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**] Please implement the key generation, signing and verification algorithms of the digital signature algorithm (DSA) 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 a 256-bit integer. The large primes p, q and g are provided in the parameters section (in the last page). For convenience, these three algorithms are described below. You are prohibited to use any DSA code available on Internet or other sources, and DSA APIs available in your programming language libraries.

$(vk, sk) \leftarrow \mathsf{KeyGen}(p, q, g)$	Signing $\sigma \leftarrow \mathsf{Sig}(sk, vk, m)$	Verify yes, no $\leftarrow \mathsf{Vrfy}(vk, m, \sigma)$
1. Randomly an element $g \in \mathbb{Z}_p^*$ and compute $h = g^{\frac{p-1}{q}} \mod p$ with $h \neq 1$	1. Choose a secret random number $k \in [2, q-1]$	1. $w = s^{-1} \mod q$
	2. Compute $r = (h^k \mod p) \mod q$	2. $u_1 = w \times \text{SHA256}(m) \mod q$ $u_2 = r \times w \mod q$
2. Select x is a random number in $[2, q-1]$	3. Compute $k' = k^{-1} = \frac{1}{k} \mod q$	3. Compute $v = (h^{u_1} y^{u_2} \mod p) \mod q$
3. Compute $y = h^x \mod p$	4. Compute $s = k'(\text{SHA256}(m) + xr) \mod q$	4. Accept signature σ if and only if $v = r$
4. Verification key $vk = (y, h, p, q)$	5. Signature: $\sigma = (r, s)$ for m	
5. Signing key $sk = (x)$		

Sample I/O:

Signing:

DSA signing key $x =$ _____

DSA verification key $vk = (y, h, p, q) =$ _____

Signing:

Message to be signed $m =$ _____

Signature $\sigma = (r, s) =$ _____

Verification:

Printing $w =$ _____

Printing $u_1 =$ _____

Printing $u_2 =$ _____

Printing $v =$ _____

Verification result: _____

A2. [**50**] Let us consider the following authenticated DH key exchange protocol to establish a secret shared key in the Internet Protocol Security (IPsec) standards. The protocol for two parties, namely Alice and Bob is described below and a high-level overview is shown in Figure 1. Use the following cyclic group of prime order q: $G = \mathbb{Z}_q^*$. Use the generator g of $\mathbb{Z}_q^*$ provided in the parameter section. Use the DSA implementation from **Task A1** for $\mathsf{KeyGen}()$, $\mathsf{Sig}()$ and $\mathsf{Verify}()$. You can generate a pair of signing-and-verification keys for Alice and Bob, denoted by (vk_A, sk_A) and (vk_B, sk_B) by calling $\mathsf{KeyGen}()$. Suppose T is the session identity of the protocol. You can choose T as a 32-bit number. Alice and Bob execute the protocol as follows:

Step 1. Alice: Alice chooses a random integer $x \in \mathbb{Z}_q^*$ and computes the DH public key $X = g^x \mod q$. Alice randomly generates a session ID T, and sends (T, X) to Bob.

Step 2. Bob: After receiving (T, X), Bob performs the following:

- Choose a random integer $y \in \mathbb{Z}_q^*$ and compute the DH public key $Y = g^y \mod q$
- Compute $Z = X^y \mod q = (g^x)^y = g^{xy} \mod q$ and the key $K = K_0 \| K_1 = \text{SHA256}(Z)$
- Compute $\sigma_B = (r_B, s_B) \leftarrow \mathsf{Sig}(sk_B, vk_B, T\|g^x\|g^y)$
- Compute a tag $tag_B = \mathsf{HMAC}(K_1, T\|ID_B)$
- Send $(T, Y, ID_B, tag_B, \sigma_B)$ to Alice

Step 3. Alice: After receiving $(T, Y, ID_B, tag_B, \sigma_B)$ from Bob, Alice performs the following:

- Compute $Z = Y^x \mod q = (g^y)^x = g^{xy} \mod q$ and the key $K = K_0 \| K_1 = \text{SHA256}(Z)$
- Check using K_1 that tag_B is the same as $tag' = \mathsf{HMAC}(K_1, T\|ID_B)$. If not, the tag verification fails.
- Perform the signature verification: $\mathsf{Verify}(vk_B, T\|g^x\|g^y) \rightarrow \{\text{yes}, \text{no}\}$. If the signature verification is successful, perform the following steps.
- Compute $\sigma_A = (r_A, s_A) \leftarrow \mathsf{Sig}(sk_A, vk_A, T\|g^y\|g^x)$
- Compute a tag $tag_A = \mathsf{HMAC}(K_1, T\|ID_A)$
- Send $(T, ID_A, tag_A, \sigma_A)$ to Bob.

Step 4. Bob: After receiving $(T, ID_A, tag_A, \sigma_A)$ from Alice, Bob performs the following verifications:

- Check using K_1 that tag_A is the same as $tag'' = \mathsf{HMAC}(K_1, T\|ID_A)$. If not, the tag verification fails.
- Perform the signature verification: $\mathsf{Verify}(vk_A, T\|g^y\|g^x) \rightarrow \{\text{yes}, \text{no}\}$.

Alice: $(vk_A, sk_A), ID_A$		**Bob:** $(vk_B, sk_B), ID_B$
Randomly choose T, x $X = g^x$	$\xrightarrow{T, X = g^x}$	Randomly choose y $Y = g^y$
$K = K_0\|K_1 = \text{SHA256}(g^{xy})$ Verify: tag_B, σ_B	$\xleftarrow{T, Y, ID_B, tag_B, \sigma_B}$	$K = K_0\|K_1 = \text{SHA256}(g^{xy})$ Compute: σ_B, tag_B
Compute: tag_A, σ_A	$\xrightarrow{T, ID_A, tag_A, \sigma_A}$	Verify: tag_A, σ_A

Figure 1: Internet Key Exchange Protocol

Please implement the above DH-based key exchange protocol (in Figure 1), using the parameters provided below, in the same code. You don't need to implement the communication protocol. In your implementation, all random numbers should be generated randomly (not hardcoded). You can use the Hash-based MAC algorithm HMAC as an API for the MAC computation. For simplicity, you can use the following definition $\mathsf{HMAC}(K_1, T\|ID_B) = \text{SHA256}(K_1\|T\|ID_B)$ if you wish. You consider $K = K_0\|K_1 = \text{SHA256}(g^{xy})$ as two keys K_0, K_1 where each K_i is of 128 bits.

Sample I/O:

```
----------------------------
DH private key for Alice x:  _____
DH private key for Alice y:  _____
Keys K_0, K_1 derived by Bob:  _____
Printing σ_B:  _____
Printing tag_B:  _____
Tag and signature verification results by Alice:
Printing σ_A:  _____
Printing tag_A:  _____
Keys K_0, K_1 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

DSA Parameters encryption parameters. New parameters for p.

$p =$
5070234208798698468459654067278529449337082408530849845053556570173045087974531059406
9460940052367603038103747343106687981163754506284021184158903198888031001641800021787
4537609196268517043810095456243314686587312551099951866986023886163451187795712120890
9041897231730193382132789753969263374090652446190491006168745964228585505227527457608
9050579224477511686171168825003847462222895619169935317974865296291598100558751976216
4184699849371105070619794009719057814103883364589088168857584191253750474083886019853
0088450073392319470005103073365343446671494360584514351993390159215829580902051323582
7728686129856549511535000228593790299010401739984240789015389649972633253273119008010
9711111070285360935431163046132694380824689607888361399993901415701582084102347337800
0734526444094688807201863211977844219482269063546088317796507837840403530642300156054
617426093544172847945488788405708248152008981027191222735088475202376066 3

$q =$ 637623513649726535646416995292055104892632668341827716175636313632779328542 27

$g = 2$