**CS: 419 – Computer Security: HW-1**

*Himesh Buch*

*2/14/2020*

1.

Known Ciphertext: NBZQARAV KTAH ETQ CJII STL WLHQ MZCAH STL HQVZEOAV

Corresponding Plaintext: Whatever does not kill you just makes you stranger

By comparing the above two strings we realize which characters can be replaced with which other characters. So,

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Letters | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z |
| key | z | ? | ? | k | a | ? | o | b | j | w | c | i | m | e | t | ? | ? | v | h | q | l | ? | n | ? | s | ? |

Now using our partial key, we will try to decrypt the below message:

Message: IAQ LH MZCA VLQOAVH OVAZQ ZOZJE

Decrypted message: JZ? IB M??Z ?I?TZ?B T?Z?? ?T?WA

This is the best we could get by the information provided. Note that we still don’t know what some of the missing characters in the plaintext correspond to and hence we will not be able to completely decrypt the message. (The letters that are not known are replaced with ‘?’ in the decrypted message)

2. The algorithm basically, encrypts the message and returns the encrypted message where the characters are shifted by one unit. Since, it is not the right message, uploading it to oracle won’t give the right plaintext, as the characters are shifted.

Now instead of uploading the cipher text c\* directly to the oracle by just using the encryption scheme E(.,.), we run the algorithm S on the message m and get c\*’ which is not the original cipher text and upload that to the oracle which will return m’, which is not the original plaintext but assembly, a plaintext where characters are shifted by one unit. We will, in the end, shift one unit back in the returned plaintext m’ and that will give us the original plaintext m in return.

By this way, instead of uploading c\* directly to the oracle we uploaded another version of it and shifted some elements of the returned message m’ to retrieve original plaintext m.

3.

1) The likelihood that two rehashing strings in the ciphertext happening from two diverse plaintexts is very uncommon. The explanation behind this is the key has relative "spacing" between its characters. For two distinct arrangements of plaintext characters to have "integral" spacing and to still make sense is very far-fetched.

Presently, envision utilizing the piece of the key UCK rather than LUCK. The relating plaintext letters, when added to UCK that would yield different ciphertext which are most likely to be incorrect and might not be an English word. Truth be told, we are probably going to find that on the off chance that we arranged the ciphertext under some other bit of the keyword than LUCK, that the comparing plaintext would not be proper English.

In this way, on the off chance that we might also see some characters in the ciphertext repeat. Hence, we proved that the probability of two 3-letter strings having different plaintexts for a given Vigenere cipher is 0 because that is highly unlikely to happen, as explained above

2) we can also have a case where the ciphertext might contain the text that is repeated. Assuming that some text ‘XYZ’ is repeated at indices 110 and 138. The difference is 28. If we repeat the process, we might find an occurrence at indices 85 and 128, where the difference is 43.

Using above numbers, we get:

56 = 1\*28 + 15

28 = 1\*20 + 8

20 = 4\*5 + 0, so the gcd here is 8

Above example explains the concept of Kasisky test, which is essentially finding the lengths between pairs of repetitions of the ciphertext, and afterward taking the greatest common divisor of every one of these qualities, since the keyword length is probably going to partition into every one of these qualities. Hence, explained the concept of gcd in Kasisky test.

4.

probability of P(D1=0) = P(D1=1) = P(D1=2) = P(D1=3) = P(D1=4) = P(D1=5) = 1/6

probability of P(D2=0) = P(D2=1) = P(D2=2) = P(D2=3) = P(D2=4) = P(D2=5) = 1/6

(According to the slide, random variable D1 denote the outcome of throwing one die (with numbers 0 to 5 on the 6 sides) randomly, then D={0,1,2,3,4,5} and Pr[D1= i ] = 1/6 for 0≤ i ≤ 5)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| D2 | D1 | | | | | |
| 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | 0 | 1 | 2 | 3 | 4 | 5 |
| 1 | 1 | 2 | 3 | 4 | 5 | 6 |
| 2 | 2 | 3 | 4 | 5 | 6 | 7 |
| 3 | 3 | 4 | 5 | 6 | 7 | 8 |
| 4 | 4 | 5 | 6 | 7 | 8 | 9 |
| 5 | 5 | 6 | 7 | 8 | 9 | 10 |

(According to the slides, random variable S1 denote the sum of the two dice, then S ={0,1,2, …,10})

Now, probability of S1 => P(S1 = 0) = P(S1 = 10) = 1/36

P(S1 = 1) = P(S1 = 9) = 2/36 = 1/18

P(S1 = 2) = P(S1 = 8) = 3/36 = 1/12

P(S1 = 3) = P(S1 = 7) = 4/36 = 1/9

P(S1 = 4) = P(S1 = 6) = 5/36

P(S1 = 5) = 6/36 = 1/6

(According to the slides, random variable S2 denote the sum of the two dice modulo 6)

Now, probability of S2 =>

P(S2 = 0) = 0, P(S2 = 1) = 1, P(S2 = 2) = 2, P(S2 = 3) = 3, P(S2 = 4) = 4, P(S2 = 5) = 5, P(S2 = 6) = 0, P(S2 = 7) = 1, P(S2 = 8) = 2, P(S2 = 9) = 3, P(S2 = 10) = 4

* 1)

Here we multiply the probability of the dices (D1) which are all 1/6, by each value in S1

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| D1 | S1 | | | | | | | | | | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 0 | 0.0046 | 0.0093 | 0.0139 | 0.0185 | 0.0231 | 0.0278 | 0.0231 | 0.0185 | 0.0139 | 0.0093 | 0.0046 |
| 1 | 0.0046 | 0.0093 | 0.0139 | 0.0185 | 0.0231 | 0.0278 | 0.0231 | 0.0185 | 0.0139 | 0.0093 | 0.0046 |
| 2 | 0.0046 | 0.0093 | 0.0139 | 0.0185 | 0.0231 | 0.0278 | 0.0231 | 0.0185 | 0.0139 | 0.0093 | 0.0046 |
| 3 | 0.0046 | 0.0093 | 0.0139 | 0.0185 | 0.0231 | 0.0278 | 0.0231 | 0.0185 | 0.0139 | 0.0093 | 0.0046 |
| 4 | 0.0046 | 0.0093 | 0.0139 | 0.0185 | 0.0231 | 0.0278 | 0.0231 | 0.0185 | 0.0139 | 0.0093 | 0.0046 |
| 5 | 0.0046 | 0.0093 | 0.0139 | 0.0185 | 0.0231 | 0.0278 | 0.0231 | 0.0185 | 0.0139 | 0.0093 | 0.0046 |

* 2)

Here we multiply the probability of the dices (D2) which are all 1/6, by what we got in S2

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| D2 | S2 | | | | | | | | | | |
| 0-Jan | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 0 | 0.0000 | 0.1667 | 0.3333 | 0.5000 | 0.6667 | 0.8333 | 0.0000 | 0.1667 | 0.3333 | 0.5000 | 0.6667 |
| 1 | 0.0000 | 0.1667 | 0.3333 | 0.5000 | 0.6667 | 0.8333 | 0.0000 | 0.1667 | 0.3333 | 0.5000 | 0.6667 |
| 2 | 0.0000 | 0.1667 | 0.3333 | 0.5000 | 0.6667 | 0.8333 | 0.0000 | 0.1667 | 0.3333 | 0.5000 | 0.6667 |
| 3 | 0.0000 | 0.1667 | 0.3333 | 0.5000 | 0.6667 | 0.8333 | 0.0000 | 0.1667 | 0.3333 | 0.5000 | 0.6667 |
| 4 | 0.0000 | 0.1667 | 0.3333 | 0.5000 | 0.6667 | 0.8333 | 0.0000 | 0.1667 | 0.3333 | 0.5000 | 0.6667 |
| 5 | 0.0000 | 0.1667 | 0.3333 | 0.5000 | 0.6667 | 0.8333 | 0.0000 | 0.1667 | 0.3333 | 0.5000 | 0.6667 |

* 5) D1 and S1 are independent
* 6) D1 and S2 are independent

5. Casually, if a key is shorter than the message, that implies the key must be reused some place (for an OTP). At the point when it's reused, aggressors can misuse the rehashed examples to decode the message snappier. For example, a key of 3 characters, and assuming it is smaller than the original text, then we will have to repeat the key, and if we use the keyword ‘key’, then the keystream will be keykeykeykeykey... This means that every third letter is encrypted using the same shift, which could be a big flaw which makes decryption easier.

On the other hand, an "arbitrary" key that is k bits long has k bits of entropy. When that key is utilized to encode a string those k bits are basically blended in with the bits of the message to expand its entropy with the proviso that a string can never contain a bigger number of bits of entropy than its length.

So, in the event that you have a message of length m and a key of length k then the entropy of the encoded message is in any event min(m, k). This implies if your message is 18 bits shorter than your key you should have a key that is 18 bits shorter since it is, from a data theoretic outlook, failing to help you.