Michael Reilly

CS-306B

Professor Triandopoulos

11/9/2021

I pledge my honor that I have abided by the Stevens Honor System.

Problem 1: Shared or forgotten keys?

1). Which two basic security properties should be considered in the design of a secure protocol for solving the above problem and why these properties become relevant in this setting?

The two basic security properties that should be considered when designing the secure protocol for Alice and Bob are confidentiality and integrity. Confidentiality is important because within this insecure channel it will still ensure that only Alice and Bob are able to read the messages being sent between them. This makes it so no outside user can read the secret key they are discussing with each other. Integrity is important because within this insecure channel it will still ensure that only Alice and Bob are able to modify the messages they send to one another, and they cannot be altered by an outside source. No outside attacker will be able to create confusion between Alice and Bob by changing the messages and create a severe miscommunication over whether the keys they have are correct or not.

2). Suppose that Alice and Bob use the following protocol to check if they store the same secret. 1. Alice generates a random n-bit value r. 2. Alice computes $x = ka \oplus r$, and sends x to Bob. 3. Bob computes $y = kb \oplus x$ and sends y to Alice. 4. Alice compares r and y. If $r = y$, she concludes that $ka = kb$—that is, Alice and Bob share a secret key. Does the above protocol satisfy the two security properties identified in question (1)?

The above protocol satisfies confidentiality but does not satisfy integrity. It satisfies confidentiality because only Alice and Bob will be able to translate the XOR messages sent between them, x and y. However, this does not satisfy integrity because there are no protections against outside attackers being made, thus an attacker can alter x and/or y, making it so the messages Alice and Bob send are incorrect and create a miscommunication between them of whether they both actually have the same key or not.

Problem 2: Perfect or imperfect ciphers?

1). Assume that an attacker knows that a user's password is either p1 = abcd or p2 = bedg. Say the user encrypts his password using the Vigen´ere cipher, and the attacker sees the resulting ciphertext c. Show how the attacker can determine the user's password, or explain why this is not possible, when the period t used by cipher is 1, 2, 3, or 4 respectively.

When encrypting by the Vigenere cipher, the attacker also learns the ciphertext to the plaintexts, which in this case the plaintext is either abcd or bedg. With both the ciphertext and both possible plaintexts the attacker will also be able to determine the keyword for any length as the keyword for a Vigenere cipher is generated by having a key of length t, so either t=1, 2, 3, or 4, which will then generate a keyword that is the same length as the plaintext, which in this case is known to be 4. For example, if the key has t=2 and is AB, thus the keyword is ABAB for a password known of length 4, as we know. Using this process the attacker could brute force until the key length is found based on the values of the Vigenere square, as the attacker knows it's between the two possible passwords. After the key length is known, the attacker then just needs to do a frequency analysis based around the Vigenere square to get which of the passwords it is as the values from the Vigenere square are deterministic. The distances between the two possible

passwords, and the corresponding ciphertext will always be the same, and thus can create a possible key of any of the four lengths. Thus by finding key length, and frequency analysis based on reversing the usage of the Vigenere square the attacker can find the user's password for all t values from t=1, t=2, t=3 and t=4, similarly to a chosen plaintext attack, with those chosen plaintexts being both possible passwords.

2). Show that the mono-alphabetic substitution cipher is trivial to break when the attacker launches a chosen-plaintext attack. How much chosen plaintext is needed to recover the entire secret key? What is the shortest chosen single-message plaintext that you can find, which is a valid English message and would successfully recover the key? Finally, under which conditions, and why, is the mono-alphabetic substitution cipher perfectly secure (against a ciphertext-only attacker)?

The mono-alphabetic substitution cipher is trivial to break when the attacker launches a chosen-plaintext attack, because mono-alphabetic ciphers are deterministic, they will always give the same cipher text back for a given plaintext due to the cipher alphabet created by the key and the open alphabet. So if the attacker chooses a plaintext to send to the cipher alphabet, the attacker can get the guaranteed ciphertext. Or, in the case of sending the open alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ as the plaintext, it will send back the cipher alphabet. This is because, let's say the key is BOARD, the cipher alphabet will become BOARDCEFGHIJKLMNPQSTUVWXYZ. In this case, the attacker sending the open alphabet will return the ciphertext as the cipher alphabet, and from there, seeing that the only letters out of order are BOARD, the attacker can determine the security key to be BOARD. Thus the mono-alphabetic substitution cipher is trivial to break with a chosen-plaintext attack. In order to recover the entire secret key, 25 letters of the plaintext must be known, as based on the ciphertext

returned every letter can be determined in order, with the 26th letter being whatever is left over that was not found from the corresponding ciphertext. The shortest type of single-message plaintext to successfully find the key would be a pangram, except with one letter missing, as that can just be determined by default after every other letter is found. For example, The quick brown fox jumps over lazy dog, with the letter “a” being missing in this case as the part of the cipher alphabet that will be learned by default after this message is sent, as every other of the 25 letters of the alphabet are in the message. This would make it trivial from here to successfully find the key as the full cipher alphabet but the first letter will be sent back, and the first letter will be whatever letter is leftover and not in the returned ciphertext. The monoalphabetic substitution cipher is perfectly secure against a ciphertext-only attack when the key is randomly generated for each letter within the plaintext. If the key is random, including a random length, then it cannot be predicted by just a ciphertext attack, a plaintext attack would need to be performed to find what the key is in this case.

Problem 3: Crypt-analyze this!

1). Below are eleven ciphertexts (in hex format) that they exchanged just minutes before our class last week, on Tuesday, October 19, 2021: Write down the 11 plaintext messages that were exchanged. You may write a program that will help you with your cryptanalysis. In designing your program, remember that most likely spaces will be among the most frequent characters in the plaintexts, and carefully observe what their role may be in the mapping from plaintexts to ciphertexts. Explain what your cryptanalysis strategy is and what algorithm your program implements.

My cryptanalysis strategy was to XOR all the ciphertexts together to get a possible key between them. This was found by doing bitwise XOR between the ciphertext hex values. This was then used to keep count of the appearances of plaintext characters that could possibly be within the text based on the given message space. These XORed values would then do another bitwise XOR with a text of just spaces in order to find possible spaces in the text, as it was very likely to appear many times within the messages. This is known as “crib dragging.” I was making the guess of spaces, and from that were able to get parts of the plaintext to create a possible key. It is best to check where the spaces appear in messages as they tend to appear often and are more easily distinguishable from the rest of the ASCII letters due to their value. On top of that, this is an 11-time pad by finding all the spaces and using them as 1 character cribs which can then find the text of all the messages much easier than just XORing the ciphertexts on their own. From there all that needed to be done was to XOR this key with one of the ciphertexts in order to give a possible ciphertext. When done against the very first ciphertext this gave “?esting ?est?ng can you read t?i>”, which I safely assumed was “testing testing can you read this”. Through getting the message “testing testing can you read this” I got the key by XORing the hex of this message with it’s ciphertext to get the key, which in ASCII is “Youfoundthekey!congratulations!!!” Then, XORing the hex of this key gives the text for all the messages.

The message exchange is as follows:

testing testing can you read this

yep I can read you perfectly fine

awesome one time pad is working

yay we can make fun of Nikos now

i hope no student can read this

that would be quite embarrassing

luckily OTP is perfectly secure

didnt Nikos say there was a catch

maybe but I didnt pay attention

we should really listen to Nikos

nah we are doing fine without him

2). To ease key management, my TAs change their shared key only every midnight. But rather than randomly generating (and securely exchanging) a new key every midnight, the TAs created an algorithm to automatically generate the new key pseudorandomly using the current (i.e., previous day's) key as input. That is, they replace the current key ki with the new key ki+1 = SHA256(ki)||00100001, where k denotes concatenation. What is the key the TAs will be using the day that this homework is due?

Based on how they are replacing the current keys, the TAs will just be the current key, with a single character concatenated to it. In order to find the character for "00100001" I used my program to convert the byteArray to ASCII. This gives the character "!". Thus the key will be the current key, as well as 27 more exclamation points ("!"). Thus the new key will be "Youfoundthekey!congratulations!!!!!!!!!!!!!!!!!!!!!!!!!!!!" According to my program that is the correct number of extra exclamation points on top of the existing three that were at the end of the original key, making it 30 exclamation points total at the end of the final key that they will have on November 12th.

README instructions for the program.

Go into the folder within the command line, with the contained index.js, package.json and package-lock.json files. Does not contain the corresponding node-modules folder as this will give you errors, but thus you may need to manually install dependencies.

Type npm start into command line, if this says a dependency is missing it must be installed using npm install {dependency}.

Once those dependencies are installed, once again type npm start, and the program will run smoothly and will log various checks I made in the program, including all the messages and the key.

Works Cited

Samer MakarySamer Makary 90311 gold badge88 silver badges77 bronze badges, et al. “How Does One Attack a Two-Time Pad (I.e. One Time Pad with Key Reuse)?” Cryptography Stack Exchange, 1 June 1960, https://crypto.stackexchange.com/questions/2249/how-does-one-attack-a-two-time-pad-i-e-one-time-pad-with-key-reuse.