Digital signature algorithms

A digital signature allows you to authenticate the sender of information. The signature is associated with both the author and the document itself using cryptographic methods, and cannot be forged for another document by copying.

For generation of a key pair (private and public) in the Electronic Digital Signature (EDS) or Qualified Electronic Signature (QES) algorithms, using different mathematical schemes, based on the use of one-way functions. Traditional schemes are divided into two groups. This division is based on well-known complex computational problems: factorization of large integers and discrete logarithm.

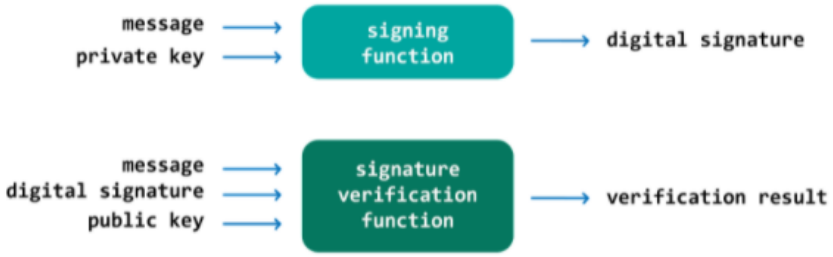

Figure 1.2 – Digital signature scheme

2. Workflow

You can use **any** programming language that is convenient for you to do the practical task. The task is to implement **one** of the proposed algorithms (you can choose to implement an algorithm that is not described in this tutorial).

Options for doing the lab:

1	Digital signature algorithm	RSA or ECDSA or Schnorr signature or RingSig

Note that the choice of a level for performing the task does not imply the need to implement all of these algorithms. The implementation of algorithms more than 1 is encouraged, but not required.

3. RSA Algorithm

The RSA algorithm is an asymmetric encryption algorithm. Today it is very often used to organize a secure channel between users (forming a shared secret and encrypting it using RSA).

The sequence of RSA algorithm steps (key generation):

1. Choose two large prime numbers (p and q);
2. Calculated: $n = p \cdot q$, $m = (p - 1) \cdot (q - 1)$;
3. A random number d is chosen that is coprime with m;
4. A number e is determined for which the expression $(e \cdot d) \bmod (m) = 1$ is true;
5. The numbers *e* and *n* are the public key and the numbers *d* and *n* are the private key.

The public key encrypts the message, and the private key decrypts it. The private key pair is kept secret.

Encryption and decryption of information using the RSA algorithm:

1. The original text is divided into blocks and each of them can be represented as a number $M(i)$;
2. Text encryption is performed as follows: $C(i) = (M(i)e) \bmod n$;
3. Text decryption is performed as follows: $M(i) = (C(i)^d) \bmod n$.

Pseudo-code:

```
KeyGen(){
      P and Q <= primary big numbers.
      n = P*Q.
      e <= integer value, not be a factor of n, 1<e<Φ(n), Φ(n) = (P-
1)(Q-1)
      d = (k*Φ(n) + 1) / e , for some integer k

      PrivateKey (d, n)
      PublicKey (e, n)
}

Encrypt(message, PublicKey){
      return powmod(message, e, n)
```

```
}

Decrypt(ciphertext, PrivateKey){
      return powmod(ciphertext, d, n)
}
```

You can find the text of the standard and the test vectors at the following link: https://datatracker.ietf.org/doc/html/rfc3447

4. ECDSA algorithm

The Elliptic Curve Digital Signature Algorithm (ECDSA) offers a variant of the Digital Signature Algorithm (DSA) that uses elliptic curve cryptography.

```
Pseudo-code:

KeyGen(){
      d <= random ∈ [1, n - 1].
      Q = d x P.

      // additional verifications
      Checking that Q ≠ 0.
      Checking that xQ and yQ are properly represented elements of Fq.
      Checking that Q is on the elliptic curve defined by a and b.
      Checking that nQ = Q.

If any of these checks fail the public key Q is invalid, otherwise Q is
valid.

Sign(m, d){
      k <= random ∈ [1, n - 1].
      k x P = (x1, y1)
      r = x1 mod n.

      if r = 0 then go to start.

      s = pow(k, -1)(H(m) + d • r) mod n.

      if s = 0 go to start.
      Signature (r, s).
```

```
}

Verify(r, s, Q){
      Verify that r and s are integers in the interval [1,n-1].

      c = s-1 mod n and H(m)
      u1 = H(m) • c mod n
      u2 = r • c mod n
      u1 x P + u2 x Q = (x0, y0)
      v = x0 mod n
      if v = r -> return true

}
```

You can find the text of the standard and the test vectors at the following link: https://datatracker.ietf.org/doc/html/rfc6979

5. **Schnorr Signatures**

Schnorr signatures are a type of digital signatures that support the aggregation of public keys and signature values, so as to make a multi-signature compact and anonymous (a single signature is no different from a signature generated by a potentially infinite number of participants).

Pseudo-code:

```
KeyGen(){
      d <= random ∈ [1, n - 1].
      Q = d x P.

      // additional verifications
      Checking that Q ≠ 0.
      Checking that xQ and yQ are properly represented elements of Fq.
      Checking that Q is on the elliptic curve defined by a and b.
      Checking that nQ = Q.

If any of these checks fail the public key Q is invalid, otherwise Q is
valid.

Sign(m, d){
      r <= random
      R := scalarMult(P, *r)

      r = r + H(X,R,m)*x

      signature.R = R
      signature.S = *r
      return
}

Verify(r, s, Q){
      sP ?= R + H(X,R,m)Q
}
```

Алгоритм мультиподписи Шнорра имеет следующий вид:

Pseudo-code:

```
Sign(){
      L = H(X1,X2,…)
      X = sum(H(L,Xi)*Xi)
```

```
      ri - rand
      Ri = ri*G
      R = sum(Ri)

      Each signer computes si = ri + H(X,R,m)*H(L,Xi)*xi

      s is the sum of the si values

Verify(){
      sG ?= R + H(X,R,m)*X
}
```

https://tlu.tarilabs.com/cryptography/introduction-schnorr-signatures

6. Ring traceable signatures

One of the CryptoNote standards describes a one-time ring signature algorithm. This approach allows the user to sign a transaction while maintaining anonymity, since the verifier can make sure that the signature was generated by one of the group members without being able to find out who exactly. To prevent a double-spend attack, it was decided to use a ring signature mechanism using a private key image.

```
Pseudo-code:

Signature input:
      M           - message
      A[0...n-1]  - public keys of ring participants,
      a[i]        - private signer key,
      A[i]        - public signer key.

Procedure generate_signature (M, A[1], A[2], ..., A[n], i, a[i]):
      I <- a[i]*H(A[i]) // Private key image
      c[j], r[j] [j=0..n-1, j!=i] <- random
      k <- random
      For j <- 0..n-1, j!=i
            4.1. X[j] <- c[j]*A[j]+r[j]*G
            4.2. Y[j] <- c[j]*I+r[j]*H(A[j])
      X[i] <- k*G
      Y[i] <- k*H(A[i])
      c[i] <- H(H(M) || X[0] || Y[0] || X[1] || Y[1] || ... || X[n-1] ||
Y[n-1]) - Sum[j=0..n-1, j!=i](c[j])
      r[i] <- k-a[i]*c[i]
```

```
Return (I, c[0] || r[0] || c[1] || r[1] || ... || c[n-1] || r[n-1])

Procedure verify_signature(M, A[0], A[1], ..., A[n-1], I, c[0], r[0], c[1],
r[1], ..., c[n-1], r[n-1]):
      For i <- 0..n-1
            1.1. X[i] <- c[i]*A[i]+r[i]*G
            1.2. Y[i] <- c[i]*I+r[i]*H(A[i])
      If H(H(M) || X[0] || Y[0] || X[1] || Y[1] || ... || X[n-1] || Y[n-1])
== Sum[i=0..n-1](c[i])
            Return "Correct"
      Else
            Return "Incorrect"
}
```

CryptoNote Standards:
https://cryptonote.org/standards/