt the message with the public keys, and their private key resulting in an identical copy of the original message sent by the sender.

The program was tested with several images of different display sizes and file size. No problems have been encountered although the time to encrypt and decrypt larger files does take more time especially when a user chooses a high number of rounds. The original file size of a chosen image was equal to the encrypted file size of that image. The decrypted file size of a previously-encrypted image was equal to the file size of that original image. Also, no options or protocols exist to reduce either image size or quality. Therefore, it can be concluded that the image received is of equal size and quality to that of the image sent.

# Project background

The system created for this project helps to solve the specific problem of secure messaging between users with minimum network requirements. It is specific because this system sends images. It can, however be expanded to send other forms of data. The problem this solves is messaging between users which may have been sent otherwise unencrypted. Here, not only are messages encrypted but they are done so using RSA, DH, and visual secret sharing to improve on the security already provided by the individual encryption protocols themselves.

## Needs statement

Countless messages are sent over networks every second. This project allows for those messages to be sent securely – a need for almost anyone looking to send messages over a network. Although the network for this project is simulated, it maintains a separation of the sender and receiver(s) and sends the message securely by enacting multiple cryptographic and secret-sharing methods. The options are the key to sending secure messages. By combining the option to choose the encryption protocol, rounds, Diffie-Hellman parameters, the DES parameters of ECB or CBC, and visual secret sharing, the messages are secure from end-to-end during transmission. The encryption protocols of which 1 can be chosen are DES and VEA. DES is a Feistel system that encrypts 64-bit blocks with a 56-bit key. The system created for this project includes options for Electronic Code Book (ECB) or Cipher-Block Chaining (CBC). VEA encrypts 64-bit blocks using a key of variable-length. VEA consists of a key-expansion part, which expands a key to a length up to 448 bits and expands it into sub keys which total 4,168 total bytes, and an encryption part.

Current protocols are generally perform well sending and receiving messages securely and quickly. Still, other options need to be researched to include the option of combining existing methods. The need for these options is evident in the problems presented existing methods which include not only the ability of an attacker to gain access to and decrypt at least part of a message, but also the performance efficiency and network requirements for sending all messages. Efficiency can be measured in by the speed at which the system opens, handles various commands, encrypts and decrypts, its ability to send messages in a manner that is quick, and its efficiency in sending a message with the least amount of information necessary for the receiver to decrypt the message. Protocols exist that are incredibly secure, but the message size is large due to the amount of data that must be sent with the message to be able to decrypt it. These protocols, while incredibly secure, are not practical when performance and network bandwidth are limited. In this project, a hybrid cryptographic system was created to combine methods of encryption and secret sharing with performance, network limitations, and security considerations. The result is a far more secure way of sending messages than some previous systems.

## Goal and objectives

The goal is to develop a system where, among other requirements, any message of any size over any network can be securely sent and could never be viewed as it would be by the sender or the intended receiver. Today, this system simply does not exist. Network messages are intercepted and data at rest is often unencrypted. Developers are working to solve these problems but as of today almost no data is safe if the machines where said data resides is connected to a network. Even then it could still be stolen if the physical machine where it resides is stolen. If the best developers in the world create programs and methods for sending messages securely still often fail then what chance exists to create something secure for this project? Little. However, an attempt to create something better than other teams on the level can be worthwhile in the sense that progress has been made in the overall knowledge of the existence of threats and the need for working authentication and cryptography procedures.

The objective was to develop a system that allows messages to be sent to a 1 or more users using a hybrid cryptosystem with many options to make the system more secure. DES is one option and DES is no longer considered to be a secure option. So a flaw already exists. But, by allowing the user to set their own key, the number of rounds, and other options, the security of DES in the system is greatly increased to a point where a real security threat is unlikely. Performance of DES is still impressive compared to more secure protocols such as AES. This performance is reduced, however, when the rounds are increased. 3-DES, for instance, requires more time than DES because it runs DES 3 times. While the detriment to performance is obviously due to rounds, it shows that running DES many times would greatly improve security while greatly reducing performance. If a user chooses to run DES or VEA for 100 rounds, as the system created by this team allows, performance will not be very good but security will be. The number of rounds vs. security is something to consider if encrypting a large message. VEA will perform better than DES when run once or many times but the improvement to security of running the system 100 times is unnecessary if the far-improved security of 3-DES over DES is considered. With a great improvement in security by simply running DES three times the performance is unlikely to be a problem if using capable hardware. So using the system created for this project will perform well while delivering a secure message, especially with multiple rounds.

## Design constraints and feasibility

Technical constraints required a simulated network environment. Because extensive knowledge of computer networks were not a prerequisite for this course, the use of a real and working computer network wasn’t feasible. Using a simulated network will alter the performance of sending and receiving messages by removing the factors that would have been present, otherwise, in a network that has its own performance variations. This means that instead of the system user sending a message which would have needed to exit the sender’s machine and travel to an intermediate host, all the while being presented with obsticles of normal network activity, to eventually be delivered to a receiver, the sender and receiver reside on the same machine which leaves performance measurements in question. The time to decrypt a message can also be questioned because the same system is doing both encryption and decryption. While the time required to decrypt a message may take n seconds on one machine, it may have otherwise taken more or less time on another machine. The time to encrypt/decrypt and send a message using the system is still impressive. One particular test was run where a 3-Megabyte image was encrypted, sent, then decrypted in a very short time. The test was performed on a powerful computer but the computer CPU was at 60% usage and the memory had 80% usage before the test was started. This test provided evidence that if a powerful computer can complete the system’s intent while the computer’s performance had already been greatly reduced, a less-powerful computer could likely complete the system’s intent with impressive performance if it, unlike the other computer, was using less of its CPU and memory. So although the network is simulated and the same computer is used to send and receive, it can be expected that performance differences will not be significant during a typical use of the system.

Time to develop the system has been the main obstacle. Because the students of this team are students with other courses and employment outside of Armstrong State University, time to devote to developing a perfect system simply does not exist. So development was completed between and during class and on nights and weekends. This system, like any other system developed by any student for any class at any school, will not be perfect. But it should be known that every attempt to not only understand each requirement but to implement those requirements into the system were tried. And, as far as it is known by the members of this team, each requirement was indeed implemented into the system to the best of the ability of the members of this team. Equally, each section of this paper were completed to the best of the ability of the members of this team.

## Literature and technical survey

Other encryption and encrypted-file-sending commercial products are available that work as desktop software, browser applications, or smart device apps. While none of these work exactly like the system described in this paper, many do offer encryption and the ability to send those encrypted messages. However, only 7-Zip allows the user to select the encryption type. None of them allow the user to select options, see keys, or choose between EBC and CBC. They do, however, all use AES which is a stronger type of encryption overall. Five applications that are somewhat similar to the system described in this paper are listed below with a brief description/review of each.

* Wickr is an iPhone and Android app that allows images to be encrypted but are soon permanently deleted automatically after being decrypted by the receiver (Greenberg). Wickr uses AES and RSA to encrypt plaintext and the cipher text remains encrypted through sending process (Greenberg). Wickr supports images, video, and text and not only deletes the content after decryption but ensures that the content is unrecoverable by overwriting the storage space previously occupied by the message content with random bytes several times (Greenberg). Further measures to protect security of the sender’s message by Wickr by disabling the cut and paste feature of phones and also requires the constant contact of a particular button (Greenberg). Releasing the button or shaking the phone will cause the image to disappear from the screen (Greenberg) to keep people from attempting screen-shots.
* InfoEncrypt offers a free service that works inside a web browser that performs encryption and send text messages. It uses AES-128 with PBKDF2, CBC and a random IV (InfoEncrypt.com). Because it works in the browser, the message doesn’t actually need to be sent to the company’s servers (InfoEncrypt.com).
* Signal is a desktop application that is also available to a variety of smart devices. Signal allows users to encrypt and send messages to a group using Curve25519, AES-256, and HMAC-SHA256 (Whispersystems.com). Signal is also open-source so it costs nothing and its source-code is available on GitHub (Whispersystem.com).
* Although WhatsApp delivers far superior features than anything our system provides, it is still like our program in that it encrypts images and sends them. WhatsApp encrypts images using Whispersystems’ Signal encryption system (the same as the previous example) so it uses Curve25519, AES-256, and HMAC-SHA256 (WhatsApp.com).
* An option in 7-Zip is to encrypt the zipped content with AES-256 or ZipCrypto. AES-256 is the stronger option of the two. 7-Zip is also free to download from their website (7-Zip.com). Other applications like WinZip allow for AES-256 encryption as well. This may possibly be the easiest option of all for most users because many computers already will have either 7-Zip or WinZip installed. The file could then just be sent by email. The problem is that the recipient of the message will also need an application to decrypt the file.

This project uses DES and VEA while every other system seems to use either AES-128 or AES-256. These systems may, however, not offer the ability to change the rounds, offer visual secret sharing, or any of the several options offered by the system described in this paper. So although the particular encryption type is stronger in other systems, the hybrid cryptosystem used in the system described in this paper may actually be stronger. Unfortunately, to provide any proof of that would require extensive testing that is out of the scope of this project and the abilities of the students who worked on it. The other programs also often offered many features for several formats and are available on many platforms. The system described in this paper is available for a single platform and has few features but could fulfill a market of customers who want a specific-use product of small size and numerous encryption options.

# Final design

## System description

This system is written in the Java programming language with a Java FXML-based graphical user interface. The file system is composed of many Java controller, modeling, utility, and service classes which aid in sharing information between the system components. The graphical user interface uses JavaFXML buttons, JavaFXML ImageView, JavaFXML accordion menus, JavaFXML sliders for numerical selections, JavaFXML text boxes for numerical input, JavaFXML radio buttons, and JavaFXML progress bars. The system design goal was to create an easy-to-use central hub for the user to succeed using whichever part of the system they wish to use. System interface parts which go together are grouped together for a more user-friendly and ergonomically-sound user environment.

Figure 1: DES Encrypt Block Diagram

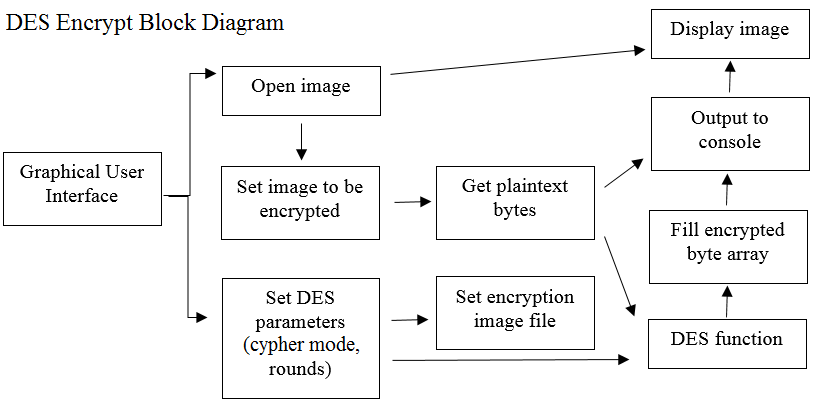


Figure 2: DES Decryption Block Diagram

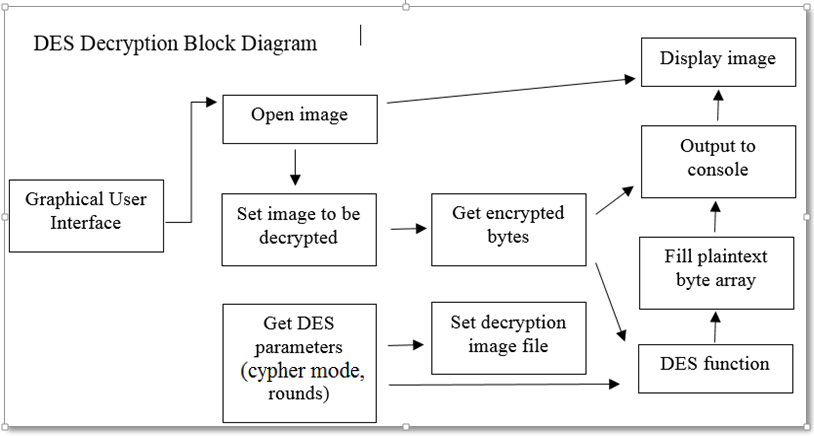


Figure 3: VEA Encryption Block Diagram

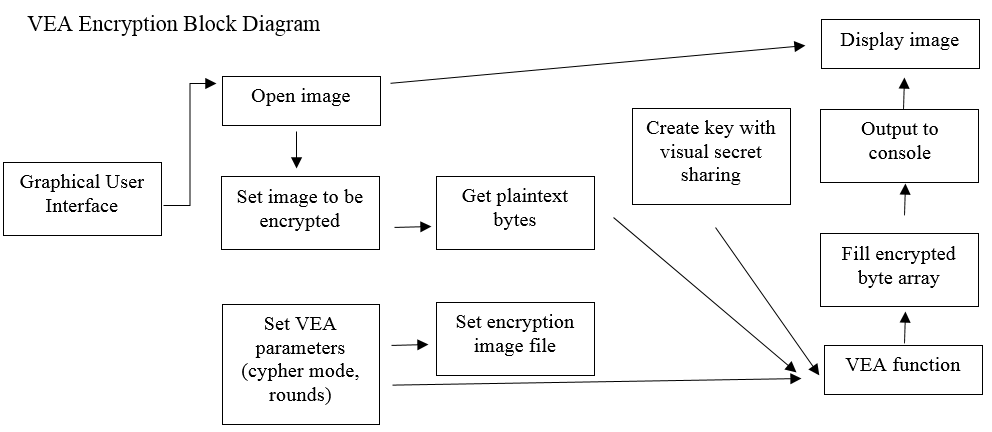


Figure 4: VEA Decryption Block Diagram

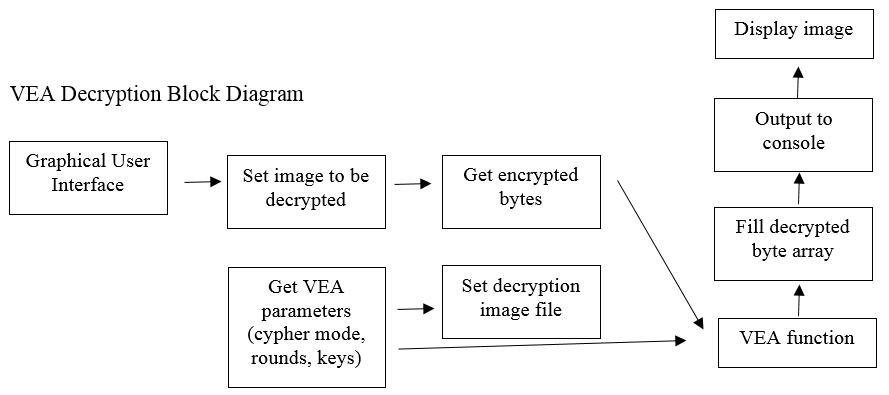


Figure 5: Polynomial-Based Secret Block Diagram

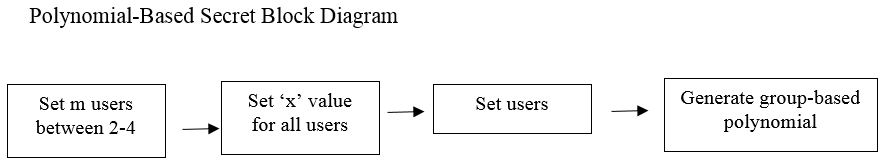
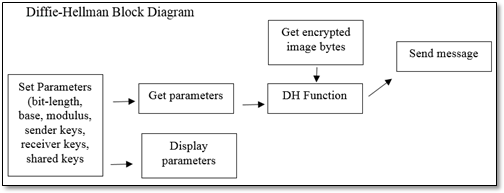


Figure 6: Diffie-Hellman Block Diagram



## Approach for design validation

# Implementation notes

This hybrid cryptography system implements DES and VEA encryption with Diffie-Hellman and RSA to create keys used by a messaging system to send secure messages between a sender and receiver(s). The DES encryption process takes an image, DES parameters of cipher modes Electronic Code Book and Cipher Block Chaining and the desired rounds. The main page of the user interface is MainApplication.java. When a user selects the “open” button and selects an image, the bytes that compose that image are read into memory and become available to the DES encryption class. An empty image file is created when a user selects the “encrypt” button and chooses an existing image or provides a file name for a new one. A new DES object is created, the plaintext image bytes along with the DES parameters are then sent to the DES class for processing the number of rounds. The encrypted bytes are then returned to MainApplication.java. When a user selects the “decrypt” button and selects an image to decrypt, the encrypted bytes are read into memory and become available to the DES encryption class. The encrypted image bytes along with the DES parameters are then sent to the DES class for processing the number of rounds. The decrypted bytes are then returned to MainAppliction.java.

VEA takes initial parameters of cipher modes Electronic Code Book, Cipher Block Chaining, and desired rounds. Key generation is completed using the polynomial-based secret key system. The polynomial-based secret share system takes options of the number of users of 2 through 4 which represent a particular polynomial. For example, 2 users would represent the polynomial ax^2 + bx + c with the x a agreed upon value. This too can be manually set in the same system or a user can use the randomly-generated value which already populates the JavaFXML text box. Once all inputs are completed the user would select the “Set Users” button or have the option to reset the inputs to the default settings. The a, b, and, optionally, c,d, and e values are then populated using the Visual Secret Sharing system. Here, a user will open an image to be shared with all users. Users will then select individual pixels within the image which represent X, Y coordinates on the image. Each user selects their own values. Once the number of users which were previously selected have set their values, the “Generate Polynomial” button is selected which will populate the polynomial with the appropriate values. These values make up the keys used for VEA.

Reproduction of this system is as simple as copying files. There are no patents, copyrights, or protection systems to foil a user from copying the files which compose this system. Modification will require at least some working-knowledge of Java and JavaFXML programming languages and DES and VEA encryption protocols and Diffie-Hellman and RSA systems. Much of this system was completed using JavaFX Scenebuilder, a program which can be downloaded from Oracle, the proprietors of Java and JavaFX, at <http://www.oracle.com/technetwork/java/javafxscenebuilder-1x-archive-2199384.html>.

Figure 7: System open and at rest

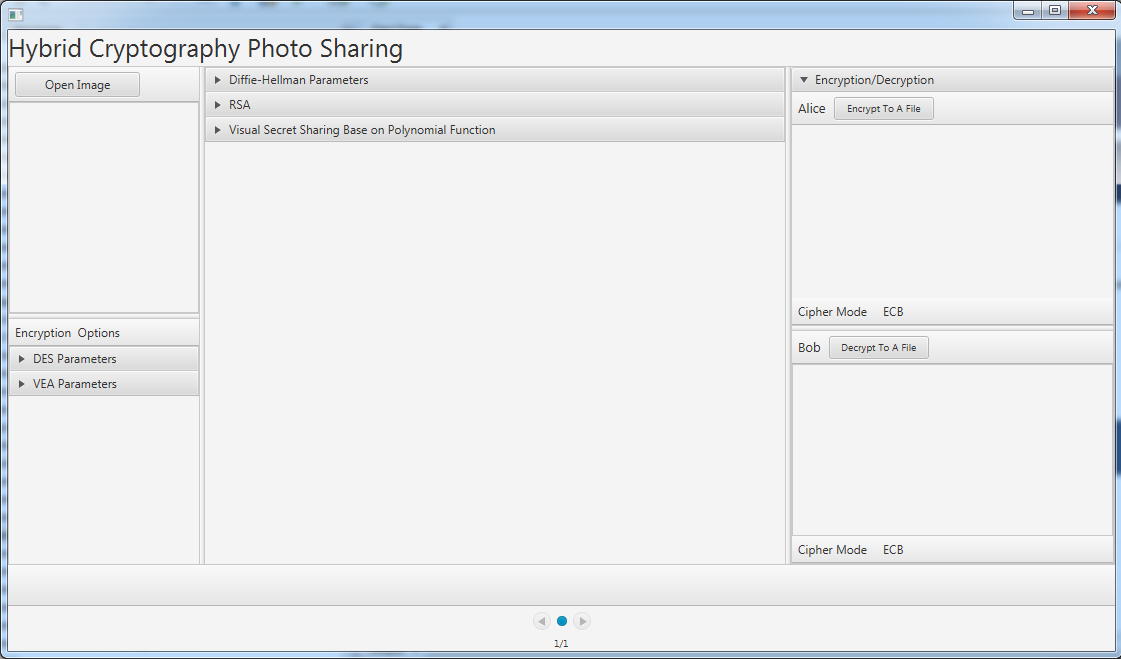


Figure 8: System functions

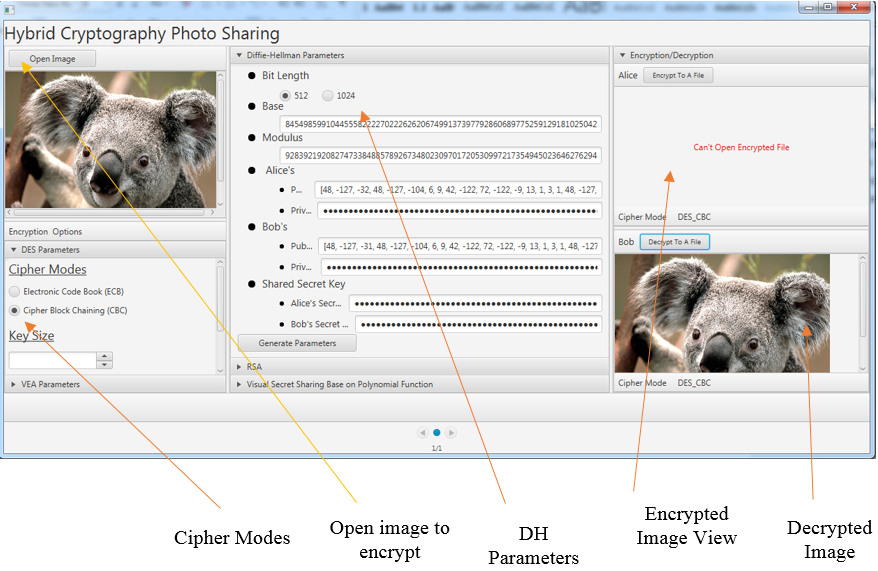


Figure 9: Visual Secret Sharing Based on Polynomial Function

# 

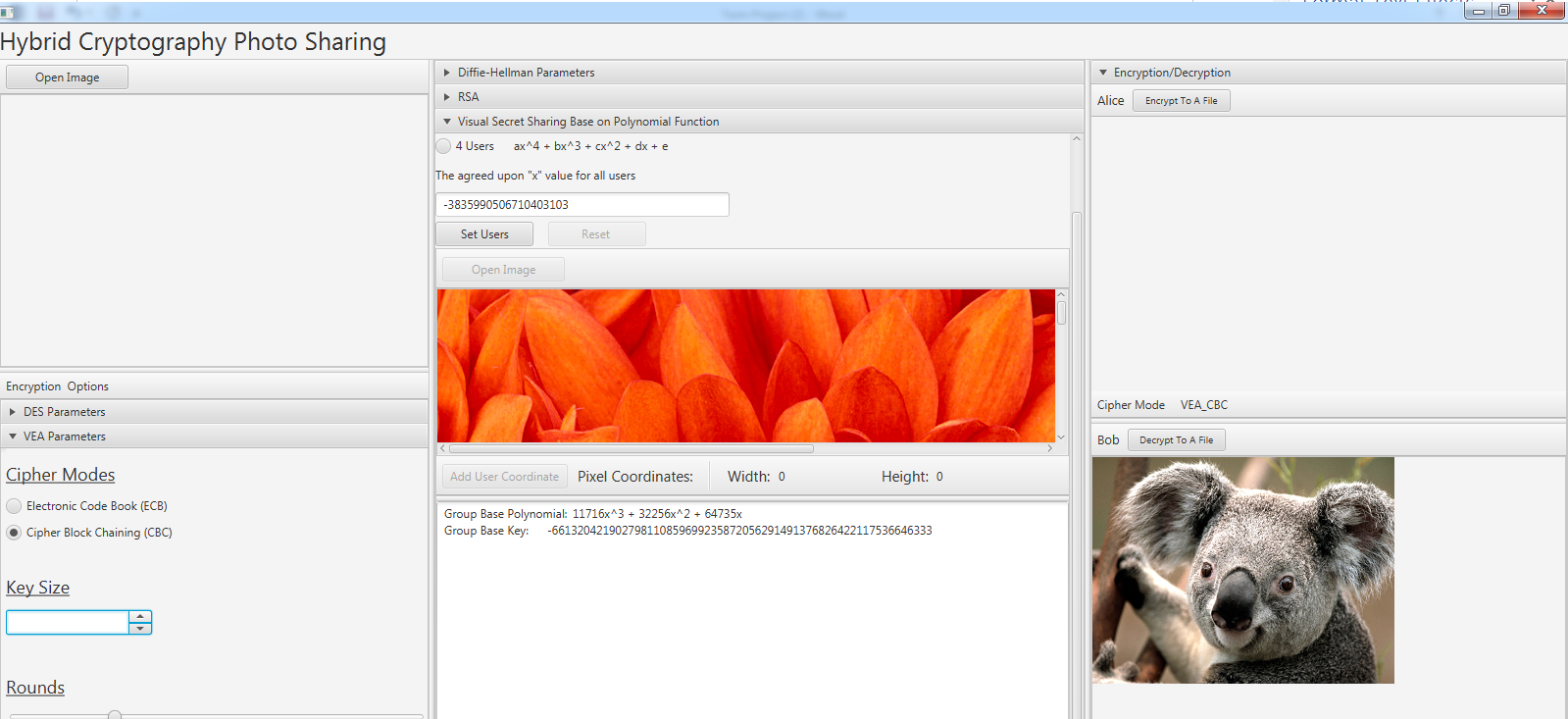
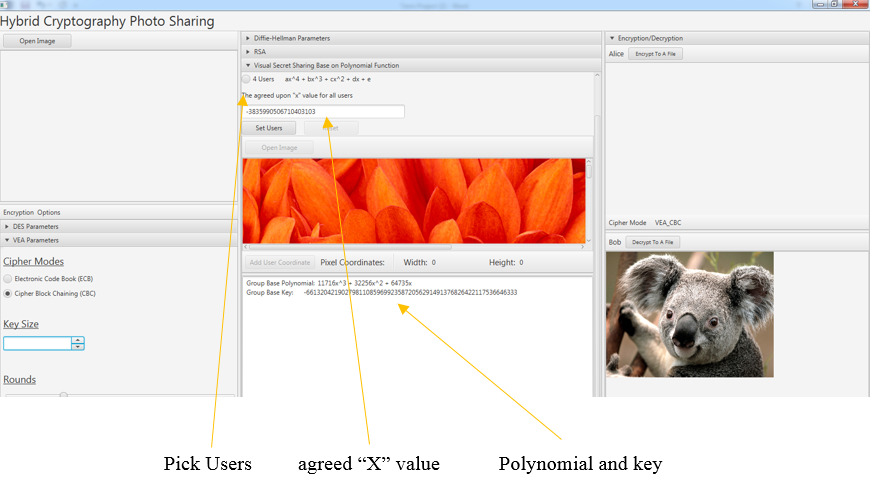


Figure 10: Visual Secret Sharing Setup



# Experimental results

The system is designed to accept .bmp, .jpeg, .png, and .gif file formats. Each image format was tested in each encryption format and sent to a receiver using system protocols. File sizes were tested to include files of 5 kilobytes to 508 megabytes and each encrypted, decrypted, and sent successfully. User-generated, user-input, and automatically-generated parameters all worked successfully as tested. Because the network is simulated, the sender and receiver all “live” in the same system, therefore, making it easy to test the sending and receiving of messages. Because this is simulated, both the sender and receiver private keys are seen by the system designer so they can be verified to work properly. If there are any instances where the system fails in any of these areas, these failures are unknown to the system designers at the time of this paper.

The system itself is very small at only approximately 1.5 megabytes. On the test machine, the program required 1% of CPU and .18 megabyte of memory at rest although this will increase during operations especially if operating on an image with a very large file size. However, the system was tested with an image of file size 508 megabytes and normal operation of the system and various other operations were still possible without serious lag. The machine performance was monitored before running the system, while running the system with it at rest, and while the system was encrypting. Each state showed pleasing results with little effect on CPU and memory consumption. System diagnostics are particularly unrealistic, however, because they can be changed with other applications, processes, and the operating system itself.

Figure 11: System performance before running system

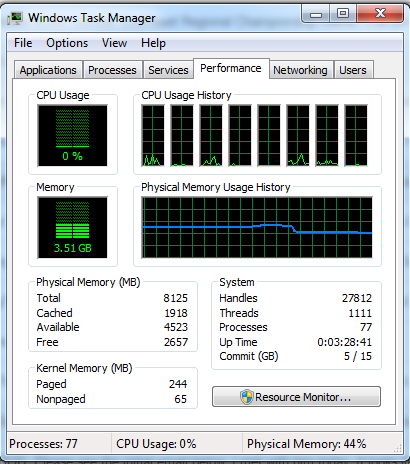


Figure 12: System performance with system open and at rest

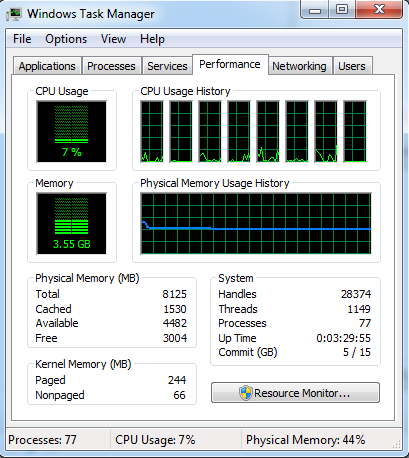
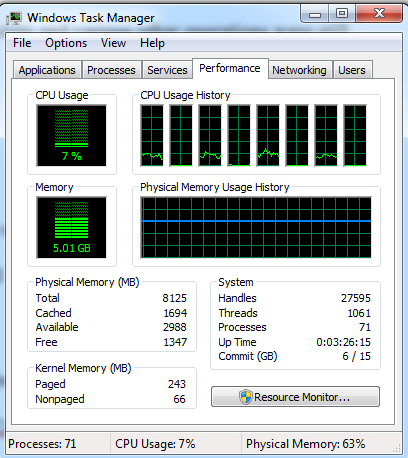


Figure 13: System performance while encrypting a typical image



As can be seen from the images provided, there is little effect on the host machine’s performance while running the system and performing typical processes. This is particularly important to machines that would run this system because it shows that the system can be run and used on host machines that have other processes running or host machines that may not have the best components for performance. Because there was little requirements for performance, this system can be applied to many different types of machines with varying performance specifications.

# User’s Manuals

This program can be run from the cmd line or by an IDE. There is no installation required. To get to the correct file to run the system, a user would go to HybridCryptography -> src -> main -> java -> hybridcryptography -> main -> MainApplication.java. No other files should need to be opened.

Image files that a user would desire to be encrypted would be stored on the user’s system. Encrypted files and decrypted files would also be stored on the user’s system. These files can be stored anywhere although various other files created during the process of encrypting/decrypting and sending/receiving an image may store within the system’s file system. The system itself is very small at only approximately 1.5 megabytes. On the test machine, the program required 1% of CPU and .18 megabyte of memory at rest although this will increase during operations especially if operating on an image with a very large file size. However, the system was tested with an image of file size 508 megabytes and normal operation of the system and various other operations were still possible without serious lag.

References

Greenberg, Andy. *Wickr Lets Your iPhone Send Both Encrypted and Self-Destructing Messages*. Forbes.com. 2012. Web. 04/29/2016