Assignment 01 - Part 1,(Vansh Agarwal,231120

Solution

Ques1 a:

A cryptosystem is said to be **perfectly secret** if:

$$\Pr(P = p \mid C = c) = \Pr(P = p)$$

for all plaintexts p and ciphertexts c.

Using Bayes' Theorem:

$$\Pr(P = p \mid C = c) = \frac{\Pr(C = c \mid P = p) \cdot \Pr(P = p)}{\Pr(C = c)}$$

Since encryption is defined as $C = P \oplus K$, and the key K is uniformly random over $\{0,1\}^n$:

$$\Pr(C = c \mid P = p) = \Pr(Enc(k, p) = c) = \frac{1}{(k|Enc(k, p) = c)} = \frac{1}{2^n}$$

No of cipher texts corresponding to one plain text are equal to no of keys available which are 2^n. Now, Assuming that plaintexts are also uniformly distributed:

$$\Pr(P = p) = \frac{1}{2^n}$$

Now compute:

$$\Pr(C = c) = \sum_{p \in \{0,1\}^n} \Pr(C = c \mid P = p) \cdot \Pr(P = p) = \sum_{p \in \{0,1\}^n} \frac{1}{2^n} \cdot \frac{1}{2^n} = 2^n \cdot \frac{1}{2^n} \cdot \frac{1}{2^n} = \frac{1}{2^n}$$

Putting it back into Bayes' theorem:

$$\Pr(P = p \mid C = c) = \frac{\frac{1}{2^n} \cdot \frac{1}{2^n}}{\frac{1}{2^n}} = \frac{1}{2^n} = \Pr(P = p)$$

Conclusion

Since the probability distribution of the plaintext does not change after observing the ciphertext, the OTP scheme satisfies perfect secrecy.

Ques1 b:

```
bit_str1 = "00011000000101110000010100000001000010 10"
bit_str2 = "00000111000111010001101100000001000000 01"
int_bit_str1 = int(bit_str1, 2)
int_bit_str2 = int(bit_str2, 2)
result = int_bit_str1 ^ int_bit_str2

str1 = "value"
str2 = "email"
xor_result = [ord(a) ^ ord(b) for a, b in zip(str1, str2)]
print(xor_result)

str3 = "hello"
str4 = "world"
xor1_result = [ord(a) ^ ord(b) for a, b in zip(str3, str4)]
print(xor1_result)
print(result)

num = 133313724427
length = (num.bit_length() + 7) // 8
byte_rep = num.to_bytes(length, byteorder='big')

# This is a bytes object (not human-readable)
print(byte_rep)

byte_data = b'\x1f\n\x1e\x00\x0b'

# Convert each byte to an integer
int_values = [b for b in byte_data]
print(int_values)
# Result I got
[19, 12, 13, 28, 9]
[31, 10, 30, 0, 11]
133313724427
b'\x1f\n\x1e\x00\x0b'
[31, 10, 30, 0, 11]
```

In this question, we are given two strings, c_1 and c_2, which are the results of XORing two plaintexts with the same key. If we compute $c_1 \oplus c_2$, where $c_1 = m_1 \oplus k$ and $c_2 = m_2 \oplus k$, then:

$$c_1 \oplus c_2 = (m_1 \oplus k) \oplus (m_2 \oplus k) = m_1 \oplus m_2$$

This operation nullifies the effect of the key. We can then try different word combinations to find two plaintexts that produce the same XOR result. In this case, we found that $m_1 =$ `"hello"` and $m_2 =$ `"world"`.

Ques1 c: part A

Here we have $c = pk \bmod n$. Since $k \in Z_n^+$ and n is a prime number, we have $\gcd(n, k) = 1$, so the modular inverse $k^{-1} \bmod n$ exists. To decrypt, we multiply the ciphertext with k^{-1} and take modulo n, i.e., the decryption scheme is $p = c \cdot k^{-1} \bmod n$.

Ques1 c: part B

This scheme is not perfectly secure. To explain, let's consider an example: if $c = 1$, then we know that p and k are modular inverses of each other. Similarly, if $c = 0$, we can be certain that $p = 0$. This reveals information about the plaintext, which violates the notion of perfect secrecy.

Ques1 d:

We analyze a hex-encoded ciphertext using Python. The following operations are performed:

1. Convert the hex string to bytes.
2. Count total and printable ASCII bytes.
3. Extract the first and last 28 bytes of the message.
4. XOR them byte by byte.

Python code:

```
# Hex-encoded ciphertext (possibly OTP-encrypted or obfuscated message)
hex_str = (
    "221C05471C0E00551B09151D4F171C550B4F164F1301011C1D000E04"
    "266E20646F20666F7220796F752C2061736B207768617420796F"
    "752063616E20646F20666F7220796F757220636F756E7472792E204A464B"
)

# Calculate total number of bytes in the hex string
length_bytes = len(hex_str) // 2
print("Total length (bytes):", length_bytes)

# Convert hex string to a byte array
byte_array = bytes.fromhex(hex_str)
print("Byte array:", byte_array)

# Get integer values of each byte
int_values = [b for b in byte_array]
print("Length of byte array:", len(byte_array))
print("Integer values:", int_values)
```

```
# Count number of printable ASCII characters (32 <= b <= 126)
printable_ascii_count = sum(1 for b in byte_array if 32 <= b <= 126)
print("Number of printable ASCII bytes:", printable_ascii_count)
print("Total number of bytes:", len(byte_array))

# Extract the first and last 28 bytes
start = byte_array[:28]
end = byte_array[-28:]

# XOR the two 28-byte segments
xor_result = bytes([a ^ b for a, b in zip(start, end)])

print("XOR result (bytes):", xor_result)
print("XOR result (as hex):", xor_result.hex())

# Possibly relates to JFK quote:
# "Ask not what your country can do for you, ask what you can do for your country.
```

I decoded the hexadecimal string and the result was:

```
"\x1c\x05G\x1c\x0e\x00U\x1b\t\x15\x1dO\x17\x1cU\x0bO\x16O\x13\x01\x01\
   x1c\x1d\x00\x0e\x04"&n do for you, ask what you can do for your
   country. JFK'
```

After examining the output, I noticed that the string ends with a readable English sentence. I searched for the full quote ending with that line and confirmed that the original message is:

"Ask not what your country can do for you, ask what you can do for your country. JFK"

Ques2: part A

To derive the decryption scheme, we are given the encryption scheme:

$$c = a(p + b) \bmod 28$$

Assuming that a is relatively prime to 28, i.e., $\gcd(a, 28) = 1$, we know that the modular inverse of $a \mod 28$ exists. Using an algorithm like the extended Euclidean algorithm, we can find a^{-1}, the modular inverse of a.

To decrypt the ciphertext c, we proceed as follows:

$$a^{-1} \cdot c \equiv p + b \pmod{28}$$

Now, subtract b from both sides:

$$a^{-1} \cdot c - b \equiv p \pmod{28}$$

Thus, the decryption scheme is:

$$p = (a^{-1} \cdot c - b) \bmod 28$$

This allows us to recover the original plaintext p from the ciphertext c.

Ques2: part B

There is only one trivial key for which $c = p$ for all inputs, and that key is $a = 1$ and $b = 0$. For non-trivial keys:

- The possible values for a are the integers in the set

  $$\{1, 3, 5, 9, 11, 13, 15, 17, 19, 23, 25, 27\}$$

  which gives 12 possible values (i.e., the elements of Z_{28}^*).

- The value b can be any integer from 0 to 27, giving 28 possible values.

Thus, the total number of keys is:

$$12 \times 28 = 336$$

Subtracting the one trivial key ($a = 1, b = 0$), we are left with:

$$336 - 1 = 335$$

non-trivial keys.

Ques2: part C

To retrieve the key using a chosen plaintext attack (CPA), we can assume that Alice is the one who provides us with the ciphertext for given plaintexts. Then, we perform the following series of operations:

1. Query Alice with plaintext $p = 0$. She will respond with $c_1 = ab \bmod 28$.
2. Query Alice with plaintext $p = 1$. She will respond with $c_2 = a(1 + b) \bmod 28 = a + ab \bmod 28$.

Now that we know c_1 and c_2, we can subtract them:

$$c_2 - c_1 \equiv a \pmod{28}$$

This gives us the value of a. Since $\gcd(a, 28) = 1$, we can compute the modular inverse $a^{-1} \bmod 28$.

Now, multiply c_1 with a^{-1} and take the result modulo 28:

$$a^{-1} \cdot c_1 \equiv b \pmod{28}$$

In this way, both a and b can be recovered, and the key is retrieved using a chosen plaintext attack.

Ques3:

Statistical Analysis and Plaintext Recovery

Using frequency analysis and educated guesses on common words and patterns, we recovered mappings and partial plaintext as follows:

1. The trigram vsl repeats very frequently, so we guessed vsl = "the".
2. Observing the context around a year, we recognized vslmqlfxm1866mrfy must correspond to "the year 1866 was", yielding:

   $$\texttt{vslmqlfxm1866mrfy} \longrightarrow \textit{the year 1866 was}$$

   and so m = space.
3. From mfchmvslm we recovered " and the ".
4. The bigram _c suggested "in", but failed for z cvoclcvyumyl. Replacing i with o gave the correct "object" later.
5. vlchlczq mapped to "tendency", and following rsosz gave "which".
6. In z cvoclcvyumyl, taking o = space helped recover " of you".
7. um appears 24 times, so we set um = ". " (period plus space).
8. yvfvly = "on", and m cm = " in".
9. rsosz = "which".
10. From ltf \,, lxfvlh we found "exaggerated" by mapping x → t.
11. l.pfnnq indicated "equally", giving mappings . → q, p → u.
12. .plyvo c hinted at "?estion", confirming "question" and mapping a → ,.
13. mvsoym kblzvm vsoy translated to ", then object".
14. _i = "of".
15. xljfxgfknl with pattern "re_ar_able" led to "remarkable", so j → m, g → k.
16. epwwnoc, from "_u__ling" gave "puzzling".
17. yldlxfn implied "several", so d → v.

Each step used frequency counts, known plaintext fragments, and trial substitutions to incrementally reveal the full message.

itc

Plain	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	.	”	
Cipher	f	k	z	h	l	i	,	s	o	m	g	n	j	c	′	′	e	.	x	y	v	p	d	r	t	q	w	m	a	u

Here, the plain-alphabet symbols on the top row map to the corresponding cipher-alphabet symbols on the bottom row.

Plaintext that I found

the year 1866 was signalised by a remarkable incident. a mysterious and puzzling phenomenon. which doubtless no one has yet forgotten not to mention rumours which agitated the maritime population and excited the public mind. even in the interior of continents. seafaring men were particularly excited merchants. common sailors. captains of vessels. skippers. both of europe and america. naval officers of all countries. and the governments of several states on the two continents. were deeply interested in the matter for some time past. vessels had been met by ”an enormous thing.” a long object. spindle-shaped. occasionally phosphorescent. and infinitely larger and more rapid in its movements than a whale the facts relating to this apparition (entered in various log-books) agreed in most respects as to the shape of the object or creature in question. the untiring rapidity of its movements. its surprising power of locomotion. and the peculiar life with which it seemed endowed if it was a cetacean. it surpassed in size all those hitherto classified in science taking into consideration the mean of observations made at divers times.—rejecting the timid estimate of those who assigned to this object a length of two hundred feet. equally with the exaggerated opinions which set it down as a mile in width and three in length.—we might fairly conclude that this mysterious being surpassed greatly all dimensions admitted by the ichthyologists of the day. if it existed at all and that it did exist was an undeniable fact; and. with that tendency which disposes the human mind in favour of the marvellous. we can understand the excitement produced in the entire world by this supernatural apparition as to classing it in the list of fables. the idea was out of the question

But I think there is bit of error in key.

Ques4: part A

In this question I have used chatgpt just for better representation nothing else.

The scenario assumes that there are at most n consoles in the world (e.g., $n = 2^{32}$), and they are represented as the leaves of a complete binary tree of height $\log_2 n$. Each node v_j in the tree holds a key $k_j \in \mathcal{K}$, where $\mathcal{K}$ is the key space. These keys are fixed at manufacturing time and remain secret from end users.

Each console $i \in \{0, 1, \ldots, n-1\}$ is assigned a unique serial number and embedded with the $1 + \log_2 n$ keys along its path from the root to leaf i, denoted by the set S_i. Hence, console i stores keys $\{k_j : v_j \in S_i\}$.

The standard encryption of a game message m is:

$$\text{Console} := E(k_{\text{root}}, k) \,\|\, E(k, m)$$

where:

- $E(\cdot, \cdot)$ is a symmetric encryption scheme,
- $k \leftarrow \mathcal{K}$ is a freshly generated content key for encrypting message m,
- The first component $E(k_{\text{root}}, k)$ is referred to as the *header*,
- The second component $E(k, m)$ is referred to as the *body*.

Revocation Scenario:

Suppose the keys embedded in console i have been exposed due to a hack. To prevent console i from accessing newly released games, the game company can construct a new header that enables all other consoles to decrypt the game, except console i.

The idea is to encrypt the content key k under a set of $\log_2 n$ keys, each corresponding to a node in the tree such that the union of these $\log_2 n$ nodes covers all other consoles except console i. This can be done using a minimal set of sibling nodes that together cover the entire tree except the path S_i.

Thus, the new header becomes:

$$\text{Header} := E(k_{j_1}, k) \,\|\, E(k_{j_2}, k) \,\|\, \cdots \,\|\, E(k_{j_{\log_2 n}}, k)$$

where $\{k_{j_1}, \ldots, k_{j_{\log_2 n}}\}$ are keys associated with nodes disjoint from S_i.

Justification:

Each node in the binary tree corresponds to a subtree. By selecting the sibling nodes of the path S_i (i.e., the sibling of each node in S_i on the way from leaf i to the root), the company ensures that:

- Every non-compromised console shares at least one key from this set.
- Console i, having only keys along its own path S_i, does not have any of the keys used in the new header.

Thus, the body of the game encrypted as $E(k, m)$ remains secure from console i, effectively revoking its access.

Ques4: part B

Now suppose that hackers have exposed the embedded keys in a set of s consoles

$$I = \{\, i_0,\, i_1,\, \ldots,\, i_{s-1} \,\},$$

with $s > 1$. We wish to encrypt a new game m so that *every* console *not* in I can still decrypt, but no console in I can.

Tree-cover construction. Recall that each console i corresponds to a leaf in a complete binary tree of height $\log_2 n$, and stores the keys along its root-to-leaf path S_i. For each revoked leaf $i_t \in I$, let

$$S_{i_t} = \{ v_{t,0}, v_{t,1}, \ldots, v_{t,\log_2 n} \}$$

be its path nodes, where $v_{t,0}$ is the root and $v_{t,\log_2 n}$ is the leaf. At each level j on the path of i_t, consider the *sibling* node $v'_{t,j}$ (i.e. the other child of $v_{t,j-1}$). Collect all such siblings over all t and all levels:

$$\mathcal{C} \;=\; \bigcup_{t=0}^{s-1} \{ v'_{t,1},\, v'_{t,2},\, \ldots,\, v'_{t,\log_2 n} \}.$$

Since each of the s paths has $\log_2 n$ siblings, $\big|\mathcal{C}\big| \le s \, \log_2 n$.

Header formation. Let $k \leftarrow \mathcal{K}$ be the fresh content key. We build the *header* by encrypting k under the key at each node in $\mathcal{C}$:

$$\text{Header} \;=\; \big\|_{v_j \in \mathcal{C}} E\big(k_j,\; k\big).$$

The full ciphertext is then

$$\text{Cipher} \;=\; \text{Header} \,\big\|\, E(k, m).$$

Correctness and cost.

- *Non-revoked consoles.* Any console $i \notin I$ has a root-to-leaf path S_i. At each level j, exactly one of the two children of the parent node lies on S_i. Since the header contains the sibling at that level for *every* revoked path, it must contain at least one key along i's own path. Hence i can decrypt some $E(k_j, k)$ in the header, recover k, and then decrypt $E(k, m)$.
- *Revoked consoles.* A revoked console $i_t \in I$ knows only the keys in its own path S_{i_t}, none of which appear in $\mathcal{C}$. Thus it cannot recover k.
- *Header size.* We use at most $\big|\mathcal{C}\big| \le s \, \log_2 n$ short ciphertexts, i.e. $O(s \log n)$.

Thus, by encrypting under the $O(s \log n)$ sibling-nodes of the s revoked paths, we revoke exactly the set I while preserving access for all other consoles.