UConn, School of Computing
Fall 2024
CSE 3400/CSE 5080: Introduction to Computer and Network Security / Introduction to Cybersecurity

Assignment 1

Instructor: Prof. Ghada Almashaqbeh
Posted: 9/9/2024
Submission deadline: 9/17/2024, 11:59 pm

Problem 1 [30 points]
Write a program that increments a counter $2^{10}, 2^{20}$, and 2^{30} times (so the initial counter value will be 0, then 1, 2, ..., up to 2^{10}, then repeat with 0, 1, ..., 2^{20}, and so on).

1. For each counter value, in each iteration compute the bitwise AND of the counter with the value 1254632 (so let the counter be $ctr = 0$, compute ctr & 1254632, then set $ctr = 1$ and compute that value again, and so on). Do that for $2^{10}, 2^{20}$, and 2^{30} counter values. Measure how many seconds your program takes to run in each case (list the machine specifications and the programming language you have used). Estimate how many years your program would take to do this operation when the counter is incremented 2^{480} times.

   My machine has an AMD R9 8945HS processor on Windows 11 using python. Time for 2^{10}: 0.0 seconds, 2^{20}: 0.07856202125549316 seconds, 2^{30}: 82.6774411201477 seconds. Estimated time for Part 1 (2^{480}): 89208.15517592432 seconds

2. Repeat the previous counter increments but now in each iteration compute the following formula: $ctr \oplus 990 + ctr/3456$ (so there is a bitwise XOR and a division operation). Measure how many seconds your program takes to run in each case (list the machine specifications and the programming language you have used). Estimate how many years your program would take to do this operation when the counter is incremented 2^{480} times.

   My machine has an AMD R9 8945HS processor on Windows 11 using python. Time for 2^{10}: 0.0 seconds, 2^{20}: 0.13591599464416504 seconds, 2^{30}: 139.3803141117096 seconds. Estimated time for Part 2 (2^{480}): 150382.65722608566 seconds

Problem 2 [40 points]
Let $G_0 : \{0,1\}^n \rightarrow \{0,1\}^{2n}$ and $G_1 : \{0,1\}^{n/2} \rightarrow \{0,1\}^n$ be PRGs. For each of the following constructions, prove whether it is a PRG or not. To prove insecurity you have to show an attack against the scheme and provide an informal argument about the attack success probability. For security provide a convincing informal argument why it is a PRG (with computing the probabilities as needed).

1. $G_2(s \,\|\, t) = G_0(s) \,\|\, (t \oplus 1^n)$

   The construction is NOT a PRG because of the vulnerability of $t \oplus 1^n$. Since t is not a truly random string, the attacker would easily be able to find t by bit flipping the second half of the output string, allowing the attacker to easily identify the PRG inside the output string (or the first half) thus the probability the attacker identifies the output coming from G_2 would be nearly 1 ($Pr[D(G(s)) = 1] \sim 1$). However, as there is no easily identifiable pattern, the attacker would have to guess so ($Pr[D(R) = 1] \sim .5$). $Adv(G) \sim 1 - .5 \sim .5$ thus non-negligible.

2. $G_3(s \parallel z) = G_0(s \oplus z)$

   The construction IS a PRG because of its unpredictability. The attacker, even if they knew s and z, would have no way to differentiate the string produced by the pseudorandom operation of G_0 from any other. The advantage, $Adv(G) = |\Pr[D(G(s)) = 1] - \Pr[D(R) = 1]| \sim .5 - .5 = 0$, is negligible.

3. $G_4(s \parallel z) = \mathsf{LH}(G_0(s)) \oplus G_1(\mathsf{RH}(z))$, where LH is the left half of the input string, and RH is the right half of the input string.

   The construction IS a PRG because of its unpredictability. G_0 is a random binary string of length $2n$, and G_1 is a random binary string of length n. $(G_0 \oplus G_1)$ would simply result in another random string of length $2n$, giving the attacker no clues. The advantage, $Adv(G) = |\Pr[D(G(s)) = 1] - \Pr[D(R) = 1]| \sim .5 - .5 \sim 0$ thus negligible.

4. $G_5(s) = (G_0(s) \mod 2) \parallel G_0(s)$, where mod is the modulus operation.

   The construction IS NOT a PRG because it is a recognizable pattern. The first and second half of the output string with length $4n$ would mirror each other because the modulus operation does nothing to G_0 since neither 0 or 1 have a remainder when divided by two, effectively not changing the string. This would allow the attacker to differentiate the PRG outputs, thus the advantage $Adv(G) = |\Pr[D(G(s)) = 1] - \Pr[D(R) = 1]| \sim 1 - .5 \sim .5$ thus non-negligible.

($\parallel$ is the concatenation operation, $\oplus$ is the bitwise XOR operation, s and z are random strings each of which is of length n bits, and t is any string—not necessarily random—of length n bits.)

Problem 3 [30 points]

Part 1: Answer the following for the monoalphabetic substitution cipher.

1. Assume you have a CTO attacker who is trying to perform the letter frequency attack we studied in class. Would the attacker be more successful in retrieving the key (the key is basically the permutation of the alphabet) if he gets a long ciphertext or a short one? why?

   The attacker would would be more successful the longer the cipher text because the longer the text, the more likely the text is going to match the general frequency that letters show up. For example, "E" is the second most common letter in the English language and appears in roughly 11 percent of words. If the cipher text is extremely long, the attacker could use this information to crack parts of the text.

2. Will your answer change for the previous question if we consider a CPA attacker? why?

   The answer does change because in a CPA attack the attacker can match plaintexts to corresponding cipher texts. The attacker would then have the necessary information to crack the cipher text regardless of length.

Part 2: Answer the following about OTP (just provide informal convincing arguments, no need for formal proofs).

1. Alice encrypted two messages of length n bits to Sponge using pads p_1 and p_2, respectively (producing two ciphertexts c_1 and c_2, respectively). She also posted the value $y_1 = p_1 \oplus x$ and $y_2 = p_2 \oplus x$ on her website and argued that these masked values are also random and will not reveal anything about the pads (and she did not tell anyone that value of x, however, everyone knows that the posted valued are the XOR of the pads with the same unknown value x). Is the scheme above secure against a CTO attacker? why?

   The scheme is not secure as the attacker could simply do $y_1 \oplus y_2$ which would cancel the x values out leaving the expression $y_1 \oplus y_2 = p_1 \oplus p_2$. All the attacker would now need is the information on one plaintext to crack the other one as well.

2. Eve wants to send two messages of length n bits to Charlie. She generated a random pad p of length n bits and sent two ciphertexts as follows: $c_1 = p \oplus m_1$ and $c_2 = (p + 1) \oplus m_2$, and she argues that this is secure. Is this secure against a CTO attacker? why?

   This is also not secure for more or less the same reasons as the previous question. The attacker would first xor the two expressions resulting with $c_1 \oplus c_2 = p_1 \oplus (p_2 + 1)$. The "+1" in (p+1) does nothing except change the LSB, so the attacker would still only need some information on one plaintext to derive information on the other.