UCS1505-Introduction to Cryptographic Techniques
Message Authentication Codes

Case Study-3

- Brute force attack on key
- Given, key size = k bits ⇒ Number of keys = 2^k
- Assume, the adversary knows the message and tag
- Aim: To find how many attempts will it take for the adversary to find the key?

Hypothesis:

We know that each message and key combination should have a unique tag. Let mlen, klen, tlen be lengths of message, key and tag respectively. The number of messages, keys and tags possible are,

Total messages possible=2^{mlen} | Total keys possible=2^{klen} | Total tags available=2^{tlen}

⇒ Number of (m,k) combinations = $2^{mlen+klen}$

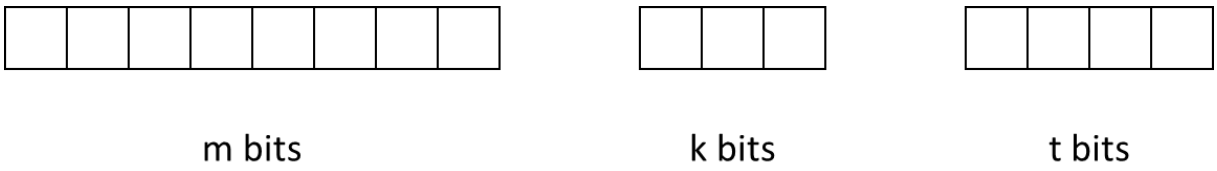
Dividing the set of combinations equally for the available tags

⇒ One tag for $2^{mlen+klen} / 2^{tlen}$ combinations

Tag selection:

To select a tag for a message and key combination, the following technique is used,

- Form a mlen+klen bit number by appending the key to the message.
- Let mk be the decimal equivalent of the number.
- Perform r=mk % 2^{tlen} operation. The result of this operation will be in the range 0 to 2^{tlen}-1.
- Let the r be the tag for the given combination.



Attacker's approach:

Assuming the attacker knows the message and tag, a brute-force attack on key is performed. The number of keys possible given the length of key is 2^{klen}. Each key is passed to the Vrfy() function until it returns true.

Program Code:

```
import random
keys=[]

#Key-Generation Algorithm to generate key
def Gen(klen):                                  #Takes in key length as input
    global keys                                 #If key length=k, 2**k keys are possible
    for i in range(0,2**klen):                  #For every number from 0 to (2**klen)-1
        key=str(bin(i).replace("0b",""))        #Binary equivalent is calculated
        key='0'*(klen-len(key))+key             #Zeroes added to maintain key length
        keys.append(key)                        #Appended to the key list
    k=random.choice(keys)                       #Random key selected from the key list
    return k                                    #Selected key is returned

#MAC Algorithm to generate valid tag
def Mac(k,m,tlen):                              #Takes in k, m & tag length as input
    mk=m+k                                      #message and key concatenated
    mk=int(mk,2)                                #Decimal equivalent is calculated
    t=mk%(2**tlen)                              #Calculation of tag
    t=str(bin(t).replace("0b",""))              #Binary equivalent is calculated
    t='0'*(tlen-len(t))+t                       #Zeroes added to maintain tag length
    return t                                    #Generated tag is returned

#Verify algorithm to check validity of tag
def Vrfy(key,msg,tag):                          #Takes in key, message and tag as input
    global t
    if Mac(key,msg,len(tag))==t:                #Checks if (k,m) generate valid tag
        b=1                                     #Sets b bit as 1 if satisfied
    else:
        b=0                                     #Sets b bit as 0 otherwise
    return b                                    #Returns the value of bit 'b'

#Brute-force attack algorithm
def Attack(m,t):
    global keys
    attempts=0                                  #attempts counter set to 0
    for key in keys:                            #For each key in key list
        attempts+=1                             #attempts incremented for each iteration
        print("Attempt:",attempts,end="")       #Displaying the attempt number
        print("\t     Key:",key,end="")         #Displaying the current key
        if (Vrfy(key,m,t)==1):                  #Performing verification
            k=key                               #k set as key if verified
            print("\t\tSuccess")                #Printing success message
            break
        print("\t\tFailed")                    #Printing failure message
    print("\n")
    print("Total attempts:",attempts,end="")    #Displaying the total number of attempts
    print("\t|    \tKey found:",k)             #Displaying the cracked key

#Sender's end
def Sender():
    global m,k,t,klen,tlen,keys
    keys=[]
    f="-"*50
    s=" "*18
    print()
```

```
    print(f,"\n")
    print(s,"Sender's side",s)
    m=input("\nEnter the message:\t")           #Getting input message from the sender
    klen=int(input("Enter length of key:\t"))   #Getting the key length
    k=Gen(klen)                                 #Passing klen to the Gen() function
    tlen=int(input("Enter length of tag:\t"))   #Getting the tag length
    t=Mac(k,m,tlen)                             #Passing tlen to the Mac() function
    print()
    print("Key:",k,"\t\t|\t\tTag:",t)           #Displaying the key and tag

#Attacker's end
def Attacker():
    global m,t                                  #Attacker knows the message and the tag
    f="-"*50
    s=" "*17
    print()
    print(f,"\n")
    print(s,"Attacker's side",s)
    print()
    Attack(m,t)                                 #Attacker's brute-force algorithm
    print()
    print(f)

Sender()
Attacker()
```

Sample output:

```
--------------------------------------------------

                  Sender's side

Enter the message:      10011011
Enter length of key:    3
Enter length of tag:    4

Key: 101                |               Tag: 1101

--------------------------------------------------
```

```
--------------------------------------------------

                  Attacker's side

Attempt: 1          Key: 000          Failed
Attempt: 2          Key: 001          Failed
Attempt: 3          Key: 010          Failed
Attempt: 4          Key: 011          Failed
Attempt: 5          Key: 100          Failed
Attempt: 6          Key: 101          Success

Total attempts: 6       |       Key found: 101

--------------------------------------------------
```

Inference:

The attacker succeeds after n+1 number of attempts, where n is the decimal equivalent of the original key, assuming the brute-force attack is done in the ascending order of the keys. In the best-case scenario, the adversary succeeds on the first attempt. Although, in the worst-case, they could take 2^k attempts, where k is the length of the key.