CS4355/6355: Cryptanalysis and DB Security
Instructor: Kalikinkar Mandal
Faculty of Computer Science
University of New Brunswick

Student Name: ______________________________ Matriculation Number: ____________________

The marking for each task is shown in [ ], and [100] constitutes the full mark.
You must implement the tasks on your own. You are NOT allowed to use any code or part of code from Internet and use any library APIs that directly implement these tasks as a whole.

A1. [**50**] This question is on implementing forging a message authentication code (MAC) value in the unforgeability under chosen message (UF-CMA) attack model. In the UF-CMA attack model, an adversary (Adv) can choose a number of messages of her own choice and can query to the tag generation or MAC oracle to obtain corresponding tags. Figure 1 shows an attack setting where the adversary wants to violate the integrity of a message by forging the MAC in the UF-CMA attack model. Suppose Alice and Bob have a shared MAC key, denoted by K. Alice computes a tag for message M as $tag = \mathsf{MAC}(K, M \| ID_A \| ID_B)$, where ID_A and ID_B are the identities of Alice and Bob, respectively and the tag length is 32 bits. The attack works as follows:

- Alice computes the tag $tag = \mathsf{MAC}(K, M \| ID_A \| ID_B)$ for the message M using the MAC algorithm and sends $ID_A, M,$ and tag to Bob.
- Adv captures $ID_A, M,$ and tag, applies the MAC forging attack on $ID_A, M,$ and tag in the UF-CMA attack model, and then forwards forged message M' along with $ID_A,$ and tag to Bob.
- After receiving $ID_A, M',$ and tag, Bob verifies whether the computed tag, denoted by $tag_1 = \mathsf{MAC}(K, M' \| ID_A \| ID_B)$ is the same as the received tag, and accepts the message M' if the MAC/tag verification is successful.

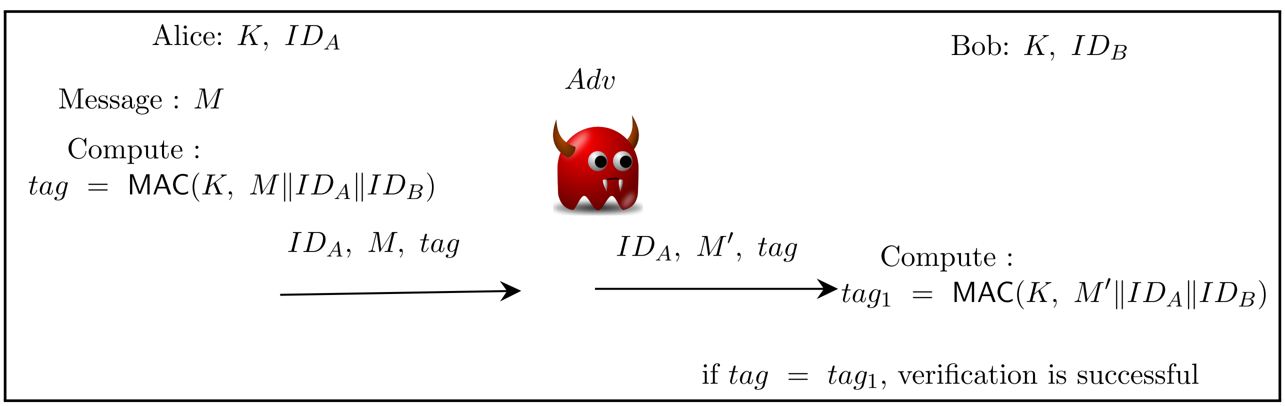

Figure 1: An MAC forgery attack on a simple message integrity check protocol

Use the following parameters and construction of MAC in your implementation:

- Shared MAC key K = `0x00112233445566778899AABBCCDDEEFF` (in hex)
- Message M = "You know my methods, Bob."

- Use your UNB student ID as an ID of Alice, i.e., ID_A and $ID_B = 0070070$
- $\mathsf{MAC}(K, M\|ID_A\|ID_B) = \lfloor \mathsf{SHA256}(K\|M\|ID_A\|ID_B)\rfloor_{32}$ which means you take only first 32 bits of the output of $\mathsf{SHA256}$ as a MAC or tag value.

In the MAC forging attack, forging a MAC is equivalent to exhaustively find another message M' while keeping K, ID_A and ID_B fixed. Please implement this functionality in your chosen programming language and find a different message M' so that $\mathsf{MAC}(K, M\|ID_A\|ID_B) = \mathsf{MAC}(K, M'\|ID_A\|ID_B)$. You can use a $\mathsf{SHA256}$ implementation in your chosen programming language. (Note that M' does not need to be a meaningful message. It could be any number or any message.)

Sample I/O:

Original input and output:

M: _____

$ID_A =$ _____

MAC tag _____

Forged input and output:

M': _____

MAC tag_1 _____

Verification:

Verification result: _____

A2. [**50**] Please implement the key generation, signing and verification algorithms of the ElGamal signature scheme either in C using GMP, Java using BigInteger or Python using gmpy library for a 128-bit security level. In your computation, you treat the output of SHA256 as an integer. The large primes p, and g are provided in the parameters section (see the last page). For convenience, these three algorithms are described below. You are prohibited to use any ElGamal code available on Internet or other sources, and ElGamal APIs available in your programming language libraries.

$(vk, sk) \leftarrow \mathsf{KeyGen}(p, g)$	Signing $\sigma \leftarrow \mathsf{Sign}(sk, vk, m)$	Verify yes, no $\leftarrow \mathsf{Vrfy}(vk, m, \sigma)$
1. Select x is a random number in $[2, p-1]$	1. Choose a secret random number $k \in [2, p-2]$ and $\gcd(k, p-1) = 1$	1. Verify $1 \le r, s \le p-1$
2. Compute $y = g^x \mod p$	2. Compute $r = g^k \mod p$	2. Compute $u = y^r r^s \mod p$
3. Verification key $vk = (y, g, p)$	3. Compute $k' = k^{-1} = \frac{1}{k} \mod (p-1)$	3. Compute $h = \text{SHA256}(m)$
4. Signing key $sk = (x)$	4. Compute $s = k'(\text{SHA256}(m) - xr) \mod (p-1)$	4. Compute $v = g^h \mod p$
	5. Signature: $\sigma = (r, s)$ for m	5. Accept signature σ if and only if $u = v$

Sample I/O:

Signing:

ElGamal signing key $x =$ _____

ElGamal verification key $vk = (y, g, p) =$ _____

```
Signing:
Message to be signed m = _____
Signature σ = (r, s) = _____
----------------------------
Verification:
Printing u = _____
Printing h = _____
Printing v = _____
Verification result: _____
----------------------------
```

Sample I/O:

```
----------------------------
DH private key for Alice x: _____
DH private key for Alice y: _____
Keys K0, K1 derived by Bob: _____
Printing σB: _____
Printing tagB: _____
Tag and signature verification results by Alice:
Printing σA: _____
Printing tagA: _____
Keys K0, K1 derived by Alice: _____
Tag and signature verification results by Bob:
----------------------------
```

Resources for implementations. Below are some libraries in C, Python, Java that you can use for large number operations.

- The GMP library. `https://gmplib.org/` (for C)
- The gmpy2 library. `https://pypi.org/project/gmpy2/` (for Python)
- The BigInteger class in Java

Parameters

ElGamal signature parameters. This prime is taken from the IKE protocol specified in RFC 3526.

$p =$

5809605995369958062791915965639201402176612226902900533702900882779736177890990861472
0947744773395811473734101856463783280437298007504700982109244878669350591643715881680
4754094398164451663275506750162643455639819318662899007124866081936120511979369398543
3297036118232914410171876807536457391277857011849897410207519105333355801121109356897
4594262718454713979526759594407934930716283941227805101246184882326024646498768504588
6124578424092925842628769970531258450962541951346360515542801716571446536309402160929
0561084025893662561222573202082865797821865270991145082200656978177192827024538990239
9691755461907706456858934380117144304264093386763147435711545371420315730042764287014
3303638180170530865983075119035294602548205993130657100472736247968841557470259694645
7770284148435989129632853918392117997472632693078113129886487399347796982772784615865
232621289656944284216824611318709764535152507354116344703769998514148343807

$g = 2$