Proactive Dynamic Secret Sharing: Implementation, Benchmarking, Applications to Distributed Password Authenticated Symmetric Key Encryption

Monday 7th November, 2022 - 18:32

Georgi Tyufekchiev
University of Luxembourg
Email: georgi.tyufekchiev.001@student.uni.lu

This report has been produced under the supervision of:
Qingju Wang, Marius Lombard-Platet
University of Luxembourg
Email: qingju.wang@uni.lu
Email: marius.lombard-platet@uni.lu

***Abstract*—Data breaches are a serious threat to both the businesses and their customers. Compromised passwords can lead to irreparable damage in the life of anyone. For this reason, strong password authentication protocols are needed to protect the personal information of the user. The aim of this paper is to look at how dynamic secret sharing schemes can help the development of such protocols. We will provide a proof-of-concept implementation of such a scheme, which was developed for retrieving secrets from the blockchain.**

1. Introduction

Over the past few decades, technology has evolved significantly - from the invention of the World Wide Web to the emergence of cloud based services. However, the more complex the systems become, the more they are prone to security vulnerabilities. Hackers are getting creative in the ways they attack and exfiltrate data. With information being one of the most valuable assets, it is crucial that we protect our personal data.

The majority of the population is using social media applications, peer-to-peer communication or playing massive multiplayer online games. We heavily rely on companies to deploy strong security mechanisms to protect their customers. Still, every year we can witness data breaches not only in small businesses but major ones as well. The users themselves can become a victim of phishing or social engineering attacks, which results in stolen credentials and identity theft.

One of the reasons for such problems is the way we create and use passwords. It is cumbersome to make and remember a new long and complex password for every application. For this reason, many create a simple one (and derivations of it) and use it in more than one place. However, this can make the job of our adversaries much simpler. So how can we protect ourselves?

One solution is for the users to use and store strong encryption keys. Even so, not everyone has the technical knowledge for this, and if someones has it, key management is a difficult task. The solution to this problem is using distributed password authentication protocols. Such protocols store the encryption key on multiple servers, releasing the end-user from the responsibility of managing keys as well as protecting them from different attacks. The aim of this Bachelor Semester Project is to look at proactive dynamic secret sharing, which can aid in the development of such protocols.

2. What is DPSS and its application?

Dynamic proactive secret sharing is a method to update distributed keys after certain periods of time, such that the attacker does not have the time to compromise the key(s). It is a useful technique for when the attacker can dynamically corrupt parties (servers), providing us with optimal-resilient systems [1]. Such schemes also provide a solution to the "Byzantines Generals Problems"[5]. In the problem, Byzantine generals communicate only with messengers, which can be corrupt and provide false information to other parties with the goal of confusing and sabotaging them.

Another usage for proactive secret sharing is in combination with the *witness encryption* [2] concept. The authors, who came up with the idea, want to find a solution to the following problem: "*Can we encrypt a message so that it can only be opened by a recipient who knows a witness to an NP relation?*". In this case, we only care if the recipient knows the solution to some NP-complete problem such as the battleship puzzle or the travelling salesman problem. Our main focus is the paper on how to store and retrieve secrets from the blockchain [3], which combines the concepts of DPSS and extractable witness encryption.

2.1. Domains

2.1.1. Scientific . The scientific domain for the project is cryptographic protocols. We will look at several cryptographic methods, which are necessary to achieve dynamic secret sharing.

2.1.2. Technical. A proof-of-concept implementation of DPSS will be shown, which covers the techniques presented in the scientific part. A few parts of the source code will be explained.

2.2. Preliminaries

In this section, we will provide a high-level overview of the scientific concepts and technical tools used in this paper.

2.2.1. Shamir Secret Sharing. The Shamir Secret Sharing scheme [6] (SSS) is a (k,n) threshold scheme, used to split a secret into n shares, where any k-subset of n can recover the secret. It provides us with homomorphic encryption, meaning we can do addition and scalar multiplication on the shares.

2.2.2. Polynomial Commitments. Polynomial commitments [4] can be found in zero knowledge proofs. They are used to store a large amount of data into elliptic curve points. The goal is to prove to a verifier that some computations have taken place, without him redoing all the computations. It finds application in blockchain technologies, where storing large amounts of data is expensive.

2.2.3. Elliptic Curve Cryptography. Elliptic curves are an algebraic structure over finite fields, which are used in public key cryptography. We will focus on paring friendly curves, which are used in the polynomial commitments.

2.2.4. Technical Tools.

2.2.5. Python. The Python programming language was created by Guido van Rossum in 1991. It is a high-level, dynamically typed language, which supports object oriented and functional programming.

2.2.6. PyCharm. PyCharm is a powerful Python IDE, which supports smart code completion, code inspections, on-the-fly error highlighting and quick-fixes etc.

3. Pre-requisites

For this project, no previous knowledge in cryptography is required. For the implementation of the DPSS, some experience in the Python programming language is required. As our goal is to provide some type of benchmarking, it is important to know about data structures, threading and networking.

4. Scientific Research

4.1. Secret Sharing

4.1.1. Shamir Secret Sharing.

4.2. Polynomial Commitments

4.2.1. KZG Polynomial Commitments.

4.3. Elliptic Curves

4.3.1. Pairing Friendly Elliptic Curves.

4.4. DPSS

4.5. Assessment (± 15% of section's words)

Provide any objective elements to assess that your deliverables do or do not satisfy the requirements described above.

5. A Technical Deliverable 1

For each technical deliverable targeted in section 2.2 provide a full section with all the subsections described below. The cumulative volume of all deliverable sections represents 75% of the paper's volume in words. Volumes below are indicated relative the the section.

5.1. Requirements (± 15% of section's words)

cf. section 5 applied to the technical deliverable

5.2. Design (± 30% of section's words)

cf. section 5 applied to the technical deliverable

5.3. Production (± 40% of section's words)

cf. section 5 applied to the technical deliverable

5.4. Assessment (± 15% of section's words)

cf. section 5 applied to the technical deliverable

Acknowledgment

The authors would like to thank the BiCS management and education team for the amazing work done.

6. Conclusion

The conclusion goes here.

7. Plagiarism statement

I declare that I am aware of the following facts:

- As a student at the University of Luxembourg I must respect the rules of intellectual honesty, in particular not to resort to plagiarism, fraud or any other method that is illegal or contrary to scientific integrity.
- My report will be checked for plagiarism and if the plagiarism check is positive, an internal procedure will be started by my tutor. I am advised to request a pre-check by my tutor to avoid any issue.
- As declared in the assessment procedure of the University of Luxembourg, plagiarism is committed whenever the source of information used in an assignment, research report, paper or otherwise published/circulated piece of work is not properly acknowledged. In other words, plagiarism is the passing off as one's own the words, ideas or work of another person, without attribution to the author. The omission of such proper acknowledgement amounts to claiming authorship for the work of another person. Plagiarism is committed regardless of the language of the original work used. Plagiarism can be deliberate or accidental. Instances of plagiarism include, but are not limited to:

  1) Not putting quotation marks around a quote from another person's work
  2) Pretending to paraphrase while in fact quoting
  3) Citing incorrectly or incompletely
  4) Failing to cite the source of a quoted or paraphrased work
  5) Copying/reproducing sections of another person's work without acknowledging the source
  6) Paraphrasing another person's work without acknowledging the source
  7) Having another person write/author a work for oneself and submitting/publishing it (with permission, with or without compensation) in one's own name ('ghost-writing')
  8) Using another person's unpublished work without attribution and permission ('stealing')
  9) Presenting a piece of work as one's own that contains a high proportion of quoted/copied or paraphrased text (images, graphs, etc.), even if adequately referenced

  Auto- or self-plagiarism, that is the reproduction of (portions of a) text previously written by the author without citing that text, i.e. passing previously authored text as new, may be regarded as fraud if deemed sufficiently severe.

References

[1] Y. Frankel et al. "Optimal-resilience proactive public-key cryptosystems". In: *Proceedings 38th Annual Symposium on Foundations of Computer Science*. 1997, pp. 384–393. DOI: 10.1109/SFCS.1997.646127.

[2] Sanjam Garg et al. "Witness encryption and its applications". In: *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*. Ed. by Dan Boneh, Tim Roughgarden, and Joan Feigenbaum. ACM, 2013, pp. 467–476. DOI: 10.1145/2488608.2488667. URL: https://doi.org/10.1145/2488608.2488667.

[3] Vipul Goyal et al. "Storing and Retrieving Secrets on a Blockchain". In: *IACR Cryptol. ePrint Arch.* (2020), p. 504. URL: https://eprint.iacr.org/2020/504.

[4] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. "Constant-Size Commitments to Polynomials and Their Applications". In: *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*. Ed. by Masayuki Abe. Vol. 6477. Lecture Notes in Computer Science. Springer, 2010, pp. 177–194. DOI: 10.1007/978-3-642-17373-8_11. URL: https://doi.org/10.1007/978-3-642-17373-8%5C_11.

[5] Leslie Lamport, Robert Shostak, and Marshall Pease. "The Byzantine Generals Problem". In: *ACM Transactions on Programming Languages and Systems* (July 1982), pp. 382–401. URL: https://www.microsoft.com/en-us/research/publication/byzantine-generals-problem/.

[6] Adi Shamir. "How to Share a Secret". In: *Commun. ACM* 22.11 (1979), pp. 612–613. DOI: 10.1145/359168.359176. URL: http://doi.acm.org/10.1145/359168.359176.

8. Appendix

All images and additional material go there.

8.1. Source Code

The following environment shows the correct and mandatory way to insert your code.

Listing 1: Caption example.

```
import numpy as np

def incmatrix(genl1,genl2):
    m = len(genl1)
    n = len(genl2)
    M = None #to become the incidence matrix
    VT = np.zeros((n*m,1), int) #dummy variable

    #compute the bitwise xor matrix
    M1 = bitxormatrix(genl1)
    M2 = np.triu(bitxormatrix(genl2),1)

    for i in range(m-1):
        for j in range(i+1, m):
            [r,c] = np.where(M2 == M1[i,j])
            for k in range(len(r)):
                VT[(i)*n + r[k]] = 1;
                VT[(i)*n + c[k]] = 1;
                VT[(j)*n + r[k]] = 1;
                VT[(j)*n + c[k]] = 1;

                if M is None:
                    M = np.copy(VT)
                else:
                    M = np.concatenate((M, VT), 1)

                VT = np.zeros((n*m,1), int)

    return M
```