Question 1

1. Consider a modification of the substitution cipher, where instead of applying only the substitution, we first apply a substitution and then apply a shift cipher on the substituted values. Give a formal description of this scheme and show how to break the substitute and shift cipher. Upto what message space size is your scheme perfectly secret?

Solution

Let's formalize the modified substitution cipher followed by a shift cipher:

Substitution Cipher: Let Es denote the substitution encryption function, where each letter of the plaintext is substituted with another letter according to a fixed substitution table. For example, 'A' with 'D', 'B' with 'E', and so on
Shift Cipher: Let Ek denote the shift encryption function, where each letter of the substituted ciphertext is shifted by a fixed amount k positions in the alphabet (modulo26), where k is the key of the shift cipher.
For example, if k=3, then 'A' becomes 'D', 'B' becomes 'E', and so on.
The overall encryption function E = Es ◦ Ek
This scheme is not perfectly secret. An attacker can break the scheme using frequency analysis. The message space size for which the scheme achieves perfect secrecy depends on the size of the key space for both the substitution cipher and the shift cipher. Note keyspace == message space doesnt work here.

Rubric

1. **4 marks**
    a. Formal definition of the composition, including Enc and Dec.
2. **4 marks**
    a. Formally specifying any statistical attacks.
3. **2 marks**
    a. **1.5 marks:** Maximum message length = 1
    b. **.5 marks:** Proving the above.

Question 2

2. **Negligible or not?** (in each of the following three cases): Let $f, g : \mathbb{N} \to \mathbb{R}$ be negligible functions. let $p : \mathbb{N} \to \mathbb{R}$ be a polynomial such that $p(n) > 0$ for all $n \in \mathbb{N}$. $3 + 3 + 4 = 10$

   1. $h : \mathbb{N} \to \mathbb{R}$ as $h(n) = f(n) + g(n)$.
   2. $h : \mathbb{N} \to \mathbb{R}$ as $h(n) = f(n) \cdot p(n)$.
   3. $f(n) := f'(n) \cdot p(n)$, for some negligible function $f'(n)$ and some polynomial function $p(n)$. Is such a $f(n)$ always negligible?

Solution

(2)

① → $h : \mathbb{N} \to \mathbb{R}$ $h(n) = f(n) + g(n)$

f is negligible means

$$|f(n)| < \frac{1}{q(n)}$$

g is negligible means

$$|g(n)| < \frac{1}{r(n)}$$

$$\Rightarrow |h(n)| = |f(n) + g(n)| \leq |f(n)| + |g(n)|$$

$$< \frac{1}{q(n)} + \frac{1}{r(n)}$$

if we choose $p(n) = r(n) \cdot q(n)$

$$|h(n)| < \frac{1}{p(n)} + \frac{1}{p(n)} = \frac{2}{p(n)}$$

which is indeed negligible.

(2) - 2

$n(n) = f(n) \cdot p(n)$

Given → f negligible

P_{poly} → polynomial

Goal: for $n > N_p$ $\quad n(n) < \frac{1}{p(n)}$

$|f(n)| < \frac{1}{p(n)}$ for $n > N_f$

$|f(n) \cdot p(n)| < \frac{1}{p(n)}$ for $n > N_f$

→ if we choose $N_p = N_f$ then

$|n(n)| < \frac{1}{p(n)}$

so n is negligible.

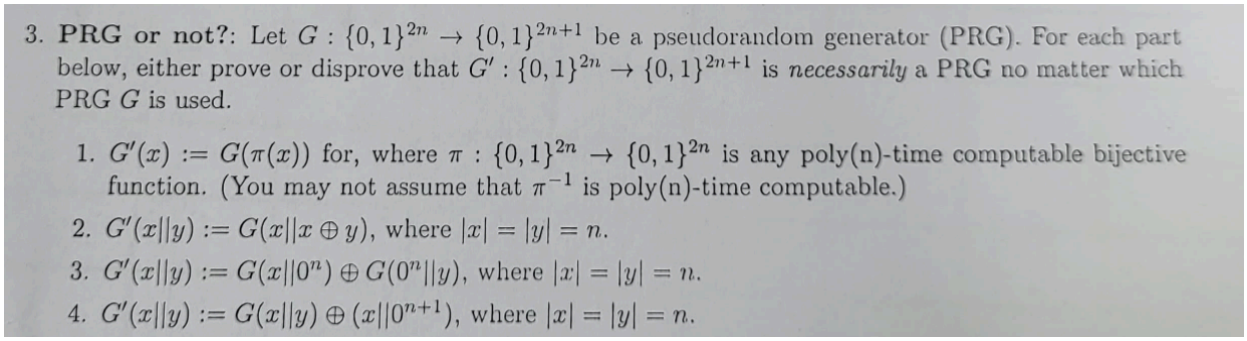
Rubric

a. **3 marks**
   i. **0.5 marks:** Negligible function

   ii. **2.5 marks:** Proof either using *limits* or from first principles.
2. **b. 3 marks**
   i. **0.5 marks:** Negligible function
   ii. **2.5 marks:** Proof either using *limits* or from first principles.
3. **c. 4 marks**
   i. **1 mark: Not Negligible function.**
   ii. **3 mark: Counterexample:-** p(n) < 0 for all n.

No marks are awarded if the reasoning is logically inconsistent and still the answer is correct.

Question 3



3. **PRG or not?**: Let $G : \{0,1\}^{2n} \to \{0,1\}^{2n+1}$ be a pseudorandom generator (PRG). For each part below, either prove or disprove that $G' : \{0,1\}^{2n} \to \{0,1\}^{2n+1}$ is *necessarily* a PRG no matter which PRG G is used.

   1. $G'(x) := G(\pi(x))$ for, where $\pi : \{0,1\}^{2n} \to \{0,1\}^{2n}$ is any poly(n)-time computable bijective function. (You may not assume that π^{-1} is poly(n)-time computable.)
   2. $G'(x||y) := G(x||x \oplus y)$, where $|x| = |y| = n$.
   3. $G'(x||y) := G(x||0^n) \oplus G(0^n||y)$, where $|x| = |y| = n$.
   4. $G'(x||y) := G(x||y) \oplus (x||0^{n+1})$, where $|x| = |y| = n$.

Solution

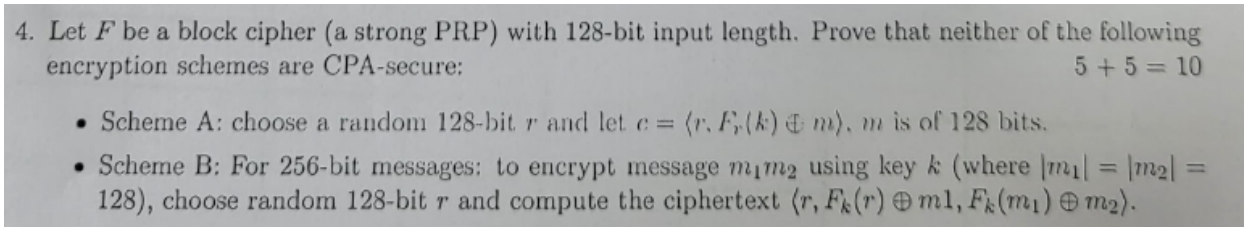
<Paste Solution here>

Rubric

1. **3 marks**
   a. **0.5 marks:-** PRG
   b. **1 mark:-** Explanation that bijective function *reorders* the distribution.
   c. **1 mark:-** Formal proof, bounding the probabilities of the Distinguisher
2. **3 marks**
   a. **0.5 marks:-** PRG
   b. **2.5 marks:-** Proof by first principles by before, or just showing that given mapping is a **bijection**.
3. **2 marks**
   a. **0.5 marks:-** Not PRG
   b. **Counterexample:-** For input x==0^n & y==0^n, create a deterministic distinguisher D'.
4. **2 marks**

a. **0.5 marks:-** Not PRG
b. **Proof by contradiction:-** Given a PRG H(x), consider (and prove) another PRG G(x) = x_1 || G(x2 x3 … xn). Then show that the first bit of G'(x) is deterministic.

Question 4



4. Let F be a block cipher (a strong PRP) with 128-bit input length. Prove that neither of the following encryption schemes are CPA-secure: $5 + 5 = 10$

- Scheme A: choose a random 128-bit r and let $c = \langle r, F_r(k) \oplus m \rangle$. m is of 128 bits.
- Scheme B: For 256-bit messages: to encrypt message $m_1 m_2$ using key k (where $|m_1| = |m_2| = 128$), choose random 128-bit r and compute the ciphertext $\langle r, F_k(r) \oplus m1, F_k(m_1) \oplus m_2 \rangle$.

.

Solution

1. The scheme is not cpa secure, it can be shown as
    a. Choose 2 plain text messages say m1 and m2, then, we can find their corresponding cipher texts as, c1 = <r,Fk(r)^m1> and c2 = <r,Fk(r)^m2>.
    b. If we simply xor the output cipher texts c1 and c2, the attacker can simply find out the XOR of m1 and m2, c1^ c2 = <0,m1^m2>. And thus the attacker learns some information about the messages m1 and m2, which could be used to differentiate between the plain texts, and thus violating cpa security.

    c. Moreover, we can see it as, the adversary can distinguish between the messages m1 and m2 by simply looking at their cipher texts (provided the random seed r, and the key for random permutation F remains the same)

2. The scheme is not secure, it can be shown as:
    a. An adversary can choose two pairs of messages (m1, m2) and (m1', m2'), where m1 ≠ m1' and m2 = m2'. If the adversary receives the encryption of (m1, m2), they can tell it apart from the encryption of (m1', m2') by looking at the first part of the ciphertext, which should not be possible in a CPA-secure scheme.

Rubric

Mentioned in the Paper : **5 marks** each for **Scheme A** and **Scheme B**

Question 5

5. Prove that the basic CBCMAC is not secure when we consider the messages of different lengths (you need to show an attack). Can you modify the basic CBCMAC for variable length messages to make it secure? Explain in detail. Does the security of CBCMAC depend on the IV? $5 + 3 + 2 = 10$

Solution

1. CBC-MACs are not secure, for variable length messages since one can easily forge an attack like shown below,
    a. Conside messages m1 and m2 with their corresponding cbc mac tags t1 and t2. One can simply show that if we can generate a tag for the message m1 || m2, by XOR-ing the tag t1 for message m1 with the first block of the message m2. And hence, this can't be secure.
2. CBC-MACs can be made secure against variable length messages by simply
    a. Appending the length of the message m to the message, before passing it through the CBC-MAC, this provides it security against the attack discussed below, since the tag generated for m1 || m2 would be different due to the additional message length block introduced.
    b. We could simple use 2 keys, to generate the final tag for the cbc mac, say we have 2 keys k1 and k2. We could generate the usual tag from cbc mac, using the key k1, and then pass the tag through a pseudorandom function with key k2, and output the final result. Thus the new tag thus generated being t' = Fk2 (t).
3. Yes the security of CBC MAC depends on the IV, conside the case, we have a message m comprising of message blocks m1,m2, m3 ….. mn, and a random IV - **IV1,** consider the output for the given message m and **IV1** to be tag **t.** If, we simply swap the first block of the message m i.e. **m1** and **IV1**, we get the same tag **t**, for a different message and thus affects the security of the CBC-MAC. Thus to ensure security of the CBC-MAC, IV is set to 0.

Rubric

Mentioned in the Paper : **5 marks** each for **Proving that CBC Mac is not secure for different message lengths, 3 marks** for **modification, 2 marks** for showing that the **security of CBCMAC does depend on the IV**

Question 6

6. Show that the CBC and Counter modes of operation for block-ciphers are not CCA-secure.

Solution

“A public-key encryption scheme is considered CCA secure if an attacker who has access to a decryption oracle cannot gain any useful information about the plaintext or the secret key beyond what is already possible without access to the oracle. CCA security ensures that the encryption scheme remains secure even if an adversary can interact with a decryption oracle”
In CBC mode, each plaintext block is XORed with the previous ciphertext block before encryption. This introduces dependency between blocks, which makes parallelization difficult. An attacker can manipulate the ciphertext blocks to modify the corresponding plaintext blocks during the decryption process, resulting in predictable modifications to the plaintext output.
In CTR mode, each block of plaintext is XORed with the output of the encryption function applied to a unique counter value. While CTR mode is parallelizable and efficient, it is not secure against chosen ciphertext attacks. An attacker can modify the ciphertext and observe the corresponding changes in the plaintext when decrypted. By carefully choosing and modifying ciphertext blocks, an attacker can gain information about the plaintext or even decrypt it fully.
In CTR mode, the attacker can modify the ciphertext blocks to directly affect the resulting plaintext blocks.

Rubric

<Insert Rubric Here>

Question 7

7. Given a family of fixed length collision resistant hash functions $h^s : \{0,1\}^{(n+1)} \to \{0,1\}^n$, show how to build a family of collision resistant hash function H^s for arbitrary length messages $H^s : \{0,1\}^* \to \{0,1\}^n$, using the Merkle-Damgard transform and prove its security. How would you go about building H^s if you are given two functions $h_1^s : \{0,1\}^{(n+1)} \to \{0,1\}^n$ and $h_2^s : \{0,1\}^n \to \{0,1\}^{(n-1)}$ where one of them is collision-resistant family of hash functions, and you do know which one? $7 + 3 = 10$

Solution

Given a family of fixed length collision resistant hash function h^* : $\{0,1\}^{(n+1)}$ -> $\{0,1\}^{(n)}$, show how to build a family of collision resistant hash function H^* for arbitrary length message H^* : $\{0,1\}^*$ -> $\{0,1\}^n$, using the Merkel-Damgard transform and prove its security.
How would you go about building H^* if you are given two functions h^*_1 : $\{0,1\}^{(n+1)}$ -> $\{0,1\}^{(n)}$ and h^*_2 : $\{0,1\}^{(n)}$ -> $\{0,1\}^{(n-1)}$ where one of them is a collision-resistant family of hash functions, and you do know which one? [7+3]

Part1

The Merkle-Damgard transform can be used to construct a collision-resistant hash function H^*:$\{0,1\}^* \to \{0,1\}^n$ from a family of fixed-length collision-resistant hash functions $h*$:$\{0,1\}^{n+1} \to \{0,1\}^n$. The goal is to construct $H*$ that is collision-resistant for arbitrary lengths.
The Merkle-Damgard transform works by breaking the input into fixed-size blocks and iteratively processing each block. The final hash output is derived from the hash of the last block and the processing of each block. The construction is as follows:

- Padding: Given an input message M of arbitrary length, pad M to ensure that its length is a multiple of n bits. This usually includes appending a single '1' bit, followed by a sequence of '0' bits, and then the length of the original message M encoded in a fixed number of bits at the end.
- Set the initial hash value h_0 to a predefined initial value (IV), typically a string of n zero bits.
- Divide the padded message M into m1,m2,…,mk blocks of n bits each.
- Iterative Compression: Define a compression function g using $h*$ as follows:
  For each block mi , let m'_i be the concatenation of mi and a single bit '0' or '1' (chosen based on the design) to make it n+1 bits long. Then compute:
  $h_i = h^* (h_{i-1} \,||\, m'_i)$, where h_0 = IV

- Each block is processed sequentially. For each block m_i, the hash function $\hbar$ is applied to the combination of m_i and the hash output of the previous block
- The final hash is $H^*(M)=h_k$.

Proof of Collision Resistance:

Suppose there exists a collision in H^* , i.e., there are two distinct messages M_1 != M_2 such that H*(M1) = H*(M2)
Due to the Merkle-Damgård construction, this would imply a collision in at least one of the intermediate hash outputs, i.e., $\hbar_i=\hbar_i'$ for some i. However, since $\hbar*$ is collision-resistant by hypothesis, finding such a collision in $\hbar_i$ would contradict the assumption that $\hbar*$ is collision-resistant. Hence, if $\hbar*$ is collision-resistant, then H* is also collision-resistant.

Part2
Given that:
$h^*_1 : \{0,1\}^{(n+1)}$ -> $\{0,1\}^{(n)}$ and $h^*_2 : \{0,1\}^{(n)}$ -> $\{0,1\}^{(n-1)}$

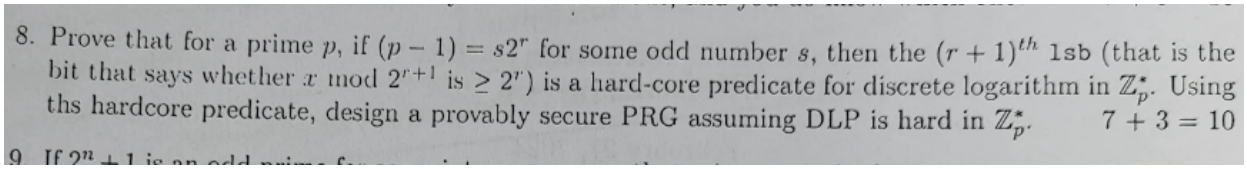
And we know which one of h^*_1 and h*2 is collision resistant. So let's assume h*1 is the collision resistant function and we use it as the compression function, the process will be same for h*2 as well.

1. Padding : Pad the input message M to make sure the length of the msg is a multiple of n.
2. Set the initial hash value $\hbar_0$ to a predefined initial value (IV), typically a string of n zero bits.
3. Divide the padded message M into m1,m2,…,mk blocks of n bits each. And for each block m_i, concatenate an additional bit to make it n+1 bits.
4. For each block we have to compute: $h_i = h_1^* (h_{i-1} || m_i)$
5. The final hash is $H^*(M)=h_k$.

Rubric

Mentioned in the Paper : 7 marks for **First Part, 3 marks** for the **second Part**

Question 8



8. Prove that for a prime p, if $(p-1) = s2^r$ for some odd number s, then the $(r+1)^{th}$ lsb (that is the bit that says whether $x \bmod 2^{r+1}$ is $\geq 2^r$) is a hard-core predicate for discrete logarithm in $\mathbb{Z}_p^*$. Using ths hardcore predicate, design a provably secure PRG assuming DLP is hard in $\mathbb{Z}_p^*$. $7 + 3 = 10$

9. If $2^n + 1$ is an odd prime for

Solution

Given a prime p such that $p-1 = s2^r$ for some odd number s, let's consider the function $f : Z_p^* \rightarrow \{0,1\}$ defined by:
$f(x) = (x \bmod 2^{(r+1)})$

We are concerned with the (r+1)th bit of this function's output, which we will denote as
b(x). That is:
b(x)= the (r+1)th lsb of f(x)
Assume, for contradiction, that there exists a polynomial-time algorithm A that, given $y = g^x \bmod p$ for some generator g of Z^*_p, predicts b(x) with non-negligible advantage over random guessing.

We could use A to construct a polynomial-time algorithm B that solves the DLP in Z^*_p as follows:

1. Algo B : Given $y = g^x \bmod p$, do the following:
    - For i = 0 to r, repeat:
        - Use A to predict $b(x \bmod 2^{(i+1)})$.
        - If A predicts 1, then we know x has the (i+1)th bit set, and we refine our guess for x accordingly.
    - After r+1 iterations, we have enough information to determine $x \bmod 2^{(r+1)}$ completely, since $2^{(r+1)}$ divides p-1, we can recover x from its residues by the chinese Remainder Theorem or successive squaring.

Since B runs in polynomial time and uses A to solve DLP, this contradicts the assumed hardness of DLP. Hence, no such A can exist, and therefore b(x) is a hard-core predicate.

PRG Construction:

Given that DLP is hard in Z^*_p , we can construct a provably secure pseudorandom generator G using the hard-core predicate b(x).

1. PRG G : Given a random seed x for r+1 bits, do the following :

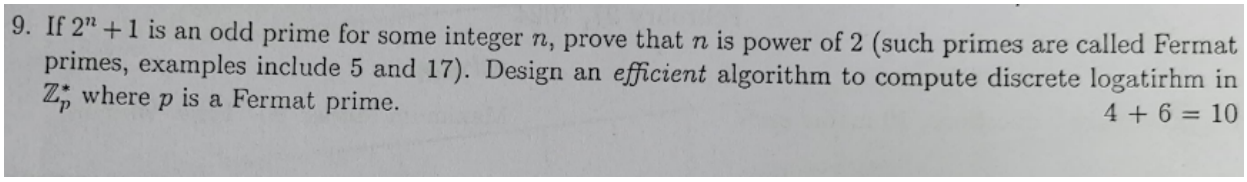
    - Choose a random generator g of Z^*_p .
    - COmpute $y = g^x$ mode p.
    - Output b(x), which is the (r+1)th lsb of x mod $2^{(r+1)}$
    - Update x to x+1 and repeat for more bits.

Since b(x) is a hard-core predicate, the output bits are computationally indistinguishable from a random sequence. This PRG is secure provided the DLP is hard.
To actually implement this PRG in practice, one would need to use specific cryptographic algorithms and parameters that are proven to be secure and efficient. The PRG's security rests on the assumption that the DLP remains hard in the chosen group Z^*_p.

Rubric

Mentioned in the Paper : 7 marks for **First Part, 3 marks** for the **second Part**

Question 9


9. If $2^n + 1$ is an odd prime for some integer n, prove that n is power of 2 (such primes are called Fermat primes, examples include 5 and 17). Design an *efficient* algorithm to compute discrete logatirhm in $\mathbb{Z}_p^*$ where p is a Fermat prime. $4 + 6 = 10$

Solution

Assume that n is not of the form 2^k. Then there exists an odd factor d of n. We know that for odd numbers d , we can write $x^d + 1$ as

$$x^d + 1 = (x + 1) \cdot (x^{d-1} - x^{d+2} \cdots - x + 1)$$

Therefore, if $d|n$, where d is an odd number then we can write $n = d \cdot q$, Hence we have

$$2^n + 1 = 2^{dq} + 1 = (2^d)^q + 1$$

Which you can then factorize using the above factorization. Hence $2^d + 1$ is a factor of $2^n + 1$. Therefore it cannot be a prime.

The DLP Problem requires us to find, given a number h, a number x such that $g^x \equiv h \bmod p$. Where g is a generator of the group Z_p^* . We know that $p = 2^k + 1$, and we also know that the numbers are between 1 and $p - 1 = 2^k$. Therefore x can be represented as a $k + 1$ bit number . Let the number be

$$x = c_k c_{k-1} \ldots c_1 c_0 = c_0 2^0 + c_1 2^1 + c_2 2^2 + c_3 2^3 ... + c_k 2^k$$

Algorithm

1. Calculate $val = h^{2^{k-1}} \bmod p$.
2. if $val = 1$ then $c_0 = 0$. Else if $val = p - 1$ then $c_0 = 1$
3. Now, remove the last bit from x, by multiplying it with the multiplicative inverse of $g^{c_0 2^0}$
4. Now continue the same steps with c_1

Why does this work?

The reason why this works is as follows. If the last bit was indeed 0. Then we know that x is even, this implies that x is of the form $2y$. Hence we have

$$\begin{aligned} h^{2^{k-1}} \bmod p &\equiv (g^x)^{2^{k-1}} \bmod p \\ &\equiv (g^{2y})^{2^{k-1}} \\ &\equiv g^{y \cdot 2^k} \\ &\equiv (g^{2^k})^y \end{aligned}$$

Now from Euler's totient theorem, we know that

$$a^{\phi(p)} = 1 \bmod p$$

Also since the number in question is a fermat prime, therefore the totient function is just $\phi(p) = p - 1$ (Fermat's little theorem).

Hence

$$g^{2^k} \equiv 1 \bmod p$$

Hence we get the final output to be 1 . On the other hand, if c_0 was instead 1, then we would have had

$$\begin{aligned} h^{2^{k-1}} \bmod p &\equiv (g^x)^{2^{k-1}} \bmod p \\ &\equiv (g^{2y+1})^{2^{k-1}} \\ &\equiv g^{y \cdot 2^k + 2^{k-1}} \\ &\equiv (g^{2^k})^y \cdot g^{2^{k-1}} \end{aligned}$$

Now we know that the first term on the right hand side, is obviously still 1, but what about the second term $g^{2^{k-1}}$? We notice that this is essentially a modular square root of $g^{2^k} \equiv 1 \bmod p$. Now, we know that if p is prime, then there are no non trivial square roots of 1 mod p . Hence, the only possible values of $g^{2^{k-1}} \bmod p$ are 1 and $p-1$. However, 1 is not a possible value either, since we know that the generator must be able to generate all the values of the group, if the value was indeed 1 then it would only be able to generate 2^{k-1} elements (will start cycling again over those elements when you perform higher powers over it). Therefore the only possible value is $p - 1$.Hence depending on the value of *val*, we get an idea of the value of c_0. We then keep performing the same set of steps to calculate the value of $c_1, c_2 \ldots c_k$.

Rubric

Mentioned in the Paper : 4 marks for **First Part, 6 marks** for the **second Part**