Checksum 4-Bit

Checksum Value

1. Compute the value of the checksum using: Message = $BF1942_{16}$

   Message:
   First, we need to convert the message in hexadecimal to binary.
   B = 1011
   F = 1111
   1 = 0001
   9 = 1001
   4 = 0100
   2 = 0010

   Addition:
   Now that the message is in unsigned binary numbers, you can add them one by one, while discarding overflow bits.

	B		1	0	1	1
+	F		1	1	1	1
=		1	1	0	1	0
+	1		0	0	0	1
=			1	0	1	1
+	9		1	0	0	1
=		1	0	1	0	0
+	4		0	1	0	0
=			1	0	0	0
+	2		0	0	1	0
=			**1**	**0**	**1**	**0**

   Now perform two's complement of the added value 1010

   $1010 = 0101 + 1 = 0110_2$

   Convert the value to a hexadecimal so we can append it to the message

   $0110_2 = 6_{16}$

   Final message that will be transmitted is then **$BF19426_{16}$**

2. Compute the value of the checksum using: Message = $C15336_{16}$

Message:
First, we need to convert the message in hexadecimal to binary.
C = 1100
1 = 0001
5 = 0101
3 = 0011
3 = 0011
6 = 0110
0 = 0000

Addition:
Now that the message is in unsigned binary numbers, you can add them one by one, while discarding overflow bits.

	C		1	1	0	0
+	1		0	0	0	1
=			1	1	0	1
+	5		0	1	0	1
=		1	0	0	1	0
+	3		0	0	1	1
=			0	1	0	1
+	3		0	0	1	1
=			1	0	0	0
+	6		0	1	1	0
=			1	1	1	0
+	0		0	0	0	0
=			**1**	**1**	**1**	**0**

Now perform two's complement of the added value 1110

1110 = 0001 + 1 = 0010_2

Convert the value to a hexadecimal so we can append it to the message

$0010_2 = 2_{16}$

Final message that will be transmitted is then **$C1533602_{16}$**

Validating Checksum

3. Validate the checksum from Question 1

Checksum from Question 1: $BF19426_{16}$

Convert into binary
B = 1011
F = 1111
1 = 0001
9 = 1001
4 = 0100
2 = 0010
6 = 0110

Addition:

	B		1	0	1	1
+	F		1	1	1	1
=		1	1	0	1	0
+	1		0	0	0	1
=			1	0	1	1
+	9		1	0	0	1
=		1	0	1	0	0
+	4		0	1	0	0
=			1	0	0	0
+	2		0	0	1	0
=			1	0	1	0
+	6		0	1	1	0
=		1	**0**	**0**	**0**	**0**

The final result (ignoring overflow) is all zeroes, so the message with checksum is validated! In another words, the message was not corrupted!

4. Validate the checksum from Question 2
Checksum from Question 2: $C1533602_{16}$

Convert into binary
C = 1100
1 = 0001
5 = 0101
3 = 0011
3 = 0011
6 = 0110
0 = 0000
2 = 0010

Addition:

	C		1	1	0	0
+	1		0	0	0	1
=			1	1	0	1
+	5		0	1	0	1
=		1	0	0	1	0
+	3		0	0	1	1
=			0	1	0	1
+	3		0	0	1	1
=			1	0	0	0
+	6		0	1	1	0
=			1	1	1	0
+	0		0	0	0	0
=			1	1	1	0
+	2		0	0	1	0
		1	**0**	**0**	**0**	**0**

The final result (ignoring overflow) is all zeroes, so the message with checksum is validated! In another words, the message was not corrupted!

Checksum 8-Bit

Checksum Value

5. Compute the value of the checksum using: Message (all in base 16) = 32 64 96 C8$_{16}$

Numbers:
First, we need to convert the hexadecimal numbers to binary.
32 = 0011 0010
64 = 0110 0100
96 = 1001 0110
C8 = 1100 1000

Addition:
Now that the message is in unsigned binary numbers, you can add them one by one, while discarding overflow bits.

	32		0	0	1	1		0	0	1	0
+	64		0	1	1	0		0	1	0	0
=			1	0	0	1		0	1	1	0

+	96		1	0	0	1		0	1	1	0
=		1	0	0	1	0		1	1	0	0
+	C8		1	1	0	0		1	0	0	0
=			**1**	**1**	**1**	**1**		**0**	**1**	**0**	**0**

Now perform two's complement of the added value 1111 0100

1111 0100 = 0000 1011 + 1 = 0000 1100_2

Convert the value to a hexadecimal so we can append it to the message

0000 1100_2 =$0C_{16}$

Final message that will be transmitted is then **32 64 96 C8 $0C_{16}$**

6. Compute the value of the checksum using: Message (all in base 16) = 12 34 56 78_{16}

Numbers:
First, we need to convert the decimal numbers to binary.
12 = 0001 0010
34 = 0011 0100
56 = 0101 0110
78 = 0111 1000

Addition:
Now that the message is in unsigned binary numbers, you can add them one by one, while discarding overflow bits.

	12		0	0	0	1		0	0	1	0
+	34		0	0	1	1		0	1	0	0
=			0	1	0	0		0	1	1	0
+	56		0	1	0	1		0	1	1	0
=			1	0	0	1		1	1	0	0
+	78		0	1	1	1		1	0	0	0
=		1	**0**	**0**	**0**	**1**		**0**	**1**	**0**	**0**

Now perform two's complement of the added value 0001 0100

0001 0100 = 1110 1011 + 1 = 1110 1100_2

Convert the value to a hexadecimal so we can append it to the message

1110 1100$_2$ =EC$_{16}$

Final message that will be transmitted is then **12 34 56 78 EC$_{16}$**

Validating Checksum

7. Validate the checksum from Question 5
   Checksum from Question 1: 36 64 96 C8 0C$_{16}$

   Convert into binary
   32 = 0011 0010
   64 = 0110 0100
   96 = 1001 0110
   C8 = 1100 1000
   0C = 0000 1100

   Addition:

	32		0	0	1	1		0	0	1	0
+	64		0	1	1	0		0	1	0	0
=			1	0	0	1		0	1	1	0
+	96		1	0	0	1		0	1	1	0
=		1	0	0	1	0		1	1	0	0
+	C8		1	1	0	0		1	0	0	0
=			1	1	1	1		0	1	0	0
+	0C		0	0	0	0		1	1	0	0
=		1	**0**	**0**	**0**	**0**		**0**	**0**	**0**	**0**

   The final result (ignoring overflow) is all zeroes, so the message with checksum is validated! In another words, the message was not corrupted!

8. Validate the checksum from Question 6
   Checksum from Question 6: 12 34 56 78 EC$_{16}$

   Convert into binary
   12 = 0001 0010
   34 = 0011 0100
   56 = 0101 0110
   78 = 0111 1000
   EC = 1110 1100

   Addition:

	12		0	0	0	1		0	0	1	0
+	34		0	0	1	1		0	1	0	0
=			0	1	0	0		0	1	1	0
+	56		0	1	0	1		0	1	1	0
=			1	0	0	1		1	1	0	0
+	78		0	1	1	1		1	0	0	0
=		1	0	0	0	1		0	1	0	0
+	EC		1	1	1	0		1	1	0	0
=		1	**0**	**0**	**0**	**0**		**0**	**0**	**0**	**0**

The final result (ignoring overflow) is all zeroes, so the message with checksum is validated! In another words, the message was not corrupted!

CRC 3

Generating CRC

1. Calculate the CRC, and the final message using: Message = 911_{16}, Divisor = $X^3 + X^2 + 1$

   Message:
   First, we need to convert the message (dividend) in hexadecimal to binary.
   So $911_{16} = 1001\ 0001\ 0001_2$

   Polynomial:
   Second, we need to convert the polynomial (divisor) into a binary form.
   So $X^3 + X^2 + 1$ is the same as $X^3 + X^2 + 0X^1 + 1$
   And so $X^3 + X^2 + 0X^1 + 1 = 1101$

   Now what?
   Now that we have the binary representations of both the message and the divisor, we can go ahead and generate the CRC! Because this is a 3-bit CRC, you will need to pad the message with 3 zeroes at the end of the message.

   You can do the full calculation bit by bit or do faster in the way shown on Question 2.

					1	1	1	1	0	1	0	1	0	0	1	0			
1	1	0	1		1	0	0	1	0	0	0	1	0	0	0	1	**0**	**0**	**0**
					1	1	0	1											
						1	0	0	0										
						1	1	0	1										
							1	0	1	0									
							1	1	0	1									
								1	1	1									
								1	1	0	1								
									0	1	1	1							
										1	1	0	1						
											0	1	1	0					
											0	0	0	0					
												1	1	0	0				
												1	1	0	1				
													0	0	1	1			
													0	0	0	0			
														0	1	1	0		
														0	0	0	0		
															1	1	0	0	
															1	1	0	1	
																0	0	1	0
																0	0	0	0
																	0	**1**	**0**

Remainder:

Your remainder is always 1 bit less than the polynomial, so in this case 3 bits = 010.

Appended Message:

1001 0001 0001 010

Now you should convert the remainder to a hexadecimal representation. First, start with padding it with a leading zero to make it 4 bits = 0010.

Now convert it to hexadecimal, and append it to the message.

$0010_2 = 2_{16}$

9112_{16} and that's it!

2. Calculate the CRC, and the final message using: Message = $A0E_{16}$, Divisor = $X^3 + X + 1$

   Message:
   First, we need to convert the message (dividend) in hexadecimal to binary.
   So $A0E_{16} = 1010\ 0000\ 1110_2$

   Polynomial:
   Second, we need to convert the polynomial (divisor) into a binary form.
   So $X^3 + X + 1$ is the same as $X^3 + 0X^2 + X^1 + 1$
   And so $X^3 + 0X^2 + X^1 + 1 = 1011$

   Now what?
   Now that we have the binary representations of both the message and the divisor, we can go ahead and generate the CRC! Because this is a 3-bit CRC, you will need to pad the message with 3 zeroes at the end of the message.

   As mentioned in Question 1, this method is quicker. Line up the leftmost '1' values, then apply XOR to subtract.

					1			1		1	1	1	1	1	1				
1	0	1	1		1	0	1	0	0	0	0	0	1	1	1	0	**0**	**0**	**0**
					1	0	1	1											
						0	0	1	0										
								1	0	1	1								
									0	1	1	0							
										1	0	1	1						
											1	1	0	1					
											1	0	1	1					
												1	1	0	1				
												1	0	1	1				
													1	1	0	0			
													1	0	1	1			
														1	1	1	0		
														1	0	1	1		
															1	0	1	0	
															1	0	1	1	
																	0	**1**	**0**

Remainder:

The remainder is 010

Appended Message:

1010 0000 1110 010

$0010_2 = 2_{16}$

$\mathbf{A0E2_{16}}$

Validating CRC

3. Validate the CRC using the result from Question 1.

Appended message:
1001 0001 0001 010

But wait, you're thinking it should be 1001 0001 0001 0010. Remember however, that we padded the remainder 010 with a leading zero, so do not use the padded remainder, but the true remainder which should be 3 bits!

Polynomial:
Use the same polynomial = 1101

Now What?
Now that we have everything, do the modulo-2 division.

					1	1	1	1		1		1			1				
1	1	0	1		1	0	0	1	0	0	0	1	0	0	0	1	**0**	**1**	**0**
					1	1	0	1											
						1	0	0	0										
						1	1	0	1										
							1	0	1	0									
							1	1	0	1									
								1	1	1	0								
								1	1	0	1								
										1	1	1	0						
										1	1	0	1						
												1	1	0	0				
												1	1	0	1				
															1	1	0	1	
															1	1	0	1	
																	0	**0**	**0**

The 000 is the "syndrome" (again, fancy word for remainder in the phase of CRC validation). If your syndrome is all zeroes, that means no change has occurred in your original message. If your syndrome non-zero, that means there was a change in your original message.

Computers as you may already know, use 0's and 1's to communicate. The generated CRC remainder that is attached to the original message serves the purpose of checking the integrity of the original message. Because communication mediums are not perfect, there are cases where the message 1001 0001 0001 010 may change to something like 1001 0000 0000 010.

When you perform the validation on the 1001 0000 0000 010, you will know that there was a change to your original message, because your syndrome will NOT be all zeros. A computer will reject this corrupted message, and ask the sender to re-transmit.

4. Validate the CRC using the result from Question 2.

   Appended message:
   1010 0000 1110 010

Remember we want the 3 bit remainder. We only padded it with a leading zero for the sole purpose of converting it to a hexadecimal.

Polynomial:
Use the same polynomial = 1101

Now What?
Now that we have everything, do the modulo-2 division.

					1			1		1	1	1	1	1	1				
1	0	1	1		1	0	1	0	0	0	0	0	1	1	1	0	**0**	**1**	**0**
					1	0	1	1											
								1	0	0	0								
								1	0	1	1								
										1	1	0	1						
										1	0	1	1						
											1	1	0	1					
											1	0	1	1					
												1	1	0	1				
												1	0	1	1				
													1	1	0	0			
													1	0	1	1			
														1	1	1	0		
														1	0	1	1		
															1	0	1	1	
															1	0	1	1	
																	0	**0**	**0**

The 000 is the "syndrome" (again, fancy word for remainder in the phase of CRC validation). If your syndrome is all zeroes, that means no change has occurred in your original message. If your syndrome non-zero, that means there was a change in your original message.

CRC 4

Generating a CRC

5. Calculate the CRC, and the final message using: Message = A0E$_{16}$, Divisor = $X^4 + X^3 + 1$

Message:

First, we need to convert the message (dividend) in hexadecimal to binary.
So $A0E_{16} = 1010\ 0000\ 1110_2$

Polynomial:
Second, we need to convert the polynomial (divisor) into a binary form.
So $X^4 + X^3 + 1$ is the same as $X^4 + X^3 + 0X^2 + 0X^1 + 1$
And so $X^4 + X^3 + 0X^2 + 0X^1 + 1 = 11001$

Now what?
Now that we have the binary representations of both the message and the divisor, we can go ahead and generate the CRC! Because this is a 4-bit CRC, you will need to pad the message with 4 zeroes at the end of the message.

						1	1			1					1						
1	1	0	0	1		1	0	1	0	0	0	0	0	1	1	1	0	**0**	**0**	**0**	**0**
						1	1	0	0	1											
							1	1	0	1	0										
							1	1	0	0	1										
										1	1	0	0	1							
										1	1	0	0	1							
															1	1	0	0	0		
															1	1	0	0	1		
																		0	**1**	**0**	**0**

Remainder:

Your remainder will be 0100.

Appended Message:

1010 0000 1110 0100

Now convert it to hexadecimal, and append it to the message.

$0100_2 = 4_{16}$

$A0E4_{16}$ and that's it!

6. Calculate the CRC, and the final message using: Message = ABC_{16}, Divisor = $X^4 + X^3 + X + 1$

Message:
First, we need to convert the message (dividend) in hexadecimal to binary.
So $ABC_{16} = 1010\ 1011\ 1100_2$

Polynomial:

Second, we need to convert the polynomial (divisor) into a binary form.
So $X^4 + X^3 + X + 1$ is the same as $X^4 + X^3 + 0X^2 + X + 1$
And so $X^4 + X^3 + 0X^2 + X + 1 = 11011$

Now what?
Now that we have the binary representations of both the message and the divisor, we can go ahead and generate the CRC! Because this is a 4-bit CRC, you will need to pad the message with 4 zeroes at the end of the message.

						1	1		1		1	1	1	1							
1	1	0	1	1		1	0	1	0	1	0	1	1	1	1	0	0	**0**	**0**	**0**	**0**
						1	1	0	1	1											
							1	1	1	0	0										
							1	1	0	1	1										
									1	1	1	1	1								
									1	1	0	1	1								
											1	0	0	1	1						
											1	1	0	1	1						
												1	0	0	0	0					
												1	1	0	1	1					
													1	0	1	1	0				
													1	1	0	1	1				
														1	1	0	1	0			
														1	1	0	1	1			
																		1	**0**	**0**	**0**

Remainder:

Your remainder will be 1000.

Appended Message:

1010 1011 1100 1000

Now convert it to hexadecimal, and append it to the message.

$1000_2 = 8_{16}$

$\mathbf{ABC8_{16}}$ and that's it!

Validating a CRC

7. Validate the CRC using the result from Question 5.

Appended message:

1010 0000 1110 0100

The CRC remainder is 4 bits = 0100.

Polynomial:
Use the same polynomial = 11001

Now What?
Now that we have everything, do the modulo-2 division.

						1	1			1					1						
1	1	0	0	1		1	0	1	0	0	0	0	0	1	1	1	0	**0**	**1**	**0**	**0**
						1	1	0	0	1											
							1	1	0	1	0										
							1	1	0	0	1										
										1	1	0	0	1							
										1	1	0	0	1							
															1	1	0	0	1		
															1	1	0	0	1		
																		0	**0**	**0**	**0**

The 0000 is the "syndrome" (again, fancy word for remainder in the phase of CRC validation). If your syndrome is all zeroes, that means no change has occurred in your original message. If your syndrome non-zero, that means there was a change in your original message.

8. Validate the CRC using the result from Question 6.

Appended message:
1010 1011 1100 1000

The CRC remainder is 4 bits = 1000

Polynomial:
Use the same polynomial = 11011

Now What?
Now that we have everything, do the modulo-2 division.

						1	1		1		1	1	1	1							
1	1	0	1	1		1	0	1	0	1	0	1	1	1	1	0	0	**1**	**0**	**0**	**0**
						1	1	0	1	1											
							1	1	1	0	0										
							1	1	0	1	1										
									1	1	1	1	1								
									1	1	0	1	1								
											1	0	0	1	1						
											1	1	0	1	1						
												1	0	0	0	0					
												1	1	0	1	1					
													1	0	1	1	0				
													1	1	0	1	1				
														1	1	0	1	1			
														1	1	0	1	1			
																		0	**0**	**0**	**0**

The 0000 is the "syndrome" (again, fancy word for remainder in the phase of CRC validation). If your syndrome is all zeroes, that means no change has occurred in your original message. If your syndrome is non-zero, that means there was a change in your original message.

1. Use the Extended Euclidean Algorithm to find the mod 36 inverse of 7. You must show all work.

First, we apply the standard Euclidean Algorithm:

Equation 0: $36 = 5(7) + 1$
Equation 1: $7 = 7(1) + 0$

Since the last nonzero remainder occurs in Equation 0, the inverse we are looking for is y_1.

Here is our table of y values (computed using the formula above):

i	y_i
0	0
1	1
2	$y_0 - (y_1)(q_0) = 0 - (1)(5) = -5$

Since the y_2 value is -5, we need to add 36 to it to get a valid modular inverse 31. Thus, the modulo 31 inverse of 7 is 1, which is our desired answer. As a check on our work, we compute (31 * 7) % 36 = 1, which confirms that 1 and 7 are inverses mod 36.

2. Given the RSA private key { d, n } = { 23, 14 }, use modulo reduction to decipher the message "10". You must show all steps and clearly state your answer.

$M = C^d \bmod n = 10^{23} \bmod 14$

$= ((10^5 \bmod 14)^4 \bmod 14)(10^3 \bmod 14) \bmod 14$

$= ((100000 \bmod 14)^4 \bmod 14)(10^3 \bmod 14) \bmod 14$

$= (12^4 \bmod 14)(10^3 \bmod 14) \bmod 14$

$= (12^2 \bmod 14) (12^2 \bmod 14) (6) \bmod 14$

$= (4) (4) (6) \bmod 14$

$= (16 \bmod 14) (6) \bmod 14$

$= (2) (6) \bmod 14$

$= 12$

3. Is 3 a primitive root of 13? If your answer is 'yes', you must show all computations that prove this result. If your answer is

'no', you must show sufficient computations to prove your answer

Correct answer is 'no'

3^1 mod 13 = 3 mod 13 = 3
3^2 mod 13 = 9 mod 13 = 9
3^3 mod 13 = 27 mod 13 = 1
3^4 mod 13 = 81 mod 13 =3 ← first duplicate result
3^5 mod 13 = 243 mod 13 = 9 ← second duplicate result
3^6 mod 13 = 729 mod 13 = 1 ← third duplicate result
3^7 mod 13 = 2187 mod 13 = 3 (and so on…)
3^8 mod 13 = 6561 mod 13 = 9
3^9 mod 13 = 19683 mod 13 = 1
3^{10} mod 13 = 59049 mod 13 = 3
3^{13} mod 13 = 177147 mod 13 = 9
3^{12} mod 13 = 531441 mod 13 = 1
3^{13} mod 13 = 1594323 mod 13 = 3

Since all values are NOT distinct mod 13, 3 is NOT a primitive root of 13

4. In setting up to do RSA public key encryption, let p = 3 and q = 23 be the two initial prime numbers. Also let e = 5 be a randomly chosen value less than and relatively prime to Φ(n).

Given the values above, compute the following. You must show all work. Given the values above, compute the following.

(a) What is the value of n?

n = p * q = 3 * 23 = 69

(b) What is *Φ(n)* ?

Φ(n) = (p-1) * (q-1) = 2 * 22 = 44

(c) Show that the value d = 9 is the modulo *Φ(n)* inverse of e.

(e * d) mod *Φ(n)* = (5 * 9) mod 44 = 45 mod 44 = 1

(d) What is the corresponding public key for these values?

public key = { e, n } = { 5, 69 }

(e) What is the corresponding private key for these values?

private key = { d, n } = { 9, 69 }