DOCUMENTATION OF PUBLIC PROOF OF RETRIEVABILITY IN CHARM

MANUEL STEGMILLER, MAGNUS WEBER, EUGEN NABIEV

Abstract. In a proof-of-retrievability system, a server has to prove, that he has stored a specific file. This can be used for storage providers. The clients are interested in knowing that all of their data is stored. The most simple prove is to just send the whole file to the client. But this is neither fast nor efficient. Hovav Shacham and Brent Waters developed a scheme where both, client's query and server's answer, are extremely short. In this project, we implemented their public proof-of-retrievability(POR) scheme with the charm framework.

1. Public Proofe of Retrievability

In modern technologie and cloud computing data is often stored by a third party. Users are outsourcing the storage of their databases and data to optimize costs. The downside of third-party-storage systems is, that they may lose the data. The user whats a system to be sure that his data is continuous available and complete. The solution is storage auditing. This can be achieved with the POR-scheme developed by Shacham and Waters. This is not only important to the user, but also to the provider of the storage. If he can prove that the data is still there, he has a high trustness level and can outsmart his competitors.

To use the scheme, the user generates a challenge. This challenge can only be answered by the server if he holds the complete data. The user choses some random parts of the data together with random coefficients. The server then calculates for every given part of the file a signature. These signatures are multiplied up and send back to the user. The answer from the server therefore is very short. The user can now check if the signatures are correct by calculating a a verification equation. The challenge can be shortened by only sending a seed for a random oracle. With this seed, the server is able to generate the challenge on his own.

2. How to use it

First, the server has to generate his keypair. After this, both parties need to split the file in the specified parts. The one who starts first has to send the other one p, so that both parties have the same prime number. The user then generates a challenge and sends it to the server. Listing 1 shows how to use the code. The server now computes the prove to the challenge and sends it back to the user. The user can now verify if the prove fits to the challenge. To do this, he needs the public key of the server. This system is fast an has only short messages to transfer. The two parties must agree on the same value for u, wich is part of the public key, and the same group to use. This can be achieved by sending them to each other like p or set them fix while doing the storage-contract. To set this values the code has setter methods.

LISTING 1. How to Use Code

```
"""Code Server:"""
from PublicProofOfRetrievability import *
proof = VerifiableProofOfRetrievability() #initialize
keys = proof.keygen() #generate a keypair
proof.splitMessage(message) #split the message
#send prime number and public key to the user
#... wait for the challenge
response = proof.proove(splitm, challenge, x) #generate the
    response
#send the response back

"""Code User:"""
from PublicProofOfRetrievability import *
proof = VerifiableProofOfRetrievability() #initialize
proof.setP(prime) #set the same prime as the server
proof.splitMessage(message) #split the message
challenge = proof.generateChallenge(splitm, 4) #generate a
    challenge
#send challenge to the server
#... wait for response
proof.verify(response, challenge, v) #proove?
```

3. Lessons learned

The biggest problem was to understand how charm works. It has a wide range of functions and classes. The documentation is not very good. There are no clues how to use the functions correctly. So maybe next time we would choose an additional partner who has some experience with the framework. The examples given with the framework weren't very helpfull, because they all implementes something complete differnt than we.

Also, next time we would start making a flowchart on paper before starting to code. It was hard to imagine how the code should work in the end, without knowing in wich order, and from whom the functions will be called. If you understand everything on paper, the implementation works like a charm (haha).

4. Files

Under the Folder charm/charm/schemes add a folder "por" for proof of retrievability. There you add the file. If you have installed charm the right way, you can now use the library in your code. from PublicProofOfRetrievability import *