Ciências ULisboa | Informática

Data Privacy and Security

Lab 3 – Secret Sharing & Threshold Cryptography

Objectives

In this lab we will be implementing some basic secret sharing schemes, including a $\boldsymbol{t} = \boldsymbol{n}$ Secret Sharing scheme and Shamir's Secret Sharing. For each implementation, you can test if the operations are working as intended (e.g., if you can share a secret value and then reconstruct it).

Exercises

1.1. Implement Shamir's Secret Sharing scheme. To implement this scheme, an incomplete implementation is given and you can add the missing functions. Some notes on the implementation:

- First, we need to decide on the public parameters to use, including the group order. For testing purposes, a small group could be used, however in our incomplete implementation, a well-known secure group was selected from: https://www.rfc-editor.org/rfc/rfc5114 (Section 2.3 where parameter **q** is the group order).
  - These parameters are secure and could be used in production software.

- A PRNG is needed for generating secure random numbers ranging from 0 to **q**;

- Function *share(…)* first needs to generate the random coefficients of the polynomial $(\boldsymbol{a_d}, \dots, \boldsymbol{a_1})$, given its degree $\boldsymbol{d}$;
  - The constant factor of the polynomial should be the secret, i.e., $\boldsymbol{a_0} = \boldsymbol{s}$

- Function *calculatePoint(…)* calculates and outputs a point from the polynomial (i.e., a share), given the polynomial coefficients and the point's id (its x value).
  - Points can be efficiently calculated using Horner's method: https://en.wikipedia.org/wiki/Horner%27s_method
  - All operations should be done module the group order (i.e., **mod q**);
  - Points can be incrementally selected from {1,…,N}, while point 0 should not be used since it encodes the secret (see function *share(…));*

- Function *combine(…)* implements Lagrange interpolation to reconstruct a secret from a set of **t** shares (https://en.wikipedia.org/wiki/Lagrange_polynomial).
    - Lagrange interpolation can be efficiently computed as follows, where the list of points on the polynomial is given as $\boldsymbol{k} = \boldsymbol{t}$ pairs of the form (x_i, y_i):

$$f(0) = \sum_{j=0}^{k-1} y_j \prod_{\substack{m=0 \\ m \neq j}}^{k-1} \frac{x_m}{x_m - x_j}$$

Additional Exercises

2. Implement a simple $\boldsymbol{t} = \boldsymbol{n}$ Secret Sharing scheme based on the addition operation. See the lecture slides for assistance.

3. Add verifiability to your implementation of Shamir's Secret Sharing by using Feldman's Commitments. For this, you will need:

    - To find a generator **g** and a group order **p** for the commitments. You can use small numbers for tests, or check section 2.3 of https://www.rfc-editor.org/rfc/rfc5114 again.
        - Remember that operations with commitments should be done module **p**.
    - A function that, given the random coefficients of a polynomial and generator g, generates its commitment;
    - A function that, given the commitment of a polynomial and a share, verifies if the share belongs to the polynomial.