Project 4: Cryptography and Steganography

CSCE 315: Programming Studio

Spring 2015

Your team has been hired by Munchkin Incorporated (Munchkin) to design and develop a set of information assurance tools. To protect their data and the data of their clients, Munchkin wants to be able to send and receive data encrypted using the RSA cryptosystem. To further protect their communications with valuable, high-profile clients, Munchkin also wants to be able to hide their ciphertexts in images. Munchkin's vibrant and popular social-media presence gives them plenty of reason to be posting images on social media and other websites, ostensibly so that their brand can maintain its recognizability and relevance.

In addition to the tools for protecting the confidentiality of their clients, Munchkin is requesting that you build tools for cracking RSA and detecting images which are hiding secret information. The reason for this, they tell you, is that they believe their direct competitor, Elphaba International (Elphaba), will attempt to use ripoffs of Munchkin's security tools to sabotage Munchkin's reputation in the market. If Munchkin has tools for detecting and decrypting Elphaba's knockoffs, they can expose Elphaba's scammy behavior and protect both their clients and their brand.

Your contract with Munchkin Incorporated begins on April 6th. You will use both Agile Programming and Test-Driven Development to design, build, and deliver the 4 tools to Munchkin Incorporated on or before Cinco De Mayo (the 5th of May).

Agile Programming and Test-Driven Development

Your team will engage in 1-week Sprints. Before each Sprint, you will take your Product Backlog and select from it the Sprint Backlog, a list of functionality to work on for the upcoming Sprint. During each Sprint, you will conduct "daily" Scrum meetings. Your team must hold at least 4 Scrums per week. Each Scrum should last not more than 15-20 minutes. At the Scrum, three questions must be answered by each member (the Sprint Status Check): *What have you done since the last scrum meeting? What has impeded your work? What do you plan on doing between now and the next Scrum meeting?* The answers to these questions should be recorded. Between daily Scrum meetings (ideally at the end of each day), each team member should update the team's Sprint Burndown Chart to show the amount of effort remaining on each task in the Spring Backlog as well as the status of the task. At the end of each 1-week Sprint, your team will hold a Sprint Review Meeting (during a lab session or TA office hours) to demonstrate the new features produced during the Sprint. At the Sprint Review Meeting, your team will also submit the Backlogs, Burndown Charts, and Sprint Status Checks to CSNet. After the Sprint Review Meeting, the process repeats, with your team selecting a new list of functionality from the Product Backlog (including functionality not completed during the previous Sprint) to become the Sprint backlog for the next Sprint. As soon as a tool is ready to be released to Munchkin for deployment (as determined at the Sprint Review Meeting), you should submit the source code to CSNet.

You **MUST** use Test-Driven Development (TDD). In your final presentation to the board of Munchkin Incorporated (during the Final Exam for CSCE 315), you must provide proof that your team used TDD during development. This proof should be provided in the form of the source code for the tests as well as documentation (e.g. timestamped screenshots) showing that your team started with failing tests and then developed code that would pass the tests. Munchkin Incorporated believes so strongly in TDD that your contract with them includes a penalty for not convincing them that you used TDD.

Important Dates

1. April 6th: Begin Sprint #1. Focus on defining requirements and assigning team roles.
2. April 13th: Begin Sprint #2. Submit progress report from Sprint #1 to CSNet.
3. April 20th: Begin Sprint #3. Submit progress report from Sprint #2 to CSNet.
4. April 27th: Begin Sprint #4. Submit progress report from Sprint #3 to CSNet.
5. May 5th: Cinco De Mayo Delivery Day. Submit progress report from Sprint #4 and all deliverables to CSNet.
6. May 8th: Final Presentations at 3:30pm in HECC 203 (504-506)
7. May 11th: Final Presentations at 10:30am in HRBB 124 (501-503).

Rubric

- RSA cryptosystem: 15 pts
  - Generate random primes with approximately k bits, for $16 \leq k \leq 512$
  - Generate correct public and private keys for k-bit moduli, for $32 \leq k \leq 1024$
  - Verify $D_K\left(E_K\left(M\right)\right) = E_K\left(D_K\left(M\right)\right) = M$ using team's own implementation
  - Verify using the *openssl* command line tool:
    - Correct encryption of a single block
    - Correct decryption of a single block
    - Correct encryption of multiple blocks
    - Correct decryption of multiple blocks
- Attacks on RSA: 15 pts
  - At least 3 working attacks on RSA
- LSB image stegosystem: 15 pts
  - Correct embedding of bits in the 1-LSB plane
  - Correct extraction of bits from the 1-LSB plane
  - Correct embedding of bits in the 2-LSB plane
  - Correct extraction of bits from the 2-LSB plane
  - Colors: Grayscale and RGB
  - Correct PSNR reported
- Attacks on LSB image stego: 15 pts
  - At least 3 working attacks on LSB image stego
- Weekly Sprint progress reports: 20 pts
  - 5 points per week
  - Include Backlogs, Burndown Charts, and Sprint Status Checks
- Final Presentation: 20 pts
  - Demonstrate tools for
    - Encryption, Decryption, Embedding, Extracting, Cryptanalysis, Steganalysis
  - Proof of Test-Driven Development
  - Do not exceed 10 minutes

RSA Encryption

RSA is a public-key cryptosystem proposed in 1977 by Rivest, Shamir, Adleman. It is the most successful public-key cryptosystem and is based on the idea that the factorization of integers is a hard problem.

A summary of the RSA algorithm:

1. Generate distinct large primes p and q and compute $n = pq$ and $\phi(n) = (p-1)(q-1)$.
2. Pick public key exponent e such that $1 < e < \phi(n)$ and e and $\phi(n)$ are coprime.
3. Compute private key exponent d such that $ed \equiv 1 \pmod{\phi(n)}$.
4. Release $\{n, e\}$ as the public key. Keep $\{p, q, d\}$ as secret key.
5. To encrypt a message, first convert the plaintext message M to a number m such that $0 \leq m < n$. Then, compute the ciphertext as $c \equiv m^e \pmod{n}$.
6. To decrypt a ciphertext, compute $m \equiv c^d \pmod{n}$ and then convert the number m into the plaintext M by reversing the procedure which converted M to m.

You must create a command line tool named *munchkincrypt* which implements the RSA cryptosystem and related functionality. At minimum, your tool must implement the following functionality:

- generate a prime number of a given approximate bit-length
- generate a public and private key pair of a given approximate bit-length
- encrypt cleartext using a given public key
- decrypt ciphertext using a given private key

You are encouraged to implement functionality in addition to the above list of minimum required functionality.

Cryptanalysis of RSA

The security of RSA depends on several factors. Fundamentally, RSA relies on the problem of factorizing integers being a hard problem. If Eve can factor Alice's public modulus n, then she can easily compute Alice's private exponent d and can therefore read all of Alice's mail and forge Alice's digital signature. If integer factorization turns out to be easy (i.e. if there exists a polynomial time algorithm for factoring), the RSA is completely useless. Another weakness of RSA (and of every cryptosystem) is that the implementation and application of the cryptosystem can be attacked directly. If not implemented and used correctly, RSA can be a piece of cake to break. Here are just a few ways that RSA can be broken due to implementation mistakes.

- If the modulus n is small (less than 512 bits), then a desktop PC and a good factoring algorithm can factor it in a matter of days.
- If the primes p and q that make up the modulus were created in a way that makes them likely to be close together, and therefore close to $\sqrt{n}$, then n can be factored using Fermat factorization.
- If either $p-1$ or $q-1$ has only small prime factors, then n can be factored using Pollard's $p-1$ algorithm
- If $q < p < 2q$ and $d < \frac{n^{1/4}}{3}$, then n can be factored efficiently using Wiener's theorem.
- If the public encryption exponent e is small (e.g. $e = 3$) and the message m is small, so that $m^e < n$, then m can be recovered by finding the e-th root of the ciphertext.

- If the same message is encrypted using the same public exponent, but different public moduli, then the message can be recovered using the Chinese remainder theorem.
- If the random number generator used to generate the prime factors is not sufficiently random, a large-enough collection of public keys generated by it will contain pairs of moduli which can be factored by Euclid's algorithm.

You must create a command line tool named *dorothy* which implements at least three attacks on the RSA cryptosystem. You may choose any three of the above attacks, or you may find/design others.

LSB Image Stego

Steganography (or Stego, for short) is the art and science of hiding data so that only the sender and intended receiver are aware of the existence of the data. It is complementary to cryptography. While cryptography protects *what* Alice sends to Bob by obfuscating the contents of her messages, steganography protects *when* and *if* Alice sends data to Bob by disguising her messages as innocuous channel usages (e.g. as images, audio, text, etc.). As in cryptography, the security of a stegosystem should not rely on the secrecy of the method. You must assume that the adversary knows the system. The strength of the stegosystem is determined by how difficult it is for a passive observer to distinguish between legitimate cover-objects which are not hiding data and stego-objects which contain or encode hidden data. The more difficult it is to detect the hidden data, the more secure the stegosystem.

One of the most basic kinds of steganography deals with images. An image is made up of many tiny picture elements, called pixels. Each pixel is a single color. The color is encoded as a number. For grayscale images, each pixel is 8-bits long and represents one of 256 possible shades of gray. For RGB images, each pixel is 24-bits long and represents 256 shades each of red, green, and blue (you can think of RGB pixels as three 8-bit pixels, one for red, one for green, and one for blue). The low order bits of each pixel control differences in shades of color which are beneath the perceptual threshold of the human visual system. Therefore, these bits can be modified without any perceptible image degradation. This property can be exploited for data hiding by replacing the least significant bits of each pixel with the bits of the secret data.

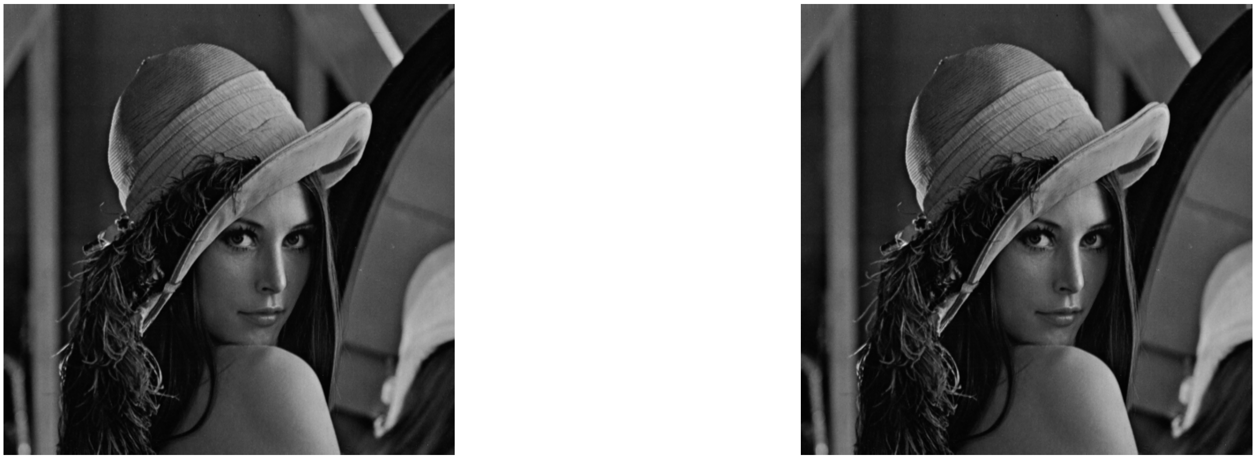

Figure 1: One of these is the original image of Lena and the other is hiding 32KB of secret data.

You must create a command line tool named *munchkinsteg* which implements LSB image stego embedding and extraction for grayscale and RGB images in BMP format. Your tool must support at least 1-LSB (use only the least significant bit) and 2-LSB (use the two least significant bits) embedding, but may support other modes in addition to these. Your tool must report the PSNR (peak signal to noise ratio) of the stego-image.

Steganalysis and Countermeasures

Since the LSB-plane is beneath the human perceptual threshold, a human adversary will not be able to detect the modification of an image by sight alone, even if she has both the original and modified versions (and does not know which is which). However, the deficiency of the human visual system can be compensated for by some cleverness and computational power. Any modification to an original object will introduce some amount of distortion. Given an accurate model of the cover media (e.g. how neighboring pixels in images relate to each other), this distortion can be detected and used to distinguish between authentic cover-objects and modified stego-objects. There are also some more straightforward, but less clever, ways to determine if an image has been modified. In cases where detection may be too difficult, or not of primary importance, there are also countermeasures which may be deployed to disrupt or eliminate the ability for an image to hide data. Some ideas for detection techniques and countermeasures are given below.

- If the original image, or a signature of the original image (e.g. MD5 hash), is known, every version can be compared to that original to determine if the image was modified.
- If the hidden data is known to be cleartext (unencrypted data), then the data can be extracted from a suspect image and tested for structure. If an image is clean, the extracted data will be random, but if the extracted data has an identifiable structure (e.g. ASCII text), then it can be concluded that the image is hiding data.
- A particularly clever technique for detecting LSB stego in images is Regular-Singular Analysis (http://www.ws.binghamton.edu/fridrich/Research/acm_2001_03.pdf). RS analysis is able to detect data hidden at rates as low as 0.05 bits per pixel by looking at the differences between neighboring pixels to estimate the amount (or *length*) of data hidden in the image. If the *length* is greater than some threshold (e.g. 3%), the image is classified as steganographic.
- RS Analysis is a special case of Sample Pairs Analysis (http://www.ece.mcmaster.ca/~sorina/papers/LSBfinalTSP.pdf). Both methods compare a kind of noise in the sample image to an expected level and type of noise in the cover object. Too much noise, or the wrong kind of noise, indicates that the image has likely been modified.
- If you can't beat 'em, make a giant mess. Since LSB-stego hides data in a place where changes do not affect perceived image quality, the entire LSB-plane can be randomized to destroy any hidden data also without affecting perceived image quality.

You must create a command line tool named *toto* which implements at least three attacks on LSB image steganography systems. You may choose any three of the above attacks, or you may find/design others.

Reminder about Resource-Intensive Jobs

If you run a resource-intensive program on any CS server other than `compute.cse.tamu.edu`, your CS account may be terminated for hogging the machine!

Libraries and Other Resources

You *may* use libraries for handling large numbers and performing basic mathematical operations on them. You *may* use libraries for manipulating image data. You *may* consider using/contributing to to Virtual Steganographic Laboratory (http://vsl.sourceforge.net/) or Digital Invisible Ink Toolkit (http://diit.sourceforge.net/) projects. You *may* use libraries for unit testing. For now, that is it. If there is a library or resource you want to use and it is not listed here, you must ask Dr. Ritchey about it before you may use it. Do not ask about using libraries or code that solve the main problems of this project, the answer will be "No".

GitHub

You must use GitHub. Your repository must be set up and active before April 12th. The TAs and the Instructors must have read access to your repository by April 12th. Your repository must be private until May 5th. Your repository must be public after May 11th.

Final Report and Presentation

The board of Munchkin Incorporated expects to receive a final report of your work and a brief presentation demonstrating the tools you built. The report should include details about how you solved each problem in this project, especially with regard to Test-Driven Development and Agile Programming. Your report also should also include the division of labor, specifying who did what and the value of that contribution to the overall project. Attached to the report, you should submit copies of your weekly Sprint progress reports, complete with Backlogs, Burndown charts, and Sprint Status Checks. Your presentation during the final exam should last no more than 10 minutes and should clearly demonstrate your usage of Test-Driven Development and Agile Programming, as well as the correct operation of your tools. Stay true to the Agile methodology, do not submit or demo something which is not yet finished.

Individual Scores

Your individual score will be computed as $\min\left(teamScore \times \sqrt{\frac{yourContribution}{100/|group|}}, 110\right)$. The team score will be calculated by the rubric given above. Your contribution will be computed as the average of your contribution percentage as given in the report and your contrubution percentages as reported by your team members in the post-project survey.

Need More Information?

This project specification is by no means intended to instruct you on how to implement the tools required by this project. You will need to attend lecture, ask questions, and do some independent study is order to complete this project. If you feel like there is some piece of information you are missing and cannot find it in this document, have not learned it in lecture, do not understand the material you have found online or in a textbook, or do not even know where to look to find it, go to Piazza and ask your question there.

Important Legal Notice

The techniques you are learning in this course and project are for educational purposes only and are not to be used for any unethical purposes. Please be responsible and use your powers for good.