Network Security and Cryptography, Summative Assessment 1

Submission Deadline 14 November 2024, Thursday 9.59 AM UK time

You must type your answers in a word processing system, and create a PDF. We will not accept handwritten and scanned or photographed answers. Please submit your PDF on Canvas.

1. Consider the AES-128 key schedule algorithm, with the key `4247316644F4823070F4744FA232172C`.

   Find the subkeys k_1 and k_2. **[4 points]**

   Answer:
   `k0 is 0x4247316644F4823070F4744FA232172C`
   `k1 is 0x60B7405C2443C26C54B7B623F685A10F`
   `k2 is 0xF585361ED1C6F4728571425173F4E35E`

2. The file at

   `https://canvas.bham.ac.uk/files/17789736/download`

   is encrypted using AES-128 in CTR mode, using:

   key = `2B75B85C6CD84D0AB432D228ADC2060D`
   nonce (a.k.a. IV) = `85C2B686921E512AFF0DB3C67F6D8D97`.

   Decrypt it and describe the image you get. *Hint. The plaintext file is an image in JPG format. To decrypt, use any suitable software or online tool you like. One option is* `openssl`. **[4 points]**

   Answer: You can use the command

   ```
   openssl enc -aes-128-ctr -in Ex2_2_ciphertext -out t.jpg
   \
        -K D2B75B85C6CD84D0AB432D228ADC2060 \
        -iv 785C2B686921E512AFF0DB3C67F6D8D9
   ```

   It decrypts to a picture of a penguin made with wool.

3. In 60 words or less, explain why an encryption system that satisfies IND-CPA can't be broken by machine learning. **[4 points]** **Answer:** An encryption system that satisfies IND-CPA ensures that ciphertexts are indistinguishable from random data. This prevents machine learning from discovering patterns or features in the ciphertext, making it exceedingly difficult to differentiate between encryptions of different plaintexts, ensuring robust security against machine learning-based attacks.

4. Find a printable string x such that the first 20 bits of SHA-256(x) are zero. *Note: the x you submit must be different from the x that everyone else submits.* **[4 points]**

   Answer: There are many answers. Note that you can guarantee your x will be different from other people's by starting your search with a random string.

5. Fred proposes to define a new hash function which takes an input of up to 2^{32} bits, as follows:

```
INPUT: bitstring message m of length < 2**32 bits
Pad m with a 32-bit encoding of the length of m
c := emptystring
while there are still bits to take from m:
   Take the next bit from m and append it to c
   c := c << 1
   Take the next |c| bits from m and xor them into c

Truncate c to 64 bits, if necessary
OUTPUT: c
```

Here, `|c|` means the length of c, and `<< 1` means rotate the string left by one bit. If there are insufficiently many bits to take from m, then we pad m with 0s. Prove to Fred that his idea is inadequate, by finding a collision for Fred's proposed function. **[4 points]**

Answer: Here is a table of outputs:

inp	out
0	0000000
1	1000000
10	1000000
11	0000000
100	1000000
101	1000100
110	0000000
111	0000100
1000	00000001
1001	00010001
1010	00010001
1011	00000001
1100	00000000
1101	00010000
1110	00010000
1111	00000000
10000	00000001
10001	00000000

There are lots of collisions.

6. The "ciphers.zip" file (linked from the Canvas page) contains some text files with RSA PKCS encrypted ciphertext. The files can be decrypted using the secret key key.pem (linked from the Canvas page) and openssl rsautl.

   (a) What is the modulus N of the secret key in key.pem. **[2 point]**

   (b) Decrypt the file matching the last 4 digits of your student id and write down the result. **[3 point]**

Answer: A. Use "openssl rsa -in key.pem -noout -modulus". Ignore the first 00.

$00 : b8 : 6d : f0 : 78 : c3 : 30 : 1c : 0f : db : e1 : 42 : e2 : d7 : 21 :$
$03 : 9d : d6 : ca : e4 : 42 : 8e : 5d : 50 : ef : 25 : 04 : a5 : 54 : 9f :$
$8c : a6 : a1 : 00 : bf : e9 : fd : 56 : 87 : 7f : e8 : cf : f4 : 88 : 65 :$
$56 : 93 : 5c : 11 : 49 : d3 : e6 : 99 : c3 : 08 : 5e : b4 : 99 : 09 : 79 :$
$a0 : 64 : 99 : 7e : 7c : cf : 28 : 41 : 95 : 63 : 3b : 82 : d2 : 5c : 7f :$
$9b : dd : 1f : 12 : 8f : e7 : 2e : 8a : 5c : 8b : 75 : c6 : 3e : 39 : 35 :$
$a4 : 33 : e5 : 59 : 2a : 1e : b7 : 35 : eb : 04 : 67 : 8a : a9 : fe : 44 :$
$ae : 19 : 87 : d4 : c8 : 81 : 56 : c5 : c0 : cb : 7a : 2c : 8c : 86 : 78 :$
$2d : a3 : ee : 2e : 08 : 2a : 63 : e2 : 0f : 2d : 5f : 88 : a1 : c2 : e9 :$
$e1 : 0d : 67 : dc : 0b : 12 : 0e : 4c : 6d : 58 : 14 : 14 : 9c : 10 : 8e :$
$fa : 38 : 4f : 5d : 1d : e4 : ab : ad : 59 : de : d6 : 3c : 21 : af : cb :$
$3f : 5b : 03 : af : 04 : 35 : 15 : fc : 5c : 13 : 54 : 05 : 71 : 25 : 11 :$
$d0 : 7e : b3 : 75 : 47 : f6 : 03 : ca : 7f : 62 : 06 : 33 : 30 : ce : 77 :$
$2f : 4d : c0 : 7e : 5d : 8d : e1 : 96 : 44 : 9c : 66 : 8e : a5 : 7f : a9 :$
$09 : 24 : 88 : d2 : fa : 72 : b9 : f6 : fa : 84 : 55 : d7 : 1d : 43 : 81 :$
$11 : 6c : 5c : 8f : 18 : f9 : f7 : ef : a9 : c9 : 22 : 06 : 7b : a8 : d4 :$
$a6 : a1 : c0 : 3a : 74 : 0e : 88 : 0f : ee : ea : 06 : 84 : 9c : c5 : 6b :$
$95 : 9b$

B. Extract the files in the folder ciphers. Use " openssl rsautl -decrypt -inkey key.pem -in ./ciphers/cipher$i.txt -out ./decrypt/files$i.txt". For correctness check, check with the files in files.zip in canvas.

7. Consider the following RSA based encryption scheme that we will call BadModPKC. H_1 and H_2 are two hash functions mapping onto $\mathbb{Z}_n^*$

```
Procedure Keygen($1^\lambda$)
01 :  Choose two random $\lambda/2$-bit primes $p$ and $q$
02 :  $n = p \cdot q$
03 :  $\phi = (p-1)(q-1)$
04 :  Select $e$ such that
        $1 < e < \phi$ and $\gcd(e, \phi) = 1$
05 :  Compute $d$ such that
        $1 < d < \phi$ and $ed \equiv 1 \pmod{\phi}$
06 :  Set $PK = (e, n)$
07 :  Set $SK = (d)$
08 :  return $(PK, SK)$
```

```
Procedure Encrypt($PK, m$)
// We assume $m \in \mathbb{Z}_n^*$
01 :  $r \xleftarrow{\$} \mathbb{Z}_n^*$
02 :  $c_1 = r^e \mod n$
03 :  $c_2 = m \cdot H_1(r) \mod n$
04 :  $c_3 = r \cdot H_2(m) \mod n$
05 :  return $c = (c_1, c_2, c_3)$
```

Show that BadModPKC is not IND-CPA secure. **[5 point]**

Answer: Let $m_0 = 1$ and m_1 be any other element of $\mathbb{Z}_n^*$. After receiving the ciphertext $c^* = (c_1, c_2, c_3)$ the adversary computes $r = c_3/H_2(1) \mod n$, and checks whether $c_2 = H_1(r) \mod n$. If yes, then the adversary outputs 0 else the adversary outputs 1.