Ungraded Homework Solutions

CSC 152 – Cryptography

Please notify me of any errors you find. If you need help, ask.

1) *GF(16) is defined like GF(256) except the polynomials all have degree less than 4 and the modulus is $x^4 + x + 1$. Calculate the following, each digit representing a field element in hexadecimal. (a) $5 + F$. (b) $5 - F$. (c) $5 \cdot F$. (d) $5/F$. Note that $5 - F$ is shorthand for $5 + (-F)$ where $-F$ is F's additive inverse, and $5/F$ is shorthand for $5 \cdot (F^{-1})$ where F^{-1} is F's multiplicative inverse.*

(a) $(x^2 + x^0) + (x^3 + x^2 + x^1 + x^0) = x^3 + x^1 = A$, or recognizing that addition is just xor of the coefficients $\texttt{0b0101} \oplus \texttt{0b1111} = \texttt{0b1010} = A$.
(b) $5 - F$ is shorthand for $5 + (-F)$, and $-F$ is the value that when added to F yields 0 (in this case $F + F = 0$, so $-F = F$). So, $5 - F = 5 + (-F) = 5 + F = A$.
(c) $(x^2 + x^0) \cdot (x^3 + x^2 + x^1 + x^0) = x^5 + x^4 + x^1 + x^0$. But, since this is not an element of GF(16), we must reduce it modulo $x^4 + x + 1$. $(x^5 + x^4 + x^1 + x^0)/(x^4 + x + 1)$ gives a quotient of $x + 1$ and a remainder of $x^2 + x$, so the answer is $x^2 + x = 6$.
(d) $5/F$ is shorthand for $5 \cdot F^{-1}$, and F^{-1} is the value that when multiplied by F yields 1. You could brute-force try all 15 candidates until you found the inverse which give us $(x^3 + x^2 + x^1 + x^0)^{-1} = x^3$. So, the answer is $(x^2 + x^0) \cdot (x^3) = x^5 + x^3 = x^3 + x^2 + x$.

2) *The AES S-box is a permutation, and therefore could be used in the modes of operation we learned (ECB, CBC, CTR, OFB). Use the S-box in each of the modes to encrypt "abc". In modes that need padding use 10^* padding. For modes that need an IV use 01010011. For modes that need a nonce, use 0110. Note that the S-box would never be used this way, I'm just using it as a readily available permutation for practice.*

ECB and CBC both require padding, so converting "abc" to ASCII and padding gives us the padded plaintext of 0x61 0x62 0x63 0x80. ECB produces ciphertext 0xEF 0xAA 0xFB 0xCD and CBC produces ciphertext 0x23 0x83 0xE1 0xEF. For CTR and OFB there is no padding, so we generate 3 bytes of keystream and xor with our plaintext. The CTR counter blocks, beginning the counter at 1, are 0x61 0x62 0x63, and produce keystream 0xEF 0xAA 0xFB which xor'd with 0x61 0x62 0x63 yields ciphertext 0x8E 0xC8 0x98. (Note that it's purely coincidence in this example that the data to encrypt matches the CTR blocks.) For OFB, we begin with IV 0x53 and the keystream produced is 0xED 0x55 0xFC, which xor'd with 0x61 0x62 0x63 yields ciphertext 0x8C 0x37 0x9f.

3) *Let's say that a ciphertext that was created using a mode-of-operation has a single bit toggled in it's i-th block before decryption. How damaging is it to the decryption? Describe the damage with respect to errors in the resulting plaintext blocks (eg, "plaintext block i has a single bit error", or "all plaintext blocks later than i look random", etc). Do this for each of the modes ECB, CBC, CTR, OFB.*

ECB) The i-th plaintext block is scrambled. CBC) The i-th plaintext block is scrambled and the next plaintext block has a single bit error. CTR) The i-th plaintext block has a single bit error. OFB) The i-th plaintext block has a single bit error.

4) *Let's say that the key used with AES-128 is `0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F`. Compute the first two round keys used by AES-128 in this case (ie, compute k_0 and k_1 in Fig 4.2 which is also $W[0]$ through $W[7]$ in the Fig 4.5).*

The first round key is just the key supplied by the user. The second round key can be computed following

Fig 4.5 from the reading.

$$\begin{aligned} W[4] &= W[0] \oplus g(W[3]) \\ &= W[0] \oplus g(\texttt{0x0C}, \texttt{0x0D}, \texttt{0x0E}, \texttt{0x0F}) \\ &= W[0] \oplus (S(\texttt{0x0D}) \oplus 1, S(\texttt{0x0E}), S(\texttt{0x0F}), S(\texttt{0x0C})) \\ &= W[0] \oplus (\texttt{0xD6}, \texttt{0xAB}, \texttt{0x76}, \texttt{0xFE}) \\ &= (\texttt{0x00}, \texttt{0x01}, \texttt{0x02}, \texttt{0x03}) \oplus (\texttt{0xD6}, \texttt{0xAB}, \texttt{0x76}, \texttt{0xFE}) \\ &= (\texttt{0xD6}, \texttt{0xAA}, \texttt{0x74}, \texttt{0xFD}) \end{aligned}$$

Once you have $W[4]$ the rest are easy: $W[5] = W[4] \oplus W[1]$, $W[6] = W[5] \oplus W[2]$, and $W[7] = W[6] \oplus W[3]$.

5) *Using the k_0 and k_1 computed in Problem 1, what is the value of the evolving AES block after "round 1" in Fig 4.2 if initially the AES block ("plaintext x" in Fig 4.2) is* `0xFF, 0xFE, 0xFD, 0xFC, 0xFB, 0xFA, 0xF9, 0xF8, 0xF7, 0xF6, 0xF5, 0xF4, 0xF3, 0xF2, 0xF1, 0xF0`

After the first KeyAddition, the block is the byte `0xFF` repeated 16 times. After the first ByteSubstitution, the block is the byte `0x16` repeated 16 times. After the first ShiftRows, the block is the byte `0x16` repeated 16 times. For MixColumns, you could do all the multiplications and then the additions but you could also use the distributive property and factor out the common term, $2 \cdot \texttt{0x16} + 3 \cdot \texttt{0x16} + 1 \cdot \texttt{0x16} + 1 \cdot \texttt{0x16} = (2 + 3 + 1 + 1) \cdot \texttt{0x16} = 1 \cdot \texttt{0x16} = \texttt{0x16}$. So, after the first MixColumns, the block is the byte $0x16$ repeated 16 times. So, the solution to this problem is the byte $0x16$ repeated 16 times xor'd with the $(W[4], W[5], W[6], W[7])$ computed in Problem 1. Note that I chose these values to reduce your work, usually all the bytes will look random.

CE 3.1) An "idealized" block cipher is conceptually just a 2D array where the first index is the key to use and the second index is the input block. The array yields the output block. In this problem there are 2^{80} keys and 2^{64} input blocks, so the array must have 2^{144} entries. Each entry is 64 bits, or 2^3 bytes, so the table's data consumes 2^{147} bytes. Not practical. That a compact algorithm can efficiently simulate such a thing is amazing.

CE 4.1) It is not always reversible because in some circumstances multiple plaintexts map to the same padded plaintext. For example, let's say $b = 4$. Then P = 0xAABBCC becomes padded to 0xAABBCC01. But, if P = 0xAABBCC01 to begin with, then this padding scheme says to add zero 0x00 bytes, so the padded plaintext is also 0xAABBCC01. Unpadding must be unambiguous for a padding scheme to work (ie, it must be a one-to-one function).

CE 4.3) Because the nonce is the same in both cases, both ciphertexts use the same keystream K. Thus $C = K \oplus P$ which means $C \oplus P = (K \oplus P) \oplus P = K \oplus (P \oplus P) = K$. Once K is obtained, P' is obtained as $P' = C' \oplus K$. This is why a nonce must not be repeated when encrypting multiple messages with the same key.

CE 4.6) The values P'_1 and P_2 would be computed as $P'_1 = E^{-1}(C'_1) \oplus C'_0$ and $P_2 = E^{-1}(C_2) \oplus C_1$. An attacker doesn't know E^{-1} because they don't know the key in use, but because we're told $C_2 = C'_1$, it must be that $E^{-1}(C_2) = E^{-1}(C'_1)$. With a little algebra, these E^{-1} terms can be eliminated, leaving $P_2 \oplus C_1 = P'_1 \oplus C'_0$. Thus, $P'_1 = P_2 \oplus C_1 \oplus C'_0$, which are all known quantities. This shows that information leaks if two CBC ciphertext blocks match when using the same key (which happens at about the birthday bound).