Cryptography Engineering Midterm

110550108 施柏江

Problem 1

(a)

f(x) + g(x) = (x^2 + 1) + (x^3 + x^2 + 1) = <u>x^3</u>

f(x) - g(x) = (x^2 + 1) - (x^3 + x^2 + 1) = <u>x^3</u>

f(x) * g(x) mod P(x) = (x^2 + 1) * (x^3 + x^2 + 1) mod P(x)

= x^5 + x^4 + x^2 + x^3 + x^2 + 1 mod P(x)

= (x^5 + x^4 + x^3 + 1) mod (x^4 + x + 1)

= <u>x^3 + x^2</u>

(b)

f(x) + g(x) = (x^2 + 1) + (x + 1) = <u>x^2 + x</u>

f(x) - g(x) = (x^2 + 1) - (x + 1) = <u>x^2 + x</u>

f(x) * g(x) mod P(x) = (x^2 + 1) * (x + 1) mod (x^4 + x + 1)

= <u>x^3 + x^2 + x + 1</u>

Problem 2

(a)

f(x) + g(x) = (x^7 + x^5 + x^4 + x + 1) + (x^3 + x + 1) = <u>x^7 + x^5 + x^4 + x^3</u>

f(x) - g(x) = (x^7 + x^5 + x^4 + x + 1) - (x^3 + x + 1) = <u>x^7 + x^5 + x^4 + x^3</u>

f(x) * g(x) mod m(x) = (x^7 + x^5 + x^4 + x + 1) * (x^3 + x + 1) mod m(x)

= (x^10 + x^8 + x^7 + x^8 + x^6 + x^5 + x^7 + x^5 + x^4 + x^4 + x^2 + x + x^3 + x + 1) mod m(x)
= (x^10 + x^6 + x^3 + x^2 + 1) mod (x^8 + x^4 + x^3 + x + 1)

= <u>x^5 + 1</u>

(b)

Because $x^4 + 1 = (x + 1) * (x^3 + x^2 + x + 1)$, $f(x)$ can be written as the product of two polynomials.

Therefore, $f(x)$ is reducible over $GF(2^8)$.

Problem 3

(a)

Let $f(x) * h(x) = 1$.

Try $h(x) = x^3 + x^2 + x + 1$.

Then $f(x) * h(x) \bmod P(x) = x * (x^3 + x^2 + x + 1) \bmod P(x)$

$= (x^4 + x^3 + x^2 + x) \bmod (x^4 + x + 1 = x^3 + x^2 + 1)$

$!= 1$

Try $h(x) = x^3 + 1$.

Then $f(x) * h(x) \bmod P(x) = x * (x^3 + 1) \bmod P(x)$

$= (x^4 + x) \bmod (x^4 + x + 1)$

$= 1$.

Therefore, the inverse of $f(x) = x$ is $\underline{x^3 + 1}$.

(b)

Let $g(x) * h(x) = 1$.

Try $h(x) = x^2 + 1$.

Then $g(x) * h(x) \bmod P(x) = (x^2 + x) * (x^2 + 1) \bmod P(x)$

$= (x^4 + x^2 + x^3 + x) \bmod (x^4 + x + 1 = x^3 + x^2 + 1)$

$!= 1$

Try $h(x) = x^2 + x + 1$.

Then $g(x) * h(x) \bmod P(x) = (x^2 + x) * (x^2 + x + 1) \bmod P(x)$

$= (x^4 + x^3 + x^2 + x^3 + x^2 + x) \bmod P(x)$

$= (x^4 + x) \bmod (x^4 + x + 1) = 1$.

Therefore, the inverse of $g(x) = x^2 + x$ is $\underline{x^2 + x + 1}$.

Problem 4

(a)

We have $(x^8 + x^4 + 1) / (x^3 + x^2) = (x^5 + x^4 + x^3 + x^2) \ldots 1$,

so $(x^3 + x^2)^{(-1)} = x^5 + x^4 + x^3 + x^2$.

$(x^3 + x^2 + x) * (x^3 + x^2)^{(-1)} \bmod (x^8 + x^4 + 1)$

$= (x^3 + x^2 + x) * (x^5 + x^4 + x^3 + x^2) \bmod (x^8 + x^4 + 1)$

$= (x^8 + x^7 + x^6 + x^5 + x^7 + x^6 + x^5 + x^4 + x^6 + x^5 + x^4 + x^3) \bmod (x^8 + x^4 + 1)$

$= (x^8 + x^6 + x^5 + x^3) \bmod (x^8 + x^4 + 1)$

$= \underline{x^6 + x^5 + x^4 + x^3 + 1}$

(b)

$(x^6 + x^3 + 1) * (x^4 + x^3 + 1) \bmod (x^8 + x^4 + 1)$

$= (x^{10} + x^9 + x^6 + x^7 + x^6 + x^3 + x^4 + x^3 + 1) \bmod (x^8 + x^4 + 1)$

$= (x^{10} + x^9 + x^7 + x^4 + 1) \bmod (x^8 + x^4 + 1)$

$= \underline{x^7 + x^6 + x^5 + x^4 + x^2 + x + 1}$

Problem 5

(a)

因為 Mix Columns 中的乘法是基於 predefined matrix，因此我們可以創建一個 lookup table，為每個可能的位元組提供乘法的結果。而且 Mix Columns 的操作基本上是 linear 的，每個輸出都是通過 xor 輸入位元組而得到的。所以透過使用 lookup table 和 XOR，就可以實現與矩陣乘法相同的結果。

(b)

Byte Substitution 將 state matrix 中的每個位元組替換為 S-box 中對應的位元組，這種替換可以使用 lookup table 實現。Shift Rows 將 state matrix 的每一行按不同的偏移量進行循環移位，由於移位模式是固定的，可以預先計算並將其儲存在 lookup table 中。MixColumns 如前所述，可以使用 lookup table 和 XOR 運算來簡化。要將這些步驟合併成單一的 lookup table，可以預先計算 Byte Substitution、ShiftRows 和 MixColumns 對所有可能的輸入狀態的影響並將其儲存在 lookup table。然後在加解密期間，整個 AES 過程可以透過在此表中查詢並使用少量的 XOR 來實現。

Problem 6

1.InvShiftRows 只是將 byte 進行移位，InvSubBytes 只是對每個 byte 進行替換，與其位置無關。因此這兩個步驟可以交換。

2.對於任何線性變換 A : x→y = A(x)，有 A(x⊕k) = A(x)⊕A(k)。由於 AddRoundKey 只是將常數 ExpandedKey[i]與其輸入做 XOR，而且 InvMixColumns 是一個線性操作，所以

{AddRoundKey(State,ExpandedKey[i]);

InvMixColumns(State);}

可以改成

{InvMixColumns(State);

AddRoundKey(State,EqExpandedKey[i]); }

其中 EqExpandedKey[i]是將 InvMixColumns 應用於 ExpandedKey[i]後獲得的。

在 straightforward decryption algorithm 中，原本逆操作的順序無法有效率的實作。透過改變 InvShiftRows 和 InvSubBytes 的順序，並將 MixColumns 透過 AddRoundKey 的 XOR 操作，可以獲得等效的 decryption algorithm，將逆操作排列在一個適合高效實作的順序中。

Problem 7

如果需要與已使用 3DES 的系統保持相容，或是在某些地區的法規標準可能規定使用特定的加密演算法，可能會選擇 3DES 而不是 AES。再者，3DES 的金鑰長度比較短，可以簡化金鑰管理。

而當更需要安全性時，AES 有更高的安全性，因為擁有較長的金鑰。AES 通常速度更快並且只需要更少的計算資源。

在 Meet-in-the-Middle Attack 中，對於 2DES，攻擊者需要進行約 2^56 次操作，這在現代的計算能力下是可破解的。然而對於 3DES，上升到需要 2^112 次，這在目前的技術下使用暴力解仍需要許多時間。儘管理論上有漏洞，但 3DES 對於大多數的情況下是足夠安全的。

Problem 8

公司需要先正式撤銷前 CTO 持有舊金鑰的權限，確保舊金鑰無法再用於解密資料，並通知相關部門和人員舊金鑰權限已被撤銷。接著使用可信的金鑰生成機制生成一個新的金鑰來取代舊金鑰，並將其分發給所有需要加密資料的授權方，可能需要使用金鑰管理系統加密新金鑰。然後更新應用程序使其受新金鑰保護，以確保新金鑰的無縫加解密操作，並測試有無因新金鑰引發的問題。可以定期更換金鑰以減低洩漏風險，進一步增強安全性。最後詳細記錄金鑰管理事件，促進未來金鑰管理流程。

Problem 9

(a)

RSA 加密的強度在於將 n 因數分解為其質因數的難度，沒有更新 n 的話，如果攻擊者破解了 n，那麼在接下來的幾個月裡，無論 e 如何變化，都能破解訊息，因為有些 RSA 的攻擊方法能夠利用多個已知的公鑰和相同的 n 來進行攻擊。例如，如果攻擊者獲得了多對 (n,e)，且這些有相同的 n 和不同的 e，攻擊者可能利用這些來推斷私鑰。並且由於 Alice 的 public key pattern 容易預測，攻擊者可以利用這種模式來預測她未來的 public key。

(b)

雖然 Alice 每個月都更新 p 和 q，金鑰生成過程中缺乏了隨機性，仍然是可預測的。攻擊者可以分析使用的質數 sequence，並預測未來可能的質數，從而預測未來的 n。再者，key space 的大小決定了 RSA 加密的強度，有限的 key space 使得攻擊者更容易使用暴力破解，如果攻擊者將 n 分解並推導出 private key，就可以根據這個月的 p 和 q 推測其他月份的 p 和 q，從而獲得每個月的 private key。

Problem 10

(a)

Inverse MixColumns 是透過將 state matrix 每列中的每個字節都與矩陣的對應元素相乘，然後將這些結果相加，其中這個矩陣是加密過程中使用的 MixColumns 矩陣的反矩陣。GF(2^8)與一般算術不同的是，加減法等同於 XOR，除法是取 Inverse，乘法要對 m(x)取 mod，以確保結果保持在定義域內。

(b)

在這些式子中，⊕表示 XOR，2^k * x 表示將 x 左移 k 位的二進制表示，相當於乘以 2^k。

x 乘以 9 等價於 x 乘以 8 再加上 x：

$9 * x = (2^3 * x) \oplus x$

x 乘以 11 等價於 x 乘以 8，加上 x 乘以 2，再加上 x：

$11 * x = (2^3 * x) \oplus (2 * x) \oplus x$

x 乘以 13 等價於 x 乘以 8，加上 x 乘以 4，再加上 x：

$13 * x = (2^3 * x) \oplus (2^2 * x) \oplus x$

x 乘以 14 等價於 x 乘以 8，加上 x 乘以 4，再加上 x 乘以 2：

$14 * x = (2^3 * x) \oplus (2^2 * x) \oplus (2 * x)$

(c)

因為 inverse MixColumns 對每個 byte 進行相同的矩陣乘法操作，這些操作可以預先計算並儲存在 LUTs 中，進一步簡化 AES 解密。LUTs 具有以下優點，包含可以提高計算速度，將所有可能的乘法結果提前計算並儲存在 LUTs 中，避免重複計算。可以將複雜的乘法運算轉換為簡單的查找動作。但 LUTs 也存在一些潛在的缺點，像是需要額外的記憶體空間。如果在運行時需要更新 LUTs 中的資料，可能會導致性能下降。可能有安全性的疑慮，攻擊者可以透過分析 LUTs 來獲取有關系統操作的資訊。

Problem 11

在 PBC 中，Ci 的計算方式為 $C2 = Ek(m2) \oplus m1$, $C1 = Ek(m1) \oplus m0$, $C0 = Ek(m1) \oplus IV$。由於 m1 = m2 = x，可以改寫為 $C2 = Ek(x) \oplus x$, $C1 = Ek(x) \oplus m0$, $C0 = Ek(x) \oplus IV$。因為 $C0 = Ek(x) \oplus IV$，攻擊者可以將 C0 與 IV 進行 XOR 得到 Ek(x)：$Ek(x) = C0 \oplus IV$。然後因為 $C1 = Ek(x) \oplus m0$，攻擊者可以將 C1 與 Ek(x)進行 XOR 得到 m0：$m0 = C1 \oplus Ek(x) = C1 \oplus (C0 \oplus IV) = C1 \oplus C0 \oplus IV$。因為 C1、C0 和 IV 皆為已知，可以計算出 m0。

Problem 12

假設 M1 = m0 || m1 || m2, M2 = m0' || m1'|| m2'。我們可以從 M1 知道 IV1 和 T1，從 M2 知道 IV2 和 T2。我們可以令 M3 = m0 || m1 || m2'，如此一來 M3 的最後一個區塊與 M1 不同，且 MAC 是 (IV1, T2)。

Extra Credit 1

Security goals:

1. 確保儲存在系統中的敏感資訊僅授權用戶可存取。
2. 防止未經授權的修改。
3. 確保用戶能夠及時存取。
4. 確保只有合法用戶可以存取其帳戶。

Roles and responsibilities:

1. 負責配置和管理系統。
2. 作者擁有對該項目的完全控制權。
3. 參與項目開發的用戶能被作者授予相應的權限。

Security policies:

1. 必須通過身份驗證才能登錄。
2. 確保用戶僅能存取其被授予權限的資源。
3. 敏感資料在傳輸過程中應進行加密。

Legal considerations:

1. 確保系統符合相關的法律。
2. 提供明確的隱私政策和服務條款。

Extra Credit 2

(a)

要解密給定的密文，我們可以利用 OTP 加密的特性。因為密文是透過對 message 和 pad 進行 XOR 得到的，所以要解密它我們只需要再次將密文和 pad 進行 XOR。因此，要解密第一個詞: $mi\_1 = ci\_1 \oplus pi$, 要解密第二個詞: $mi\_2 = ci\_2 \oplus pi$。由於兩個詞都是使用相同的 p 加密的，$mi\_1 \oplus mi\_2 = ci\_1 \oplus pi \oplus ci\_2 \oplus pi = ci\_1 \oplus ci\_2$。計算得到 e9⊕f4 = 1d; 3a⊕3a = 0; e9⊕fe = 17; c5⊕c7 = 2; fc⊕e1 = 1d; 73⊕68 = 1b; 55⊕4a = 1f; d5⊕df = a。我們可以把 mi_1, ci_1, ci_2 全部 XOR 起來，$mi\_1 \oplus ci\_1 \oplus ci\_2 = mi\_1 \oplus mi\_1 \oplus mi\_2 = mi\_2$。因此我們可以令 M1 為隨便一個 8 字母長的英文單字，將 M1 與 $(C1 \oplus C2)$做 XOR，若得到的結果也是一個 8 字母長的英文單字，我們就能得出 M1 和 M2 了。經多次嘗試後得到 M1, M2 為 networks 和 security。

(b)

我們可以先從 10 個密文中計算所有可能的 pi 和可能的英文字母跟標點符號(共 60 個)。因為僅取決於 mi、ci 和 ci−1，pi 與其周圍的 pad 是 independent 的。因此我們可以跑遍每個字元，計算在第一個密文中導致該字元的 pi，並確認它是否能產生其他 9 個密文中的字元。我們可以檢查 plaintext 中出現了多少有效的英文單字，並給他 length(word)^2 的分數，這樣較長的單字可以獲得較高的分數。最後輸出得分最高的 message 和 pad。

Extra Credit 3

(a)

首先考慮一對沒有 offset 的密文 ci 和 cj (i != j)。我們可以將在 ci 和 cj 的 n 位中最長連續配對序列視為擲 n 次硬幣中的最多次連續正面，其中擲一次硬幣代表在相同位置上的配對。由於每個密文都是從 OTP 加密的，0 和 1 的可能性一樣，因此在相同位置上配對的機率為 0.5，最長連續序列 R0.5(n) 有很大的機率下小於 log 1/0.5 (n) + log 1/0.5 (ln n) = log2 (n ln n)。接下來考慮一對位移 k 位的密文，有 2n – 1 種位移。我們可以像上面一樣找到在 ci 和 cj 的 n – k 位中的最長配對序列的 upper bound。由於 n – k <= n，所以 R0.5(n – k) <= R0.5(n) <= log2 (n ln n)，因此之前的 upper bound 仍然成立。

(b)

如果我們將結果解讀為 bit，每個英文字母佔 8 bits，因此共同子字串的長度會是先前資料的 8 倍，而 log n 則基本相同。我們可以觀察到隨著 n 變大，最長連續序列的長度與 log n 的比值正在增加，也就是說在英文文章中相同字串的長度將比隨機密文長得多，可能是因為英文文章並不是隨機的，句子之間存在一定的關聯性。

(c)

我們可以將所有字串串接在一起，同時添加一個 $ 來分隔它們。將連接的字串表示為 S，我們可以將原始問題轉化為在 S 中尋找最長的重複子字串。根據假設每對明文都共享一個很長的相同字元的連續序列，而任何 independent 填充字符的密文 pair 則不會。假設我們找到了最長的重複子字串 S，並且兩個相同的子字串是 a 和 b。

如果 a 和 b 沒有重疊，它們應該都包含 $ 或都不包含。如果它們包含 $，假設 a 被 $ 分成 a1、a2，b 被 $ 分成 b1、b2。那麼 a1、b1 必須相同，a2、b2 也必須相同，因此很有可能包含 a1、b1 的兩個密文共享相同的密鑰。如果它們都不包含 $，同樣地密文 a 和 b 應該共享相同的密鑰。如果 a 和 b 重疊，它們就不會包含 $，否則在 n 位中 $ 會出現兩次，因此 a、b 在同一個密文中。根據(a)小題我們可以得知這種情況發生的機率不高。因此我們將原始問題轉化為在 S 中尋找最長的重複子字串，只需要找到從根節點到最長路徑的內部節點，這可以透過 traverse the suffix tree 來完成。創建 S 的 suffix 需要 O(N log N)，traversal 需要 O(N)，因此整個演算法的時間複雜度為 O(N log N)。