Practical case

Context:

A company is developing a new authentication protocol. The company cryptographers have been investigating alternatives to password hashing and have found that Zero-Knowledge Proof (ZKP) is a viable alternative to hashing in an authentication schema. Moreover, the cryptographers have found a ZKP protocol suitable for this purpose. As part of an agile team, you have accepted the challenge of implementing the ZKP Protocol and a Proof-of-Concept application that utilizes the protocol to register and authenticate users.

The ZKP Protocol

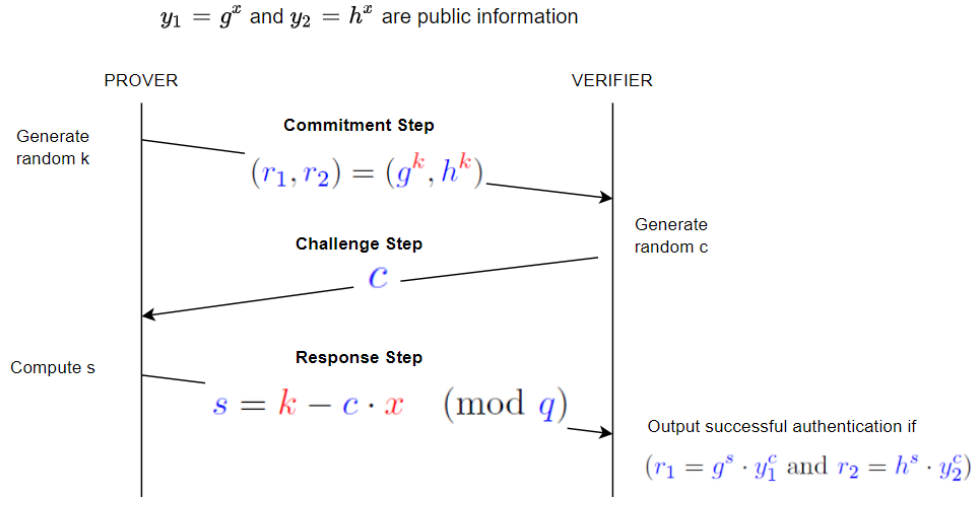
The ZKP protocol is described in the book "Cryptography: An Introduction (3rd Edition) Nigel Smart" page 377 section "3. Sigma Protocols" subsection "3.2. Chaum–Pedersen Protocol." We want to adapt this protocol to support 1-factor authentication, that is, the exact matching of a number (registration password) stored during registration and another number (login password) generated during the login process. We now describe the registration and the login processes.

Registration Process

The prover (client) has a secret password *x* (i.e. it is a number) and wishes to register it with the verifier (server). To do that, they calculate y1 and y2 using public g and h and the secret x and sends to the verifier y1, y2.

Login Process

The login process is done following the ZKP Protocol shown in the diagram. Been the Prover the authenticating party and the Verifier the server running the authentication check:



Primary ask

Design and write the code that implements the ZKP Protocol outlined above. Solution should be implemented as server and client using gRPC protocol according to the provided interface described in “protobuf” schema. The code should implement very simple server and client applications. We would like to see how you are thinking about and approach the problem, so a simple description of your approach and how you’d extend it or integrate would be helpful.

Bonus requirements

The following bullet points are all things that would be nice to see in a working implementation. None of them is mandatory, but if there are any of these items you can add to your submission it would be beneficial for us in determining the level of your submission.

- Unit tests, where appropriate
- Functional test of the ZKP Protocol
- A setup to run the Client and the Server.
- Use Rust as language for the implementation
- Performance and optimizations
- Coverage of test cases (not code coverage)
- Code soundness
- Code organization
- Code quality
- Well documented code
- Each instance runs in a separated docker container and have a docker compose to run the setup
- There is code to deploy the two containers in AWS. The client in one machine and the server in another machine
- Implement two flavor: One with exponentiations (as described in the book) and one using Elliptic Curve cryptography (see for example this ZKP implementation in Rust)
- Allow using “BigInt” numbers

Protobuf Definition

```
syntax = "proto3";
package zkp_auth;

message RegisterRequest {
  string user = 1;
  int64 y1 = 2;
  int64 y2 = 3;
}

message RegisterResponse {}

message AuthenticationChallengeRequest {
  string user = 1;
  int64 r1 = 2;
  int64 r2 = 3;
}

message AuthenticationChallengeResponse {
  string auth_id = 1;
  int64 c = 2;
}

message AuthenticationAnswerRequest {
  string auth_id = 1;
  int64 s = 2;
}

message AuthenticationAnswerResponse {
  string session_id = 1;
}

service Auth {
  rpc Register(RegisterRequest) returns (RegisterResponse) {}
  rpc CreateAuthenticationChallenge(AuthenticationChallengeRequest) returns
(AuthenticationChallengeResponse) {}
  rpc VerifyAuthentication(AuthenticationAnswerRequest) returns (AuthenticationAnswerResponse)
{}
}
```