Information Systems Security
Mandatory TP 4 - Theoretical Part Correction

December 1st, 2021

Theoretical part

Exercise 1 : Entropy and Asymetrical Ciphers

1. Given the ASCII 8-bit alphabet (we consider every character to be printable and usable), compute the entropy of choosing randomly a key for Caesar cipher.

   In ASCII 8-bit, there are $2^8 = 256$ different characters, and thus 256 different keys for Caesar (all possible shifts from 0 characters to 255 characters).
   So the entropy of choosing a random key K is :
   $H(K) = \sum_1^{256} p(k_i) log(\frac{1}{k_i}) = \sum_1^{256} \frac{1}{256} log(256) = log(256) = 8$

2. Given the ASCII 8-bit alphabet (we consider every character to be printable and usable), compute the entropy of choosing randomly a key for monoalphabetical substitution.

   A key in monoalphabetical substitution is a random permutation of the 2^8 characters : we have 256 possible choices for the first character, then 255 for the second one, then 254, ...
   So this means we have 256! different keys, all equiprobable, giving us the following entropy for the key K :
   $H(K) = \sum_1^{256!} p(k_i) log(\frac{1}{k_i}) = \sum_1^{256!} \frac{1}{256!} log(256!) = log(256!) \simeq 1684$

3. Given the ASCII 8-bit alphabet (we consider every character to be printable and usable), compute the entropy of choosing randomly a key of length k for a Vigenere cipher.

   A Vigenere key of length k is just k characters from the alphabet, with the following entropy :
   $H(K) = k * E(char) = 8k$

4. From now on, we consider a deck of 54 cards : we have four colors (♠, ♡, ♢, ♣) with 13 cards each (numbered from 1 to 10, plus the Jack, Queen and King), and two Jokers.

What would be the entropy of drawing a suite of 5 cards from the deck one by one, with replacement (i.e. we put the card back in the deck and shuffle again before drawing another card) ?

We draw 5 cards randomly, from a deck of 54 cards. If we consider both jokers to be different cards, then each card c_i has the same probability, which gives us :
$H(5Cards) = 5H(1Card) = 5\sum_1^{54} p(c_i)log(\frac{1}{c_i}) = 5\sum_1^{54} \frac{1}{54}log(54) = 5log(54) \simeq 28.77$

5. And what would be the entropy of a suite of 5 cards drawn from the deck without replacement ?

   Same thing, but this time the entropy changes slightly for each card :
   $H(5Cards) = H(FirstCard)+H(SecondCard)+H(ThirdCard)+H(FourthCard)+H(FifthCard)$
   $= log(54) + log(53) + log(52) + log(51) + log(50) \simeq 28.50$
   (We can also get the same result by computing directly the entropy of the 5 cards, for which we have 54*53*52*51*50 possible combinaisons, each equiprobable)

6. And if we only keep hands where we have a hand of 5 cards with one Joker and one card from each color, what is the entropy ?

   First thing to notice : once again, each possible hand is equiprobable. That means we just have to compute how many hands there are.
   To do that, we will slice this into 5 subcases, depending on whether the joker is in position 1, 2, 3, 4 or 5.
   For a joker in first place, we have the following number of possibilities : $2 * 52 * 39 * 26 * 13 = 1370928$ possibilities (first card is one of the two jokers, second card is any non-joker, third card is any non joker from a different color, and so on).
   Similarly, for a joker in second position, we have the same number of hands : $52 * 2 * 39 * 26 * 13$. And so on for each joker position.
   That means we have in total $5 * 1370928 = 6854640$ possibilities of hands, all equiprobable, which gives us the following entropy :
   $H(Hand) = \sum_1^{6854640} p(h_i)log(\frac{1}{h_i}) = \sum_1^{6854640} \frac{1}{6854640}log(6854640) = log(6854640) \simeq 22.71$

Exercise 2 : Symmetrical and Asymmetrical Encryption

1. Of the three following functions f, g and h, which one would be usable as an encryption function for a block cipher with 128 bit block size ? Why ?

   Here, x is a 128 bit block, that we can write as $x = x_1 \parallel x_2$ that are its two halves of 64 bits, and k is a 128 bit key that we can express as two halves (k_1, k_2) the same way :

   - $f(x) = SHA - 256(AES(x, k))$
   - $g(x) = DES(x_1, k_2) * DES(x_2, k_1)$ *($*$ is the multiplication)*

- $h(x) = DES(x_1 \oplus x_2, k_1) \cdot DES(x_1 \oplus k_1, k_2)$ *($\cdot$ is the concatenation)*

These functions need one property to be used in block ciphers : they need to be invertible. This means we can only use the third one :
- The first function is not usable, as even if the AES part can be inverted, we don't know if SHA-256 is bijective over 128 bit entries.
- The second function is not usable either : as the result is the product of two 64 bits parts, we can find a lot of collisions with different products giving the same result.
- The third function can be used : we can't directly find x_1 and x_2 from their XOR in the first part. However, we can invert the second DES to find x_1 (using k_1), and then use that to find x_2 after inverting the first DES.

2. You get the following keys for RSA : (n=143, e=21), (n=147, e=37), (n=161, e=35).
Which one of these three keys is a valid RSA key ? Compute d for that valid key.

   - The first key is not valid : $143 = 11 * 13$, so $\Phi(n) = 10 * 12 = 120 = 2 * 2 * 2 * 3 * 5$. Thus $e = 21 = 3 * 7$ is not prime with $\Phi(n)$ (their pgcd is equal to 3), and thus is not invertible.
   - The second key is not valid either : 147 is not the product of two primes, as 147 = 3*49 = 3*7*7).
   - The third key is valid : $161 = 7 * 23$ (n is a product of two primes), thus $\Phi(n) = 6 * 22 = 132 = 2 * 2 * 3 * 11$, and $e = 35 = 5 * 7$ is prime with $\Phi(n)$ and thus is invertible.
   Then, we just have to compute d such that $e * d = 1 \mod \Phi(n)$ (using for example Euclide's Extended Algorithm), and we find d=83 (that we can check by verifying $35 * 83 = 2905 = 1 \mod 132$).

3. Given the prime number p=23, build a set of keys for an ElGamal encryption.

   To build an El-Gamal key, we need three things : a prime number (which is our p=23), a generator, and a secret.
   We need to find a generator of $\mathbb{Z}_p^*$: 2, 3, 4 are not generators, but 5 is : the consecutive powers of 5 give all the different values in $\mathbb{Z}_p^*$ (i.e. all values from 1 to 22). So we have our generator $\alpha = 5$.
   And finally, to build a key, we just need to choose a secret, let's say $a = 7$, and compute $\alpha^a \mod p = 5^7 \mod 23 = 17$.
   Now we have our public key K_{Pub} : $(p = 23, \alpha = 5, \alpha^a \mod p = 17)$, and our private key K_{Priv} : $(a = 7)$

Exercise 3 : Tryout for a MAC

We want to use AES in CBC mode as a MAC : the AES 128 bit key k is our secret MAC key, and to create a hash, we do an XOR of all the cipher blocks from the AES encryption. We consider IV=0.

That means we have one entry, which is a message m, one key k, and the output is a 128 bit hash (that is created by XORing all ciphers from AES) :

$$AES - MAC(m) = AES - MAC(m_1||m_2...||m_n) = c_1 \oplus c_2 \oplus ... \oplus c_n.$$

Where the m_i are the blocks of the message, and the c_i are the resulting ciphers from the block cipher encryption (reminder, we use CBC mode).

Note that the ciphers are not sent : we only send the message and the resulting MAC (thus the ciphers are not known, they can only be computed by the sender and the receiver as they both have the key)

1. If we suppose this system has second pre-image resistance and is secure, what computing power should we need to find a second pre-image (in other terms, how many times should we expect to try before finding a second pre-image) ?

   If we suppose this system has a second pre-image resistance, since we have 128 bits, we should expect to have a second pre-image in 2^{127} calculations (we have a 50% chance to have found a second pre-image at that point).

2. If we suppose this system has collision resistance and is secure, what computing power should we need to find a collision (in other terms, how many times should we expect to try before finding a second pre-image) ?

   If we suppose this system has collision resistance, since we have 128 bits, we should expect to have a collision in 2^{64} calculations (as by the brithday paradox, we need approximately the square root of the total number of possibilities).

3. Now, let us suppose we change the key and use a 256 bit key instead. How would that change the two previous results ? Why ?

   That would not change a thing : we still have only 128 bits for the message, and the fact that the key is longer has no impact here (aside from a slightly longer run time for each MAC calculation).

4. You have access to one pair (message, MAC). Can you use this to falsify the MAC for another message (i.e. build another pair (message,MAC) without the key ?

   In the general case, no, we can't falsify another pair (message, MAC) just from this. However, if the message is short enough to be only one bloc, then we can, because we also know $MAC = c_1$, and then we fall in the same case as in the following question.

5. Now suppose you have access to one pair (message, ciphertext). Can you falsify the MAC for another message ?

   In this case, we can falsify a MAC : for message m, let's build the following message : $m' = m \parallel m_{n+1} \parallel m_{n+2}$
   with $m_{n+1} = m_{n+2} = (c_n \oplus m_n)$ (where m_n and c_n are respectively the last bloc of the initial message and the last cipher of the initial cipher).
   That means this new message has two more blocs, and given that CBC

mode chains the previous cipher before encryption, we have the following blocks for our new cipher c' :
$c'_i = c_i \quad \forall\, i \leq n$
$c'_{n+1} = AES(c'_n \oplus m_{n+1}) = AES(c_n \oplus (c_n \oplus m_n)) = AES(m_n) = c_n$
$c'_{n+2} = AES(c'_{n+1} \oplus m_{n+2}) = AES(c_n \oplus (c_n \oplus m_n)) = AES(m_n) = c_n$
And thus, the new MAC' is equal to $MAC \oplus c_n \oplus c_n = MAC$, so the new message m' has the same MAC as the previous message m. So we just found a new valid pair (m', MAC) without the key.

Exercise 4 : Trying signature schemes with RSA

Let (d,n) be the RSA private key of A, and (e,n) the corresponding public key. A uses these keys to sign messages, by computing $y = m^d \mod n$, and sending (m,y) to B.

To validate the signature, B computes $x = y^e \mod n$, and accepts if and only if $x = m$.

1. Find a message easy to falsify when you possess only the public key.

   We can choose any number k, and then build a message $m = k^e \mod n$. This way, the signature of m is $s(m) = m^d \mod n = (k^e)^d \mod n = k$, and thus we can falsify pairs (m,k) very easily this way, by choosing the signature k, and then computing a message m from this signature.

2. B chooses a number x, and computes $x^{-1} \mod n$. Then he asks A to sign $m = x^e k \mod n$. What will B be able to obtain, without A noticing ? What do we call this process ?

   B obtains the message signed, $s(m) = m^d \mod n = (x^e)^d k^d \mod n = xk^d \mod n$. Then, by multiplying by the previously computed x^{-1}, B obtains $k^d \mod n$, which is the signature of A over message k, even though A never saw that message.
   This is what we call blind signatures.

3. Show that you can falsify the signature of a given message m (m different from 0, 1, p and q) with only two chosen signatures, i.e. you can choose two messages, obtain the signatures, and then falsify the signature for m.

   To falsify message m, let's choose m_1 and m_2 such that $m_1 m_2 \mod n = m$. If we obtain these two signatures, then we can falsify the signature for message m :
   $s(m) = m^d \mod n = (m_1 m_2)^d \mod n = m_1^d m_2^d \mod n = s(m_1)s(m_2) \mod n$.
   In short, the signature of the product is the product of the signatures.

4. Alice decides to use the previous signature scheme, and she decides it's easier to use the same pair of keys to sign and to encrypt/decrypt messages.
   Why is that a bad idea, and what does Alice give access to when she signs messages ?

That's a bad idea, because each time Alice signs a message, she gives the decryption of a cipher : a pair $(m, s(m)) = (m, m^d \mod n)$. That means by asking Alice to sign documents, people may in fact be asking her to decrypt some ciphers, and each time Alice signs something, she gives a pair (cipher, message).

5. Now, let us try another signature scheme : this time, the signature is $s = h(m)^d \mod n$, where $h(m) = SHA - 256(m)$, and we send the pair (m, s) to Bob. Do we still have all the previous problems ?

   With this new signature scheme, the previous problems are no more : we don't have the mutiplicative property of RSA anymore, thanks to the addition of SHA-256.