Cryptography & Secure Development Coursework Report

1. Known Plain Text Attack (KPA)

1.1. Result

Key: captain

Decoded Message: This is the plaintext (task 1) for you HAET NIRAV with the ID number = 27713547437 to decode. Good luck!

1.2. Approach

The Known Plaintext Attack approach begins with the given ciphertext and the first 2 characters “Th” of the plaintext. A *knownPlainTextDecrypt* method of the class *KPA.java* takes ciphertext, the first 2 letters “Th” of the plaintext, and the passwords file which contains 9473 keys as input. This method iterates through all the keys present in the password file, takes the first 2 letters of the ciphertext, and decodes the message by using the *encdec* method of the provided *Rotor96Crypto.java* file, which is responsible for taking mode, ciphertext, and the key as input and it either encrypts or decrypts the plaintext based on the mode (mode=1 is encrypting and mode=2 is decrypting) given. The decoded message is then passed through a condition that checks the decoded message matches with the string “Th” and stores the first 2 letters of the plaintext that starts with “Th” in the HashMap java collection *resultKPA* as a pair of key and its decrypted message. Once the code iterates through all the keys, then the results stored in the HashMap *resultKPA* which contains a pair of key and its 2 letters of the decoded message are then stored in the new HashMap as a pair of key and its complete decoded message. The *main* method is responsible for displaying the result as the key and its decoded plaintext. The result obtained is the single key and its decoded message as mentioned above for the given ciphertext.

Estimation of how likely the probability of getting 0, 1, or more keys, knowing that the first 2 letters are “Th” from an English message and there are about 9473 possible keys, can be done by using the binomial distribution.

The probability of a string starting with “Th” is $\frac{1}{96} * \frac{1}{96} = \frac{1}{9216}$. As there are a total of 96 possibilities of the characters. So, p = $\frac{1}{9216}$

The probability of finding n keys starting with “Th” is $p(r) = {}^{n}_{r}Cp^{r}(1-p)^{n-r}$

The probability of finding 0 key starting with “Th” is $p(r=0) = {}^{9473}_{0}Cp^{0}(1-p)^{9473-0} = \left(\frac{9215}{9216}\right)^{9473}$ = 0.357

The probability of finding 1 key starting with “Th” is $p(r=1) = {}^{9473}_{1}Cp^{1}(1-p)^{9472} = 9473 * \frac{1}{9216} * \left(\frac{9215}{9216}\right)^{9473}$ = 0.364

The probability of finding more than 1 keys starting with “Th” is $p(r>1) = 1 - p(r=0) - p(r=1) = 0.279$

Therefore, the probability of finding 1 key and more than 1 keys starting with “Th” is 36% and 28% respectively.

2. Ciphertext Only Attack (COA)

2.1. Result

Key : mona

Decoded Message: This is the second task (Ciphertext Only Attack) of the assignment of the Cryptography and Secure Development course. HAET NIRAV - you are expected to decode it with the assumption that this plaintext is English sentences. This plaintext contains this random number 4058 so that you cannot guess it from the others :) Finger-crossed!

The number of ciphertext letters required to decode the encrypted text: 4

2.2. Approach

The Ciphertext Only Attack approach uses the n-grams technique to decode the encrypted message where the *cipherTextOnlyAttackDecrypt* method of the class *COA.java* takes the given ciphertext and passwords file which contains 9473 keys as input. This method fetches all the dictionary words consisting of uni-grams, bi-grams, tri-grams, and n-grams by calling the *getCommonWords* method, and stores the total length (333) of the given ciphertext with the help of the *length* method. Then it iterates through all the keys in the passwords file to store them in the Java List Collection named as *keys* to easily remove keys from the list which generates gibberish plaintext. Keys are then iterated through all the lengths of the ciphertext that begins with ciphertext length = 2 and decodes the message of the current length of the ciphertext as *cipherText.substring(0, i)* by using the *encdec* method of the provided *Rotor96Crypto.java* file, which is responsible for taking mode, ciphertext, and the key as input and it either encrypts or decrypts the plaintext based on the mode (mode=1 is encrypting and mode=2 is decrypting) given. The decoded message is then passed to the *containsGibberish* method which checks whether the given plaintext comprises English letters, digits, and spaces and returns *True* for gibberish plaintext else *False* for non-gibberish plaintext. Keys are then removed with the help of *removeKeys* List from the List *keys* whose generated plaintext contains gibberish. The decoded message that does not contain gibberish is then split based on the space delimiter and all the letters are converted into lowercase to easily match with the dictionary words. The words obtained after splitting are stored in the Java List collection *decryptedValues* which is then passed to the java filter function to match the words with the dictionary words and store the count of the matched words as a score. The score obtained is then checked with the best score, if it is greater than the best score then the best score will be that score, and its key and score are stored in the HashMap Java collection *resultCOA*. Once, all the keys are iterated for the specific length of the ciphertext, the results obtained are then passed through a condition that checks whether the single pair of key-score is present or not. If there are multiple pairs of key-score, then it empties the HashMap *resultCOA* of the result and starts the iteration again with the next length of the ciphertext until it gets a single pair of key-score. Once, a single pair of key-score is obtained it is stored in the *result* HashMap as a pair of key-plaintext by passing the key and the entire ciphertext to the *encdec* method to get the complete plaintext, then the length of the ciphertext for that round is taken as the number of ciphertext letters required to decode the encrypted message unambiguously. It is then displayed along with the result stored in the HashMap as the key and its decoded plaintext through the *main* method. The decoded message obtained with the key has the highest score which means that the large number of words from the plaintext got matched with the dictionary words this indicated that the potential plaintext message is correct. The result obtained is the number of ciphertext letters required to decode the encrypted message unambiguously, a single key, and the decoded message as mentioned above for the given ciphertext.

From my experiment, the actual number of ciphertext letters required to decode the encrypted message unambiguously is 4. The theoretical calculation of the number of cipher text letters needed before unambiguous decoding can be done by calculating the unicity distance.

Unicity distance represents the minimum number of ciphertext letters required to be sure that a good, decrypted message is the real one. The formula is :-

$$N_u = \frac{H(K)}{D}$$

Where, N_u is the Unicity Distance, H(K) is entropy, and D is the redundancy of the language.

For English, D = 3.2

Entropy :- H(K) = $log_2(N)$ where N is the Number of keys in the key space

In our case, N = 9473 keys present in the passwords file.

Entropy: $H(K) = log_2(9473) = 13.209$

The Unicity Distance: $N_u = \frac{13.209}{3.2} = 4.127$

So, according to theoretical calculations we need to decode about 5 letters to make sure that a message that looks like English is the real message. The difference between the theoretical calculation and the actual experimental calculation for the number of encryption letters needed to decode the encrypted message depends on various factors mentioned below:

1. **Computational limitations:** The theoretical unicity distance assumes that an attacker has the unlimited computational power to perform an exhaustive search of all possible keys and plaintexts. However, in practice, the attacker may be limited by the computational resources available, such as time and memory, which can significantly affect their ability to perform a brute-force attack. Therefore, the actual unicity distance may be shorter than the theoretical value.
2. **Keyspace quality:** The theoretical unicity distance assumes that the keys used in the cryptosystem are generated randomly and uniformly. However, the keys used in our case is limited to 9473, which does not represent all the possible keys.
3. **Statistical properties of plaintext:** The theoretical unicity distance assumes that the plaintext is distributed uniformly and randomly. However, in practice, some plaintext may have patterns or structures that can be exploited by the attacker, such as repeated words or phrases, which can reduce the unicity distance. In our case, there is a name of the user mentioned in the plaintext which makes it easy for the attacker to decrypt the encrypted message.