COSC312: Assignment 1

Solutions

1. *The inhabitants of* Cryptologia *use a ten character alphabet where the characters are the digits 0, 1, 2, . . . , 9. This makes implementing Vigenère ciphers rather simple—you just take a sequence of digits, repeat it as often as necessary and add it to the text, discarding any carries. The only statistical non-uniformities that have been observed in (unencrypted) Cryptologian texts are that successive letters are never the same. That is, the pairs 00, 11, 22, . . . , 88, and 99 never occur as consecutive letters. In fact, any sequence of digits that does not contain a repeated pair like this is a valid Cryptologian text.*

   (a) *How would you propose determining the likely key length for a Vigenère cipher based on Cryptologian from a ciphertext?* 2

   Discussion: Starting from the methods normally used for English text, we might consider the Friedman and Kasiski examinations. However, the Friedman examination seems unpromising since the only non-uniformity in Cryptologian is the lack of repeated pairs. In particular, every letter occurs with equal frequency among all possible texts, so the Friedman test that makes use of the non-uniformity of letter frequencies is not appropriate. The Kasiski examination seems more promising – especially one based on digrams (i.e., two character sequences) since these have a non-uniform distribution in Cryptologian text. Specifically, the 90 digrams of two distinct characters occur with equal likelihood, while the 10 repeated character digrams never appear at all. This approach is discussed below (it has some limitations on relatively short messages).

   Proposal: Instead, we consider starting from scratch and ask the question: how could we rule out any particular key length for certain? The simplest case is this: we can rule out a key length of one if there are repeated digits (and if there aren't we can't!) What about a key length of two? Suppose the key length were two. Then the "no repeated digits" condition on the plain text would imply that a certain difference between the 0th and 1st characters of the ciphertext was impossible (specifically, the difference that is the same as the difference between the 0th and the 1st digits of the key). The same difference would be ruled out between the 2nd and 3rd, 4th and 5th, . . . characters. Turning this around – if we observed all 10 possible differences in those positions then we could rule out a key length of two. Even if this does not work we would also need to check the 1st and 2nd, 3rd and 4th, 5th and 6th . . . difference since the same condition would apply there (some difference must be unobserved).

   The same argument applies to general key lengths. Given a proposed key length k we consider the consecutive pairs in positions congruent to i and $i + 1$ modulo k in the ciphertext. If the key length is

possible there must be some difference between such pairs that does not occur (the difference that corresponds to the difference between the corresponding elements of the key). If, for any i, we observe all possible differences then we can rule out the key length. If we don't then we can't.

On Kasiski To carry out a Kasiski examination we'd compute the gaps between all repeated digrams and look for numbers that are common factors of those gaps more frequently than we'd expect. This is hampered somewhat in that 90 of the 100 possible digrams occur naturally and with equal frequency and so there are more accidental collisions than there would be in English. However, for longer texts this seems to work fairly well – particularly when we multiply the number of occurrences of gaps of length g by g to account for the fact that "on average only 1 out of g numbers are multiples of g" (if we don't do this we overweight e.g., even length gaps with $g = 2$ relative say to gaps of whose lengths are a multiple of 5). Some of this is done in my `cryptologia.py` code and I'll discuss it in the tutorial as well.

(b) *If you know the key length, can you ever be certain about what the key is? Why, or why not?*

One property of Cryptologian text is that any shift of all the characters in a message by a fixed amount creates another valid text. But, if we think of encoding a given text with a key, versus first shifting it and then encoding with that key, the effect will be the same as encoding it with a shifted key. In other words, if a key k is valid then so is any key k' where $k'_i = k_i + c$ for some constant c. This means that, if we know the key length, we can never be certain about what the key is.

(c) *You will receive by email a message encoded from Cryptologian using a Vigenère cipher. Try to determine the key length and as much information as possible about the key. Explain your methods and submit any program you used.* 3

Explanation: The basic technique is outlined above. In this case that turns out to rule out all possibilities except a key length of 11. Then to determine the key we can look at the "possible differences" allowed. There are two possible between the 0th and 1st positions, and only one elsewhere. However, there's a hidden additional condition that requires the sum of all the differences as we cycle through the entire key to be 0. To see this consider the shorter example of a three-digit key, say *abc*. Then the differences are $a - b$, $b - c$ and $c - a$ and these add up to 0 (or more generally to something that's 0 mod 10 if we make some modifications e.g., replacing -2 by 8). That observation leaves only one possible key (up to the shift that we can't rule out) and one version of this key is: 31415926535 (which, if you noticed it, was intended to be an indication you might be right!)

See `cryptologia.py` for the code used in various parts of this question.

2. *Neither of the following pseudo-random generators are secure (for fairly trivial reasons). In each case, demonstrate this fact by giving an efficient statistical test with a significant advantage over the generator.*

   Note: An *efficient statistical test* in this context is just a map from binary strings to a two-element set (let's think of this as red and blue) that can be efficiently computed. It demonstrates the insecurity of a generator if the proportion of outputs of the generator that would be coloured red differs significantly from the proportion of all strings that would be coloured red. In probabilistic terms this means that the probability that an output of the generator will be coloured red differs significantly from the probability that a truly random string will be coloured red.

   (a) $G : \{0,1\}^{24} \to \{0,1\}^{48}$ *where* $G(k)$ *is just two copies of* k *concatenated together.*

   Test: Colour a string red if the first half of the string is the same as the second half.

   Analysis: All outputs of the generator will be coloured red. On the other hand, a truly random word is coloured red only if the coin flips producing its final 24 digits are the same as the coin flips producing its first 24 digits. The probability of this is 2^{-24}, which is negligible.

   (b) $G : \{0,1\}^{63} \to \{0,1\}^{70}$ *where* $G(k)$ *is* k *concatenated with the seven-bit binary representation (including leading zeros) of the number of* 1*'s in* k*.*

   Test: Colour a string red if the last seven bits of the string encode the number of 1's in the first 63 bits of the string in binary.

   Analysis: All outputs of the generator will be coloured red. On the other hand, a truly random word is coloured red only if the number of 1's in the first 63 bits of the string is the encoded in binary in the last seven bits of the string. The probability of this is 2^{-7}.

3. *Suppose that* $G : \{0,1\}^s \to \{0,1\}^n$ *is a secure pseudo-random generator. Which, if any, of the following modifications of* G *is also secure? Explain your answer in each case. A sentence or two will suffice.*

   (a) $H(k)$ *which is defined from* $G(k)$ *by taking the exclusive-or of* $G(k)$ *and the sequence* 010101... *of alternating* 0*'s and* 1*'s.*

   Intuition and discussion: The output of H is determined from that of G by negating every second element but the output of G can also be determined from that of H. This makes it feel like they are equivalent in some sense and suggests that H should be secure. To make this formal we need to think about how we could prove a generator to be secure. As noted in class, it's essentially infeasible to actually prove that a generator is secure (since this would resolve $P \neq NP$ which is rather beyond the scope of this assignment). So, we'll need an indirect method and the only lever we have is that we are allowed to assume that G is secure. This suggests the strategy: argue that, if H is insecure, then G must also be insecure.

   Solution: We claim that H is secure. Suppose to the contrary that there were some efficient statistical test, T, that demonstrated otherwise. Then consider the following test against G: given a string w compute the exclusive-or of w and 010101... and then apply T to that. This colours exactly the same proportion of outputs of G read as T colours outputs of H red, and exactly the same proportion of truly random strings red as T colours truly random strings red (since taking the exclusive-or of a random string against a fixed string produces a random string). So, if T is a good test against H then this is a good test against G. This contradicts the assumption that G is secure.

   (b) $H(k)$ *which is defined from* $G(k)$ *by appending the exclusive-or of the bits in* $G(k)$ *(that is,* $H(k) = G(k)0$ *if the number of* 1*-bits in* $G(k)$ *is even, and* $H(k) = G(k)1$ *if the number of* 1*-bits in* $G(k)$ *is odd).*

   This is insecure since the last bit of an output of H is determined from the previous bits. Specifically:

   Test: Colour a string red if the last bit of the string is the exclusive-or of the other bits in the string, i.e., if the total number of 1's in the string is even.

   Analysis: All outputs from H are coloured red, while truly random strings will be coloured red only 1/2 of the time.