Rose-Hulman Institute of Technology
Rose-Hulman Scholar

Rose-Hulman Undergraduate Research Publications

Summer 5-14-2019

Comparison between Two Group Signature Schemes

Hao Yang

Follow this and additional works at: https://scholar.rose-hulman.edu/undergrad_research_pubs

 Part of the Computer Sciences Commons, and the Engineering Commons

Comparison between Two Group Signature Schemes

Hao Yang

May 14, 2019

Abstract

Zerocoin [9] is a cryptographic extension to Bitcoin [12]. During its development, the developers decided to make use of group signature schemes to store and verify the coins. In order to compare the performance of Simple Authentication Scheme [3] and the Dynamic Signature Scheme [7] and figure out which one is the optimal choice for the Zerocoin scheme, I implemented them in Java and analyzed them theoretically. This paper will discuss the performance difference between two schemes, the Java implementation of them and the analysis.

1 Introduction

Bitcoin is popular nowadays, and its value has increased greatly since it was created. Meanwhile, other cryptocurrency schemes emerged and some of them require to verify a coin's validity without leaking extra information. Zerocoin is a cryptographic extension to Bitcoin [9]. It allows fully anonymous Bitcoin currency transactions by providing a distributed e-cash system that breaks the link between individual Bitcoin transactions without adding trusted parties. That is, no third parties are involved.

To achieve this aim, the Zerocoin proposers introduced a decentralized e-cash scheme. In the scheme, the user's information is hidden and the only thing he will show during the transaction is the serial number he chose and a zero-knowledge proof of knowledge that demonstrates the coin being spent is in the set of arbitrary valid coins. Therefore, it is hard for attackers to track down the user based on the coin they received.

When Zerocoin scheme is trying to determine whether a coin appears in a set without harming information of their users, it utilizes the group signature scheme, which allows a member to sign a message on the behalf of the group or to prove his membership. Since the goal is to show that the coin belongs to a set of valid coins without leaking information of the whole set, it is similar to showing a member belongs to a group. Despite that there are multiple kinds of schemes with different properties, the creators of Zerocoin eventually chose to adopt a design that is based on the Dynamic Signature Scheme [7].

To figure out the reason why the developers chose design similar to the Dynamic Signature Scheme, I looked into the schemes they mentioned and considered. Eventually, I decided to compare the Simple Authentication Scheme [3] and the Dynamic Signature Scheme [7]. The Simple Authentication Scheme is chosen here for its capability of representing the other Schemes that do not have the desired property that Dynamic Signature Scheme has. The Dynamic Signature Scheme allows the new member to be added with constant time and not changing the public keys, while the Simple Authentication Scheme does not. Therefore, I chose to analyze the difference between them.

2 Definitions

We begin by defining the necessary terms for this paper.

- Quasi-commutative [3]: For function $h : X \times Y \to X$: we have: $h(h(x, y1), y2) = h(h(x, y2), y1)$.
- Modular Exponentiation [13]: Given base a, exponent b and modulus n, the modular exponentiation e is $e = a^b \bmod n$. Readers should notice that modular exponentiation is quasi-commutative.
- One-way Accumulator [3]: A one-way accumulator is a family of one-way hash functions each of which is quasi-commutative.
- Simple Authentication Scheme [3]: A group signature scheme which utilized the One-way Accumulator and allowed members to prove their membership without leaking extra information of the group.
- Double discrete logarithm [7]: Let $G = \langle g \rangle$ be a cyclic group. The double discrete logarithm of $y \in G$ to base g and a is smallest positive integer b which satisfies $y = g^{a^b}$.
- $b - th$ root of discrete logarithm [7]: Let $G = \langle g \rangle$ be a cyclic group. The $b - th$ root of discrete logarithm of $y \in G$ to base g is smallest positive integer a which satisfies $y = g^{a^b}$.
- Dynamic Signature Scheme [7]: A group signature scheme which allows members to join the group at constant cost utilizing double discrete logarithms.
- Zero-Knowledge Proof [6]: In cryptography, a zero-knowledge proof or zero-knowledge protocol is a method by which one party (the prover) can prove to another party (the verifier) that they know a value x, without conveying any information apart from the fact that they know the value x.

3 Preliminaries

In this section, we will introduce the two signature schemes that were created by other scholars before. We are going to compare these two schemes in the following sections.

3.1 Simple Authentication Scheme [3]

The group manager picks an initial key k, a Quasi-commutative function h and $N = p \cdot q$, where p and q are two primes with similar length. He will then publish N and k as the public key of the group.

Then, when a member is trying to join the group with private key y_1, the manager will compute value $k' = h(k, y_1)$ and update the public key of the group to be k'. Then, the manager will give k to the new member as the certificate (denoted as c). When proving the identity, the member will show that $h(c, y_1) = k'$. Then, the authentication protocol will recognize the membership.

Notice that if there are also other members in the group when a new member joins, the manager needs to update their certificates as well. Assume the new member has private key y', for each member with certificate c, the manager need to compute $c' = h(c, y')$ and update their certificate.

3.2 Dynamic Signature Scheme [7]

When setting up this scheme, the group manager will first perform the standard RSA setup and obtain public keys N and e. Then, the manager will pick a value $a \in \mathbf{Z}_n^*$ from cyclic group $G = \langle g \rangle$ such that G has an order N.

When a new member joins the group, he will choose a private key x, compute the value $y \equiv a^x \pmod{N}$ and the membership key $z = g^y$. Then, he will send y and z to the group manager (while holding x in secret), who will verify that this new member does know the $y-th$ root of g. The manager will then provide him with the certificate $v \equiv (y+1)^{1/e} \pmod{N}$. When proving the membership, the member will construct a zero-knowledge proof that he holds valid key triples (x, y, v) using double discrete logarithm and $b-th$ root of discrete logarithm. The group will help to verify the proof when needed, however, since it is a zero-knowledge proof, the group (or the people who are viewing the proof) cannot obtain any information of the triple.

4 Background

When developing Zerocoin, the developers faced the issue in which they would like to efficiently proof that a coin C_i belongs to a set of valid coins $C = \{C_1, C_2, C_3, ..., C_n\}$ without showing the set of coins [9]. Naively, the proof will be the XOR of the Boolean values $C_1 = C_i, C_2 = C_i$, etc. However, such proof will be expensive. Since this proof will be needed for each coin transaction, the costly proof is not desired when the number of transactions is large. Therefore, they looked into existing data structures and protocols to solve this problem.

Originally, the developers of Zerocoin intended to apply the One-way accumulator [3] to the E-cash scheme. Their reasoning is that by utilizing the One-way accumulator and a design similar to the Simple Authentication Scheme, the cryptocurrency scheme can compress the size of the proof and increase the efficiency of it. Since during the proof, the only operation needed is to run the hashing function once.

As I mentioned in Section 2, modular exponentiation is quasi-commutative. In addition, since it is considered to be easy to compute and hard to revert (based on the strong RSA assumption), it is commonly used in the group signature schemes. The One-way accumulator and Simple Authentication Scheme also utilize this advantage. With the help of modular exponentiation, the hashing function can be computed quickly, therefore making the proof easy.

Unfortunately, the Simple Authentication Scheme has its disadvantage. It was not able to satisfy certain properties that the developers desired. For example, the public key of the group keeps changing as new members join. This implies that in the cryptocurrency scheme, the public key of coin set will change whenever a new coin is minted, which is extremely inconvenient and might cause problem if two coins are minted almost at the same time. In addition, despite the fact that the proof can be easily constructed, it is not secure enough. The design of the Simple Authentication Scheme requires the existence of a trusted authority, which is not desired in the Zerocoin scheme (since the Zerocoin scheme aims to protect the privacy). Obviously, involving trusted authority will decrease the level of privacy protection. Besides, since the main purpose of Bitcoin is to avoid relying on a trusted authority, Zerocoin would also like to avoid doing so. Thus, the Simple Authentication Scheme is definitely not the best choice for Zerocoin.

Therefore, the developers moved on and looked at other schemes. Eventually, they chose

Dynamic Signature Scheme [7]. As I introduced in Section 3, the Dynamic Signature Scheme will keep the privacy of the users. Even though the group manager (or something equivalent) is still needed to generate certificates and verifying proofs, the user's private key and private triples will not be leaked during the whole process. Therefore, they decided that the scheme is ideal for Zerocoin and developed Zerocoin scheme based on it.

Based on the choices made by the Zerocoin developers, I decided to focus on these two schemes. Even though one of them is not perfectly suitable for Zerocoin, it represents the alternative schemes which we should consider and therefore is worth studying. In the following sections, I will talk about my findings of these two schemes.

5 Brief Analysis of Two Schemes

After implementing the two schemes, I realized that the main difference between these two schemes is the way they manage their members' certificate. In the Simple Authentication Scheme, when a member tries to join the group, he/she will provide the group manager with his private key and the manager will compute a certificate based on the public key of the group for the member. Then, since the Simple Authentication Scheme makes use of a One Way Accumulator, the group manager needs to update the public key of the group as well as modify all the certificates of other users. This makes adding new member costly when the group size grows.

In the dynamic signature scheme, the group manager does not have to do any extra work after computing the certificate for the new member. However, proving the membership for the dynamic scheme is more expensive compared with the simple authentication scheme. The reason is that this scheme utilized zero-knowledge proof for the double discrete logarithm that used when generating certificates. Constructing the zero-knowledge proof will require more computations, which leads to slower proof of membership.

In conclusion, when choosing one of these two schemes, the developers are facing the trade-off between faster member adding and faster membership proving. When applying to Zerocoin, we need to take the number of coins minted (which influence the number of member added) and transactions per coin (which influence the number of membership proving) into consideration before making the final decision. To compare the two signature schemes, I implemented them using Java, analyzed them theoretically and timed their performance. In the following sections, I will go into details about the comparison of their performance.

6 Performance Comparison

6.1 Theoretical Performance Comparisons

In the implementation of both signature schemes, the most costly operation is modular exponentiation. This operation computes the modular exponentiation of an integer over another. Since both schemes are based on such operation, I chose to focus on this operation during the theoretical analysis.

In the Simple Authentication Scheme, there are $N + 2$ modular exponentiation executions when adding a new member to a group with size N. The group needs to first compute its public key and new member's certificate using modular exponentiation. Then, it has to update all existing members' certificates by executing modular exponentiation. Notice that the update

computation cannot be done on the members' side. Otherwise, they will know the information of the new member.

On the other hand, verifying membership for the Simple Authentication Scheme is easy. Only one modular exponentiation is enough due to the scheme's design. Therefore, the cost of adding a new member and performing one verification of membership is $N + 3$.

In the Dynamic Signature Scheme, in contrast, there is far less number of modular exponentiation executed when adding a new member. The group simply need to compute 3 modular exponentiation to generate the certificates needed by the new member. It does not need to change the existing public keys or modify the certificates of other members. However, the zero-knowledge proof when verifying membership requires $128 \cdot 2 = 256$ modular exponentiation operations in total. Therefore, adding a new member and performing one verification of membership for the Dynamic Signature Scheme needs 259 modular exponentiation.

It should be intuitive that the Dynamic Signature Scheme will outperform the Simple Authentication Scheme when the group size grows. The question is when exactly will this happen. Therefore, I assumed that each member is added to the group and performed one membership verification. I tried to find out a number N such that if the number of members is beyond this number, creating a group size of N using Dynamic Signature Scheme (and each member perform one verification) will be more efficient than creating such a group using Simple Authentication Scheme.

To compute the value of N, I solved the equation shown below:

$$\sum_{i=0}^{N}(i + 2 + 1) \leq 259N$$

The integer solution for N is 511. Which means that the threshold is 511 members.

However, it came into my mind that we are trying to apply these two schemes towards the Zerocoin scheme. In the cryptocurrency, a coin will be transacted several times after its creation. Therefore, the number of verification per member will be likely greater than one and the threshold will increase. Since Zerocoin is designed to be an extension of Bitcoin, I plugged in the data of Bitcoin to simulate the actual world usage. At the time this thesis is written, the number of Bitcoins in the market is around 17,629,900 [5] and each Bitcoin is transacted 2450 on average [4].

Plug in 2450 into the equation above, we obtain:

$$\sum_{i=0}^{N}(i + 2 + 2450) \leq (3 + 256 \cdot 2450)N$$

From the equation, we can solve that N is $1,249,501$, which is much higher than our previous result. However, notice that the number of Bitcoin is still much larger than this number. Since Zerocoin is designed to be an extension of Bitcoin, we can assume that similar things can happen for Zerocoin.

6.2 Empirical performance comparison

Besides comparing the two schemes theoretically, I also compared their performance in Java. I measured the time cost of adding a member to the group and verifying his/her membership on my computer (which has an AMD Ryzen 7 1700 eight-core processor and 16GB RAM) tried to find the threshold where the Dynamic Signature Scheme will outperform the Simple Authentication Scheme. Since the results varies for each trial, I ran 10 trials and obtained the range for empirical threshold. After running the trials, I figured out that the empirical threshold ranges from 600 members to 650 members. It is pretty close to the theoretical threshold and the comparison are shown in table 1.

Scheme Name	Number of powmod	Theoretical Threshold	Empirical Threshold Range
Dynamic Scheme	259 per member	511	$600 \sim 650$
Simple Scheme	N+3 per member		

Table 1: Threshold comparison: assuming one transaction (verification) will happen for each coin after being minted (joining the group)

Due to the limited computation power, I was not able to simulate the results when applying Bitcoin scenario. However, we can run the simulation based on different scenarios where the number of transactions varies and then estimate the empirical equilibrium. Again, I ran 10 trials for each scenario and obtained the range for empirical threshold.

Number of transactions	Theoretical Threshold	Empirical Threshold Range
1	511	$600 \sim 650$
5	2551	$2900 \sim 3100$
10	5101	$5700 \sim 6200$

Table 2: Threshold comparison: finding relationship between number of transactions per coin and the threshold

Based on table 2 shown above, we can see that the empirical threshold is roughly $1.1 \sim 1.2$ times the theoretical threshold. Therefore, since the theoretical threshold is 1,249,501. The empirical threshold would be around 1,400,000 to 1,500,000.

6.3 Error Analysis and Explanation

From section 6.1 and 6.2 we can see that the empirical threshold differs from the theoretical threshold. I believe that the error came from multiple aspects.

1. Since I do not have full control of Java's memory space and cache, I cannot make sure that it clears the cache when I switch to another scheme. If Java pre-calculates values and store them for future BigInteger operations, it will affect the performance of the schemes greatly. When performing the theoretical analysis, I was assuming that the cost of each modular exponentiation stays constant. Errors can easily be introduced if this is incorrect for Java.

2. When computing the theoretical threshold, I only take the powmod into consideration. However, there are also other expensive operations when verifying members using Dynamic Signature Scheme. The Java program needs to convert the numbers to strings and process it using SHA-256, which can cost a lot of extra time. In addition, the Dynamic Signature Scheme also used extra multiplications and generated random numbers during the process, whereas the Simple Authentication Scheme only used powmod. If the analysis took these into consideration, I believe that the estimated threshold would have been more accurate.

However, since the empirical threshold has the same magnitude as the theoretical threshold, we can say that the analysis is effective and we should be able to extend the results to the cryptocurrency schemes.

7 Implementation decisions

7.1 Number Representation

When implementing the group signature schemes, I first tried to create my own version of BigInteger to support the calculation needed in the schemes. Originally, I believed that Java's BigInteger class is designed for really huge numbers (up to 2^{32} bits) and that's overkill for group signature schemes. Since I would like to speed up the process of adding new members and verifying membership to achieve more accurate performance analysis, I decided to implement a class that optimizes the computation for numbers that have a length around 200 bits (which is the common size of number used in group signature schemes).

My first implementation tried to store the number in their binary form using an array. By storing each bit in a cell, it was easy to perform the bit operations and therefore speed up the computation. However, storing numbers in their binary form greatly increased the number of bits the class needs to loop through whenever an operation is performed. The cost of looping exceeds the cost of the operation itself. Therefore, I switched to the second implementation.

The second implementation stores the number in an array by splitting it into chunks of 24 bits. Each cell will store the integer represented by certain 24 bits from the original integer. By combining them the original value can be easily retrieved. After optimizing, this implementation did achieve slightly better performance compared with Java's BigInteger class. However, after an investigation into Java's BigInteger class, I realized that Java's BigInteger class is actually storing the numbers in a similar way. The only difference is that it splits the number into chunks of 32 bits. Since this is the only difference between the two implementations, I concluded that the better performance of my implementation was caused by less overhead (since BigInteger need to consider more situations) and was trivial.

Eventually, I ended up using Java's BigInteger class for its capability of future development. Since both classes are having similar performance and the BigInteger covers more possibilities of input size (thus the future expansion is easier), I believed that picking BigInteger for number storing is reasonable.

7.2 Error Prevention

As mentioned in section 5.3, the empirical threshold differs from the theoretical threshold. Therefore, I made some modification to my code in order to prevent the errors from happening.

7.2.1 Avoid Java memory space issue

In order to prevent the effect of Java's caching on performance measurement, I need to get rid of the influence of it. Originally, I first simulate the Efficient Signature Scheme and then simulate the Group Signature Scheme. In my new implementation, I put them in the same loop and thus both schemes will add a new member using exactly the same parameter (and this prevent minor errors that can happen) every loop. Therefore, the influence of caching should be minimized.

When running concurrently, the ratio between empirical equilibrium and theoretical Equilibrium decreased to roughly 1.1 : 1 (as shown in Table 3). However, I believed that it is possible to estimate better. Therefore, I tried to eliminate the other error.

Number of transactions	Theoretical Threshold	Empirical Threshold Range
1	511	$560 \sim 580$
5	2551	$2700 \sim 2800$
10	5101	$5200 \sim 5500$

Table 3: Relationship between number of transactions per coin and the threshold when running two schemes concurrently

7.2.2 Not counting operations other than modular exponentiation

Since I proposed that the error can be caused by the extra operations inside the Dynamic Signature Scheme, I modified the timing phase so that it only measures the time cost of the modular power operations. Under the new timing method, other operations are ignored and thus the empirical equilibrium should be close to the theoretical equilibrium. Consequently, we can prevent this error from happening. The results are shown in Table 4:

Number of transactions	Theoretical Threshold	Empirical Threshold Range
1	511	$517 \sim 525$
5	2551	$2525 \sim 2800$
10	5101	$5200 \sim 5500$

Table 4: Relationship between number of transactions and the threshold when running schemes concurrently and only timing modular exponentiation

We can see that after such improvement, the theoretical threshold is almost the same as the empirical threshold. However, the reader should notice that the error eliminated in this step can hardly be ignored in real life scenario. Since the dynamic signature scheme will still contain extra operations besides modular exponentiation, we should still take them into consideration. That is, the empirical threshold will still be 10% higher than the theoretical threshold.

8 Conclusion

In conclusion, the choice of which signature scheme should be used varies based on the scenario. If we estimate that the coin will be used only once after being minted, the threshold is around 600 coins. That is: if the number of coins is less than 600, the Simple Authentication Scheme would be better. If the number of coins is larger than 600, it would be better to adopt the dynamic scheme.

However, in the actual cryptocurrency scheme, the number of transactions is generally higher. If we assume the coins in the scheme will have a similar number of transactions as Bitcoins (which is around 2450 by the time this essay is written), the theoretical threshold will rise to $1,249,501$ and we expect the empirical threshold rise to around 1.5 million. However, in the meantime, the number of Bitcoins existing is around 17,629,900, which is far beyond threshold. Therefore, if the Zerocoin is aiming to be an extension of Bitcoin, it should still choose the Dynamic Signature Scheme.

9 Future Work

In the future, people can work on multiple aspects of this topic. They can discuss whether using new group signature schemes (for example, schemes that are based on Universal Accumulators [8] or fail-stop signature scheme [2]) would be helpful for Zerocoin. They can also plug in the data of Zerocoin (if any) instead of those of Bitcoin and talk about the benefits of Dynamic Signature Scheme if it is still the desired choice. In addition, people can try to implement the Simple Authentication Scheme and Dynamic Signature Scheme in a different language and analyze them to see if they are able to eliminate more errors.

Besides, readers should notice that even though the group signature scheme itself is safe, the cryptocurrency scheme might has some flaws in it. It has been pointed out that the Zerocoin scheme have some flaws and adversary can burn coins through it [10]. Even though this problem is solved Zerocash [11], a more sophisticated scheme created by developers of Zerocoin, it is possible that other potential attacks exist.

For example, my implementation did not take the existing side-channel attack methods against modular exponentiation [1] into consideration. Since the signature scheme is applied to the cryptocurrency scheme, we should be aware that the adversary might attack the system using such a method. If so, the adversary can obtain information about the private keys and then gain information about the coin. Therefore, people can try to implement the modular exponentiation that resists side-channel attacks in the group signature schemes and analyze the new threshold to see if the dynamic signature scheme is still the best choice under this scenario.

10 Acknowledgements

First of all, I would like to express my sincere gratitude to my senior thesis advisor, Prof. Chenette. He guided me throughout my research and thesis writing and made sure that I was on the right track. Whenever I faced difficulties, he was able to give helpful advice and help me out. Without his patience and support, I would not have been able to complete this senior thesis.

Besides my advisor, I would also like to thank Rose-Hulman Institute of Technology for all the resources granted including the access to Cryptography related articles and technology support.

Lastly, I would like to thank my friends and my family. They encouraged me throughout my research and supported me spiritually.

References

[1] J. Ambrose and A. Ignjatovic. *Power Analysis Side Channel Attacks: The Processor Design-level Context*. Omniscriptum Gmbh & Company Kg., 2010.

[2] N. Barić and B. Pfitzmann. Collision-Free Accumulators and Fail-Stop Signature Schemes Without Trees. In G. Goos, J. Hartmanis, J. van Leeuwen, and W. Fumy, editors, *Advances in Cryptology — EUROCRYPT '97*, volume 1233, pages 480–494. Springer Berlin Heidelberg, Berlin, Heidelberg, 1997.

[3] J. Benaloh and M. de Mare. One-Way Accumulators: A Decentralized Alternative to Digital Signatures. In T. Helleseth, editor, *Advances in Cryptology — EUROCRYPT '93*, Lecture Notes in Computer Science, pages 274–285. Springer Berlin Heidelberg, 1994.

[4] BlockChain. Average number of transactions per block. `https://www.blockchain.com/charts/n-transactions-per-block`, 2019. [Online; accessed 14-May-2019].

[5] BlockChain. Bitcoins in circulation. `https://www.blockchain.com/charts/total-bitcoins`, 2019. [Online; accessed 14-May-2019].

[6] M. Blum, A. De Santis, S. Micali, and G. Persiano. Noninteractive Zero-Knowledge. *SIAM Journal on Computing*, 20(6):1084–1118, Dec. 1991.

[7] J. Camenisch and M. Stadler. Efficient group signature schemes for large groups. In B. S. Kaliski, editor, *Advances in Cryptology — CRYPTO '97*, Lecture Notes in Computer Science, pages 410–424. Springer Berlin Heidelberg, 1997.

[8] J. Li, N. Li, and R. Xue. Universal Accumulators with Efficient Nonmembership Proofs. In J. Katz and M. Yung, editors, *Applied Cryptography and Network Security*, volume 4521, pages 253–269. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[9] I. Miers, C. Garman, M. Green, and A. D. Rubin. Zerocoin: Anonymous Distributed E-Cash from Bitcoin. In *2013 IEEE Symposium on Security and Privacy*, pages 397–411, Berkeley, CA, May 2013. IEEE.

[10] T. Rufng, S. A. Thyagarajan, V. Ronge, and D. Schröder. Burning Zerocoins for Fun and for Profit. page 5.

[11] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. Zerocash: Decentralized Anonymous Payments from Bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474, San Jose, CA, May 2014. IEEE.

[12] Wikipedia contributors. Bitcoin — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Bitcoin&oldid=896584517`, 2019. [Online; accessed 14-May-2019].

[13] Wikipedia contributors. Modular exponentiation — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Modular_exponentiation&oldid=895376526`, 2019. [Online; accessed 14-May-2019].