Homework 01

Md Moniruzzaman Monir, # 50291708

September 13, 2018

Problem 01

Solution:

A desktop publishing system is being used to produce documents for various organization.

a) The system will have to assure confidentiality if it is being used to publish :

- Bank account details of the clients of a bank
- Contracts and commercial documents of an organization
- Extremely sensitive investigation information of a law enforcement organization etc.

b) Data integrity is the maintenance of, and the assurance of the accuracy & consistency of, data over it's entire life cycle. The system will have to assure integrity if it is used to publish :

- Allergic information of patients of a hospital
- Laws and regulations of an organization etc.

c) The system will have to assure availability if it is used to publish :

- A daily newspaper
- Important transaction details in real time etc.

Problem 02

Solution:

a) There is a big security flaw in the given code segment. The intention of the code segment is to provide access to a particular resource. If **IsAccessAllowed()** returns **ERROR_ACCESS_DENIED**, then the user will not get permission to the requested resource. Otherwise the user will get access. But **IsAccessAllowed()** function can return many other errors like **OUT_OF_MEMORY**. In this case the user will get access to the requested resource. So if **IsAccessAllowed()** fails to execute due to memory issue or any other system issue then the user will get access which is a dangerous security flaw.

b)

```
DWORD dwRet = IsAccessAllowed(...);
if (dwRet == NO_ERROR) {
    // Secure check OK.
}
else {
    // Security check failed.
    // Inform user that access is denied.
}
```

Now, if the call to **IsAccessAllowed()** fails for any reason, the user is denied access to the requested resource.

Problem 03

Solution:

a) Ciphertext, as a function of the input and key blocks as well as the constants δ_i , is expressed below:

$$L_2 = L_0 \boxplus (((R_0 << 4) \boxplus K_0) \oplus ((R_0 >> 5) \boxplus K_1) \oplus (R_0 + \delta_1))$$
$$R_2 = R_0 \boxplus (((L_2 << 4) \boxplus K_2) \oplus ((L_2 >> 5) \boxplus K_3) \oplus (L_2 + \delta_2))$$

b) The constants δ_i are publicly known. I don't have the knowledge of the key but I can mount the chosen plaintext attack. Given this ability, I can't learn any information about the key using the 2-round version of **TEA**. I know the value of (L_2, L_0, R_0) and δ_1 in the first equation of 3(a). The unknowns are (k_0, k_1). I have no more equations containing these unknown and also the operations $\boxplus$ and $\oplus$ are not **associative** or **distributive** with one another, so it is not possible to solve this equation for (k_0, k_1). Similarly, I also can't get the value of (k_2, k_3) from the 2nd equation of 3(a).

c) The answer would not change if the 4-round version of TEA is used. Because the operations $\boxplus$ and $\oplus$ are not **associative** or **distributive** with one another. So though I had two equations containing two unknowns if the 4-round version is used, I can't get the value of the unknowns (keys).

Problem 04

Solution:

a) **AES-NI** are a set of six new instructions introduced by **Intel** when they introduced the new 2010 Intel Core processor family code named **Westmere**. AES-NI stands for Advanced Encryption Standard - New Instructions. These instructions implement hardware accelerated versions of certain compute intensive steps used in the AES (RijnDael) algorithm.
We can use hardware accelerated AES in python programming language by using the **PyCryptodome** package. PyCryptodome is a self-contained Python package of low-level cryptographic primitives. PyCryptodome is a fork of PyCrypto and it brings some enhancements (e.g. Accelerated AES on Intel platforms via AES-NI) with respect to the last official version of PyCrypto (2.6.1). This package has a module named AES which uses AES-NI instructions by default.

```
from Crypto.Cipher import AES
from Crypto import Random
plaintext = Random.new().read(AES.block_size)
print(plaintext)
key = Random.get_random_bytes(16)
obj = AES.new(key, AES.MODE_ECB)
ciphertext = obj.encrypt(plaintext)
print(ciphertext)
```

b) Hardware accelerated AES is not available in all programming languages.

Explanation:
Modern microprocessors can execute SIMD (Single Instruction, Multiple Data) instructions, but these instructions are part of different extended instruction sets. Because the need for greater computing performance continues to grow across industry segments, most CPU manufacturers have incorporated extended instruction sets in their new CPU models. For example, the most advanced Intel CPUs added two new SIMD instruction sets: AES-NI (Advanced Encryption Standard New Instructions) and AVX (Advanced Vector extensions).
However, if we use JavaScript or PHP code, there is no way to take advantage of the available SIMD instructions in the underlying hardware. With JavaScript, the engine provided by the Web browser might decide to use SIMD instructions under certain circumstances, but we cannot suggest the usage of even the simplest SIMD instructions via code. Other modern and popular high-level programming languages such

as C#, F#, Ruby and Scala don't provide direct support for calling SIMD instructions.

c) All the resources that I find useful in working on this problem are listed below :

- https://en.wikipedia.org/wiki/AES_instruction_set
- https://software.intel.com/en-us/blogs/2012/01/11/aes-ni-in-laymens-terms
- https://pypi.org/project/pycrypto/
- https://www.dlitz.net/software/pycrypto/api/current/Crypto.Cipher.AES-module.html
- https://github.com/dlitz/pycrypto/tree/master/lib/Crypto
- https://github.com/dlitz/pycrypto/blob/master/lib/Crypto/Cipher/AES.py
- https://blog.sqreen.io/stop-using-pycrypto-use-pycryptodome/
- https://pypi.org/project/pycryptodome/
- https://www.pycryptodome.org/en/latest/src/introduction.html
- https://pycryptodome.readthedocs.io/en/latest/src/cipher/aes.html
- https://github.com/Legrandin/pycryptodome/blob/master/lib/Crypto/Cipher/AES.py
- http://www.drdobbs.com/tools/high-level-programming-languages-should/230600043
- https://stackoverflow.com/questions/23058309/aes-ni-intrinsics-enabled-by-default

Problem 05

Solution:

a) An alernative implementation in **Java** using the package **javax.crypto**.

```java
import java.security.Key;
import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;

//Class
public class AES_ECB {

    private static final String ALGO = "AES/ECB/PKCS5PADDING";
    private byte[] keyValue;

    //Constructor
    public AES_ECB(String key){
        keyValue = key.getBytes();
    }

   // Key Generation
   private Key generateKey() throws Exception{
       Key k = new SecretKeySpec(keyValue,"AES");
       return k;
   }

  // Encryption
   public String encrypt(String data) throws Exception{

        Key k = generateKey();
        Cipher c = Cipher.getInstance(ALGO);
        c.init(Cipher.ENCRYPT_MODE,k);
        byte[] encVal = c.doFinal(data.getBytes());
        return Base64.getEncoder().encodeToString(encVal);
    }
```

```
32
33      // Main method
34        public static void main(String[] args){
35
36         try{
37
38             AES_ECB aes = new AES_ECB("This is a key123");
39             String encryptedString = aes.encrypt("The answer is no");
40             System.out.println(encryptedString);
41
42           }catch (Exception e){
43
44              System.out.println("Error while encrypting: " + e.toString());
45           }
46       }
47   }
```

b) Comparing the speed of the implementations in 4(a) and 5(a). I run both the program to encrypt 1000 1-block messages. Total time for 4(a) is 208.7376 ms and 5(a) is 281.34 ms. The difference in total time is very little because we are encrypting only 1000 1-block messages. If we increase the block size then 4(a) will perform slightly better.