Introduction to Cryptography: Practice Questions:

Sean al-Baroudi

April 21, 2022

Chapter 4 Practice Questions:

4.1.1:

- AES was completely open in design.
- NIST held a competition where non-government employees could submit proposals. Various groups, companies and researchers did.
- There were multiple revision rounds to narrow down to a final candidate - all participants could review designs and comment.
- A civilian algorithm was chosen in the end (Rinjdael).

4.1.3 and 4.1.4: Rinjdael, made by Vincent Rijmen and Joan Dameen.

4.1.5: Block size of 128 bits, and 128/192/256 keys are supported, for gradiations in encrypted security.

4.2 The addition and mutliplication tables for GF(7) are below. Note the banded diagonal structure for all modulo addition tables. For a multiplication table, each row and column (excluding zero margins) should have a "1" in it, indicating a complete set of inverses.

+	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1	2	3	4	5	6	0
2	2	3	4	5	6	0	1
3	3	4	5	6	0	1	2
4	4	5	6	0	1	2	3
5	5	6	0	1	2	3	4
6	6	0	1	2	3	4	5

×	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6
2	0	2	4	6	1	3	5
3	0	3	6	2	5	1	4
4	0	4	1	5	2	6	3
5	0	5	3	1	6	4	2
6	0	6	5	4	3	2	1

4.4.1 Consider GF(2^4), an extension field.

$$A(x) + B(x) mod P_I(x) = x^3 mod(x^4 + x + 1) = x^3 = (1000)_2$$

4.4.2

$$A(x) + B(x) mod P_I(x) = x^2 + x mod(x^4 + x + 1) = x^2 + x = (0110)_2$$

For both questions, the summands directly represent polynomial equivalence classes. No modulus division needs to be done.

4.4.3: The choice of irreducible polynomial affects the equivalence class that elements are mapped to.

4.5.1: Consider GF(2^4) again, and our polynomial representations for this extension field:

$$A(x)B(x) mod P_I(x) = (x^5 + x^4 + x^3 + 1) mod(x^4 + x + 1)$$

Performing polynomial long division, we get the following:

$$(x + 1)(x^4 + x + 1) + (x^3 + x^2) = x^5 + x^4 + x^3 + 1$$

So our equivalence class after modulus division is $x^3 + x^2$, represented in binary as $(1100)_2$

4.5.2:

$$A(x)B(x) mod P_I(x) = (x^3 + x^2 + x^1 + x^0) mod(x^4 + x + 1)$$

This maps direclty to an equivalence class $(1111)_2$, and does not have to be reduced.

4.9: For this, lets calcualte each state in stages: (A) Key Whitening, (B) Byte Substitution, (C) Shift Rows and (D) Mix Columns. We have AES(128,192) with the following:

$$x = [1_1 ... 1_{128}]_2 = [F_1 ... F_{32}]_{16}$$

4.9.A: For key whitening, k_0 is just the inputted key. We are XORing a message of 1's with a key of all 1's, so our result is all zeros.

$$x \oplus k_0 = [0_1 ... 0_{16}]_{256}$$

in terms of bytes.

4.9.B: For each of the 16 zero bytes, we use our Substitution table. X = 0000 and Y=0000, which maps to $[(6)(3)]_{16}$ as seen in the table. Each of the 16 bytes is mapped to the same value.

$$S([00]_{16}) = [63]_{16} = [(0110)(0011)]_2$$

Then:

$$\bar{B} = [61]_1 ... [63]_{16}$$

4.9.C: For the ShiftRows operation, our output will be the same as we move bytes around, and all our bytes are the same. So:

$$\bar{B}' = [61]_1 ... [63]_{16}$$

4.9.D: For the Mix Columns operation, all of our bytes are the same. We only need to perform three multiplications on the GF(2^8) Extension Field, for our summations later.

$$[01]_p[63]_p = (x^6 + x^5 + x + 1) =$$

$$[02]_p[63]_p = x(x^6 + x^5 + x + 1) = x^7 + x^6 + x^2 + x$$

$$[03]_p[63]_p = (x + 1)(x^6 + x^5 + x + 1) = x^7 + x^5 + x^2 + 1$$

For all calculations, degree < 8 so no modular reduction has to be done wiht P_I. At first, our MixColumn Matrix Transform looks very complicated - do we have to solve 16 rows of equations to get our output bytes?

$$\begin{pmatrix} C_i \\ C_{i+1} \\ C_{i+2} \\ C_{i+3} \end{pmatrix} = \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix} \begin{pmatrix} B_i \\ B_{i+1} \\ B_{i+2} \\ B_{i+3} \end{pmatrix}$$

Note that this matrix equation is conceptual (the B's are not indexed properly). We can see that each matrix row contains the numbers 1 1 2 3. As our B's are all the same value, this means that each of our 16 row equations are just the same permuted sum. Solving for one, we solve for all:

$$[01][63] = ... \text{ Cancels with lower equation.}$$

$$[01][63] = ... \text{ Cancels with upper equation.}$$

$$[02][63] = x^7 + x^6 + x^2 + x$$

$$[03][63] = x^7 + x^5 + x^2 + 1$$

Summing the terms in GF(2),

$$= (x^6 + x^5 + x) = [62]_{16}$$

So

$$\bar{C} = [62]_1 ... [62]_{16}$$

4.16.1:

- 100 000 ASICs can test 3×10^12 keys/second.
- We have $2^{192} \leq 6.3 \times 10^57$ possible keys to test (at most).
- Dividing the number of keys by number of keys we can test, we get $2.1x10^{45}$ seconds.
- This is $6.67x10^{37}$ years. This is many, many times the current age of the universe $(\tilde{1}0^10yrs.$

4.16.2: Moore's Law: We double computing power every 1.5 years.

- We will scale each ASIC accordingly, to get the required answer. We must have all of our ASICS test $6.3x10^{57}$ keys within a 24 hour period.
- Divide our number of keys by the number of seconds in a 24 hour window, and the number of ASICS we have. Result: 7.29×10^47.
- Let k be the number of doublings we must perform, to scale up our hardware to reach the result:

  $$2^k(3 \times 10^7 = (7.29 \times 10^{47})$$

  Rearranging and solving, we get:

  $$k = 134.158 \text{ doublings}$$

  Now t = 1.5k, so t = 201.225 years!!

Chapter 5 Questions:

5.1: The answer to this question depends on what is meant by the records "not being related to one another". We know that AES has diffusion and confusion properties built in - so even if each record had a similar header/body/footer section, a change of a few bits should produce a largely different ciphertext.

If "not related" means that no record ever repeats, and an attacker cannot probe the encryption algorithm, then ECB mode is acceptable, as an adversary cannot easily perform traffic analysis. We can parallelize ECB mode here (run multiple instances, and encrypt our database quickly.

5.3: Suppose we have the encryption key K_{128}, and we have two (p/c) pairs. One pair has a plaintext that is unknown, the other has a plaintext that is padded with 1s' (0xFF.. file).
Based on the way CBC inititalizes and operates, it is possible to recover the IV.
Suppose that on re-initializing the IV for the day, the first file sent is the padded file. We have (x_1,y_1) and K, so using our decryption equation:

$$d(y_1) = d_k(e_k(x_1 \oplus IV)) = x_1 \oplus IV$$

so we can cancel the XOR operation by using the x_1 plaintext again:

$$x_1 \oplus x_1 \oplus IV = IV$$

Note, if our known (p/c) pair occurs later, we would require more plaintexts in order to solve for IV. For example, if our padded file occured in the third position, we would have to invert the following equation to get the IV:

$$x_3 = e_k^{-1}(y_4) \oplus e_k(x_2 \oplus e_k(x_1 \oplus IV))$$

But we do not know x_2 or x_1 here, so it is in vain.

5.9: For CTR mode, we encrypt a concatinated IV+Counter bit string, and XOR it with the block input. In general, the maximum amount of information we can encrypt (for AES) is 8 bytes $\times$ 2^c, where c is the number of bits for our counter.

To find the minimum number of bits C to encrypt 1tb of data, we divide 1tb by 8 and take the logarithm:

$$log_2(\frac{1tb}{8}) = log_2(\frac{1024^4 bits}{8}) = 37$$

Chapter 6: Practise Questions:

6.3

$$\frac{120*119}{2} = 7140$$

Key pairs to maintain (!!).

6.4

6.5.1 Let $r_0 = 7469$ and $r_1 = 2464$. Run the Standard Euclidian Algorithm:

Iteration:	Mod Div:	gcd() equalities:
1	7469 = 3(2464) + 77	g(7469,2464) = g(2464,77)
2	2464 = 32(77) + 0	g(2464,77) = g(77,0)
3	0	END

So our GCD is 77.

6.5.2 Let $r_0 = 4001$ and $r_1 = 2689$. Run the Standard Euclidian Algorithm:

Iteration:	Mod Div:	gcd() equalities:
1	4001 = 1(2689) + 1312	g(4001,2689) = g(2689,1312)
2	2689 = 2(1312) + 65	g(2689,1312) = g(1312,65)
3	1312 = 20(65) + 12	g(1312,65) = g(65,12)
4	65 = 5(12) + 5	g(65,12) = g(12,5)
5	12 = 2(5) + 2	g(12,5) = g(5,2)
6	5 = 2(2) + 1	g(5,2) = g(2,1)
7	2 = 2(1) + 0	g(2,1) = g(1,0)
8	0	END

So our GCD is 1. Both numbers are prime.

6.6.1: Lets apply the Euclidian Algorithm

6.8.1: m = 12 = $(2^2)(3)$ for our prime factorization. Apply the formula:

$$\Phi(12) = (2^2 - 2^1)(3^1 - 3^0) = (2)(2) = 4$$

Iteratively checking for co-primes, we see that 1, 5, 7 and 11 have a gcd(12,*) = 1. Everything else shares a prime factor.

6.8.2: m = 15 = (3)(5) for the factorization, so:

$$\Phi(15) = (5^1 - 5^0)(3^1 - 3^0) = (4)(2) = 8$$

Our 8 elements that are co-prime are: (2,4,6,7,8,11,13,14)

6.8.3: m = 26 = (2)(13), apply the formula:

$$\Phi(26) = (2^1 - 2^0)(13^1 - 13^0) = (12)$$

Our 12 co-prime elements are: 1,3,5,7,9,11, 15,17,19,21,23,25.

6.9.1: If m is prime, then m is its own prime factorization. This trivializes our Euler Phi Function:

$$\Phi(m) = \Phi(p) = (p^1 - p^0) = (p - 1) = p(1 - \frac{1}{p})$$

6.9.2: If m = pq, two primes, then we can extend the formula in 6.9.1 to:

$$\Phi(m) = \Phi(pq) = (p^1 - p^0)(q^1 - q^0) = (p - 1)(q - 1)$$

Let's verify the formula. For m = 15, $\Phi((3)(5)) = (3 - 1)(5 - 1) = (2)(4) = 8$, and m = 26 $\Phi((13)(2)) = (2 - 1)(13 - 1) = 12$

6.10.1: a=4, n = 7. Here, FLT applies as n is prime. So lets calculate the inverse:

$$4^{7-2} mod 7 \equiv 4^5 mod 7 \equiv (16 mod 7)(64 mod 7) = 2 mod 7$$

Check: Now 4*2 mod 7 = 1 OK.

6.10.2: a=5, n=12. Our power is not prime, so use the Euler Theorem: $\Phi(12) = 4$

$$5^3 mod 12 = (25 mod 12)(5 mod 12) = (1)(5)$$

Check: Now 5*5 mod 12 = 1 OK.

6.10.3: a=6, n=13. Here, we have a prime power again, so we can use FLT, however we end up with:

$$6^1 1 mod 13$$

Which is a huge number. We apply repeated mod 13 division to clip the number down:

$$(36 mod 13)(36 mod 13)(36 mod 13)(36 mod 13)(36 mod 13)(6 mod 13) = (10^5)(6) mod 13$$

$$(100 mod 13)(100 mod 13)(60 mod 13) = (9)(9)(8) mod 13 = (81 mod 13)(8 mod 13)$$

$$(3)(8) mod 13 = 11$$

Check: 6*11 mod 13 = 1 OK.

6.13:

Chapter 7: Problems:

7.1.1: Consider $e_1 = 32$, $e_2 = 49$. n = 41 x 17 = 697, and $\Phi(n) = 640$. Now, both e's need to be in the set of ∈ {0....639}. Check: gcd((32,640) ≥ 1, but gcd(49,640) = 1. As stated in RSA Key Geneartion Box, e must have a gcd of 1 with $\Phi(n)$. So we choose e_2.

7.1.2: To obtain the private key, we calcualte the EEA and extract the T coefficent. As a reminder:

$$gcd(\Phi(n), e) = s\Phi(n) + te$$

$$d = t mod \Phi(n)$$

.

EEA:

i:	r_i	$r_{i-2} = q_{i-1}r_{i-1} + r_0$:	$r_i = s_i r_0 + t_i r_i$:
0	640	NA	NA
1	49	NA	NA
2	3	$640 = (13)(49) + 3 \rightarrow$	$3 = (1)(640) - (13)(49)$
.	.	...	$r_2 = (1)r_0 - 13r_1$
3	1	$49 = (16)(3) + 1 \rightarrow$	$1 = (1)(49) - (16)(3)$
.	.	...	$r_3 = (1)r_1 - 16r_2$
.	.	...	$r_3 = (1)r_1 - 16(r_0 - 13r_1)$
.	.	...	$r_3 = (1)r_1 - 16r_0 + 208r_1)$
.	.	...	$r_3 = 209r_1 - 16r_0$
4	.	$3 = (1)(3) + 0$	END

So $gcd(640, 49) = -16(640) + 209(49)$, and t = 209. as t is less than $\phi(n)$, modular division is not required.

Check: $de \equiv 1 mod \Phi(n) \rightarrow 2401 mod \Phi(n) = 1!$

7.2.1 : Given our parameters x=2, e=79 = $(1001111)_2$ and m=101, lets calculate each step:

Step:	X Calc	Base10 Exp	Base2 Exp:
0	$x_0 = 2^1 = 2$	1	(1)
1	$x_1 = (2^1)^2 = 4$	2	(10)
2	$x_2 = (2^2)^2 = 16$	4	(100)
3	$x_3 = (2^4)^2 2 = 512$	9	(1001)
4	$x_4 = (2^9)^2 2 = 524228$	19	(10011)
5	$x_5 = (2^{19})^2 2 = ...$	39	(100111)
6	$x_6 = (2^{39})^2 2 = ...$	79	(1001111)

So computationally, we would have a very large number in a floating point / integer register in our CPU, and then we would compute our modular division. $2^{79} mod 101 = 42$

7.2.2 : Given our parameters x=3, e=197 = $(1001111)_2$ and m=101, lets calculate each step:

Step:	X Calc	Base10Exp	Base2Exp:
0	$x_0 = 2^1 = 2$	1	(1)
1	$x_1 = (2^1)^2 2 = 8$	8	(11)
2	$x_2 = (2^3)^2 = 64$	6	(110)
3	$x_3 = (2^6)^2 = 4096$	12	(1100)
4	$x_4 = (2^1 2)^2 = 16777216$	24	(11000)
5	$x_5 = (2^2 4)^2 2 = ...$	49	(110001)
6	$x_6 = (2^4 9)^2 = ...$	98	(1100010)
6	$x_6 = (2^9 8)^2 2 = ...$	197	(11000101)

So computationally, we would have a very large number in a floating point / integer register in our CPU, and then we would compute our modular division. $2^{197} mod 101 = 38$

7.3.1: Let p=3, q=11 x=5 and d=7. Then n=33, and $\Phi(n) = 20$, forcing our e,d ∈ {0...19}. As These numbers are so small, we can guess that e = 3 by inspection.

Encryption:

$$y = e_{kpub}(x) \equiv x^3 mod n$$

.

So calculating: $5^3 mod 33 = 125 mod 33 = 26 \equiv y$

Decryption: To check:

$$x = d_{kpriv}(y) \equiv y^d mod n$$

Calculating: $26^7 mod 33 = 5 \equiv x$
So our roundabout encryption works.

7.3.2: Let p=5, q=11, e=3 and x=9. Then n=55, and $\Phi(n) = 40$. So e,d ∈ {0...39}. Let d = 27, then 3*27 = 81, so it can be seen it is the inverse by inspection.

Encryption: $x^e mod n = 9^3 mod n = 729 mod 55 = 14 \equiv y$

Decryption: $y^d mod n = 14^2 7 mod 55 = 9 \equiv x$

So it works out.

7.5.1: Recall the Attacker's Problem:

$$\Phi(n) = (p-1)(q-1)$$

$$d^{-1} \equiv e mod \Phi(n)$$

$$x \equiv y^d mod n$$

The attacker knows y and n. if D is small, then equation 3 can be quite easily brute forced. So, d must be relatively large.

7.5.2: Looking at our chart in Table 7.3, we see that a 664 bit number was factored in 2005. If we consider that our textbook was written in 2008 - giving estimates of 10-15 years for a 1024 bit number (by large institutions). More recent developments show that 829 bit numbers (for n) have been factored as of 2020 [1]. As we are close to factoring 1024 bit numbers, this means RSA should be moved to 2048 bits as standard. Using the "0.3t" rule mentioned in the textbook, d should be minimum 615 bits!

7.7.1: Let p=31, and q=37. N=1147, $\Phi(n) = 30(36) = 1080$. So e,d $\in$ {0.....1079}. e=17, so lets get the private key. Since we are not running the EEA, we can just get this via a calculator: d=953 . Check: (17)(953) mod 1080 =1. OK.

Given ciphertext y=2, lets transform our problem, calculate the CRT, and then transform back:

Transform to CRT Range:

$$y_p \equiv 2 mod 31 = 2$$

$$y_q \equiv 2 mod 37 = 2$$

Exponentiation in CRT Range:

$$d_p \equiv 953 mod 30 = 23$$
$$d_1 \equiv 953 mod 37 = 17$$

Then:

$$x_p = 2^{23} mod 31 = 8$$

$$x_q = 2^{17} mod 37 = 18$$

Transform back to Domain:
First lets calculate coefficients c:

$$c_p = q^{-1} mod p \text{ and } c_q = p^{-1} mod q$$

Inverses are calcualted relative to the other prime (NOT in Z_n). Using an online calculator, cp = 26 mod 31, cq = 6 mod 37.

Putting it all together:

$$x \equiv [(37)(26)]8 + [(31)(6)]18 mod 1147$$

$$= 8440 mod 1147 = 721$$

As we wanted.

7.8.1: For a given modulus of length t bits, we require p,q = t/2 bits. Then, for each of p and q, we expect to test:

$$\frac{2}{ln(2^{t/2})} = \frac{2}{(t/2)ln(2)}$$

Tests for each prime. So double this, for both p and q.

7.11.1: y = 1141, kpub = (n=2623,e=2111). Consider the encryption formula:

$$y \equiv x^e mod n$$

We have all pieces of information but x. Why can we not just rearrange the equation? Given the tools we have available, we can't isolate x (no discrete logarithm). to isolate x with inverses would put x on boht sides of the equation. So it can't be done.

However, given our small n value (which bounds everything), we could bruteforce x, as Schoolbook RSA is deterministic.

7.11.2: Our efficient formula is (p-1)(q-1). We cannot immediately use this, as we do not know p and q.

7.11.3: Because n is so small, we can easily factor it using online tools (Wolfram Alpha), which gives p=43,q=61. Much like our calculations above, we can get $\Phi(n)$, run the EEA algorithm or just use an online calculator (as the nubmers are smal).

More generally, computing p and q cannot be done with ease for large key sizes (1024 bits or more).

References

[1] https://en.wikipedia.org/wiki/RSA_numbers#RSA-250

[2]

[3]

[4]

[5]

[6]