Final Report

Team: // No Comment

Division of Labor

- Ana Briones - **16.7%**
    - RSA, RSA Attacks
- Ivins Jove - **16.7%**
    - RSA, RSA Attacks
- Jacob Mathews - **16.7%**
    - Stego, Stego Attacks, Sudoku, Sudoku Attacks
- Jonathan Kocmoud - **16.7%**
    - Stego, Stego Attacks, Excel Stego, Excel Stego Attacks
- Victor Martinez - **16.7%**
    - Stego, Stego Attacks, Sudoku, Sudoku Attacks
- Mark Seegers - **16.7%**
    - RSA, RSA Attacks

RSA Encryption

Explanation of Approach

We implemented an encryption/decryption system based on RSA. First, we had to determine a block size using base 95 so that all printable characters were usable. This was done by dividing N by 95. Once the block size has been determined, the message is split into blocks and converted to base 95. Then, each block is raised to the power of e. Lastly, each block is divided by 95 to get printable characters. To decrypt the message the same procedure is used in reverse with the use of a block size plus one.

For the three attacks on RSA, we implemented two that factor N and one that finds the encrypted message. The first attack, Fermat factorization, can factor N if p and q are relatively close together and are therefore close to the square root of N. Fermat factorization attempts to solve the equation B^2 = A^2 - N by testing all cases of A where A starts near the square root of N. The second attack, Pollard's p-1 algorithm, can factor N if p-1 and q-1 have relatively small factors. Pollard's p-1 algorithm implements trial division to find factors of N and returns p. The third attack, the Chinese remainder Theorem, can recover the message if same message is encrypted e times using the same e but different n's.

TDD and Agile Practices

For RSA encryption, Test-Driven Development was somewhat difficult. To test the RSA encryption, we had to encypt a message, decyrpt the encrypted message and make sure we got out the same message we put in. However, the attacks on RSA were fairly simple. To make sure the attacks properly cracked RSA we made simple test cases that checked to make sure that N was factored correctly. The agile programming practices stimulated progress by keeping the RSA team updated on what everyone was doing and what else needed to be completed.

Usage Instructions

- cd RSA/
- make
- ./munchkincrypt
- ./dorothy

Image LSB Steganography

Explanation of Approach

We created an image steganography program that hides bits using the least significant bits (LSB) of a BMP image. While there is a difference between the original image and the modified one, the difference is so minute that to the human eye they look the same. To embed the message (in this case, a file), we first convert it to binary. This we embed into the 1st/2nd LSB in the RGB/Grayscale range (depends on choice). To extract the message, we read the last 1/2 LSB and then turn that back into binary data, which then is turned into a file. When passing the file to embed, the program parses the extension to it and includes in with the binary data, so that after extraction it can be written to a correct file type.

We included two attacks with the Image LSB Steganography. The first is a simple compare. When someone wants to embed a message, they are not going to waste the time to take then use an image. Thus, it should be easy to find the original image on the web. With this in hand, the program can check if there are discrepancies between the files. If there are differences, then there could be a message hidden in the image. The second attack is to null data in the image. Based on the user's choice, this will write over the 1-2 LSB of the image to remove any previous data hidden there.

TDD and Agile Practices

This one was hard to work with Test-Driven Development because this dealt with images. However, we did figure out how to code the test cases to properly preempt the writing of the code. The Agile practices we used were very handy for this program, as the requirements changed while we were almost done with this section. Thus, we were able to switch gears without very much idling.

Usage Instructions

- cd Image_LSB/
- make
- ./munchkinsteg \
- ./toto \

Excel Steganography

Explanation of Approach

This steganography uses the look and feel of Microsoft Excel to hide data. Using a CSV (Comma

Separated Values) file, we can create cells which do not anything but spaces in them. By having this white space, the cells look exactly the same, but we can embed a message using them. This is very hard to detect within Excel as someone would have to pick a random cell that has data and then see there are multiple spaces. This is done by encoding the message as hexadecimal numbers, which are put in the form of number of spaces. To extract the data, the program finds any cells with only spaces and converts the corresponding numbers back into text.

There are several attacks for Excel Steganography. First, it checks the CSV file to see if there are any cells with just spaces. If there are, then it warns the user. Also, it outputs the sequence of numbers so that the user can find a corresponding encoding map. The other attack is to null out the data by deleting any spaces in "empty" cells. This way the message is gone and cannot be recovered.

TDD and Agile Practices

This program was fairly simple to use Test-Driven Development as it deals with strings. The tests were created to test simple functionality first before moving into harder examples. Likewise, the attacks were the same way. As this was part of the requirements change during the project, our Agile practices allowed us to quickly add this to our outstanding work. This also greatly helped when we accidentally implemented Sudoku, thinking that it was a steganography system instead of the code that it is.

Usage Instructions

- cd Excel/
- make
- ./munchkinexcel \
- ./glinda \

Sudoku Encryption

Explanation of Approach

This program encrypts a string using a Sudoku puzzle. If given a partial Sudoku puzzle and a key, the program can first complete the puzzle. Then, it can use the puzzle and key to solve the hidden message. The puzzle solver splits the puzzle into different segments and analyzes each segments individually adjusting incorrect ones as the program goes along.

The sudoku attack simply destroys the puzzle making it unsolvable. By putting nonsense numbers into the puzzle it will not give any sort of readable message.

Usage Instructions

- cd Sudoku/
- make
- ./Sudoku