Dining Cryptographers with 0.924 Verifiable Collision Resolution

Christian Franck

Abstract. The dining cryptographers protocol provides strong anonymity and low latency, without requiring a trusted third party. A problem of the protocol is that a malicious participant can disrupt communication by deliberately creating collisions. We propose a computationally secure dining cryptographers protocol, wherein it is possible to verify that a participant correctly executes a collision resolution algorithm. Further, we show that a maximum stable throughput of 0.924 messages per transmission slot can be achieved.

1 Introduction

The dining cryptographers protocol [4] implements a multiple access channel in which senders and recipients are anonymous. In one round of the protocol, each participant broadcasts a ciphertext (O), which may or may not contain a message (M). The encryption vanishes when the ciphertexts of all participants are combined (e.g., $\prod_i O^{(i)}$). If exactly one ciphertext contains a message, then this message appears (e.g., $\prod_i O^{(i)} = M$).

However, there is a collision when several ciphertexts contain a message (e.g., $\prod_i O^{(i)} = M \cdot M' \cdot M''$). A problem of the protocol is that a malicious participant may disrupt the communication by deliberately creating collisions in each round. The identification of such a disruptor is difficult, as an investigation may compromise the anonymity of honest senders.

In this paper, we propose a way to implement a verifiable collision resolution algorithm, which allows to verify the correct participation of every participant, without compromising anonymity. We use ciphertexts with an algebraic structure [8] and the SICTA collision resolution algorithm [12], and we obtain a maximum stable throughput of 0.924. Possible applications exist in the fields of anonymous communication and secret shuffling.

The rest of the paper is organized as follows. In section 2, we discuss algebraic ciphertexts and proof statements for zero-knowledge proofs. In section 3, we show how to use these statement to verify collision resolution algorithms. We analyze the performance of the SICTA algorithm in section 4 and we discuss practical considerations in section 5. In section 6, we review possible applications, and we conclude in section 7.

2 Algebraic Ciphertexts and Zero-Knowledge Proofs

In this section, we review the concepts of algebraic ciphertexts and zero-knowledge proofs, and we propose new statements adapted to prove the correct execution of some collision resolution algorithms.

Golle and Juels showed how to realize a dining cryptographers protocol with algebraic ciphertexts in [8]. In their approach, participants share finite groups $H = \langle h \rangle$ and $G = \langle g \rangle$ with a bilinear map $e : H \times H \to G$ wherein $e(h^a, h^b) = e(h, h)^{ab}$. It is further assumed that the bilinear decisional Diffie-Hellman problem is hard in H and G. In a round j of the protocol, each participant then generates a ciphertext $O_j \in G$. This ciphertext is either of the form

$$O_j = {A_j}^x$$

or of the form

$$O_j = {A_j}^x M.$$

The value $x \in \mathbb{Z}$ is a secret key and $M \in G$ is a message. Because of the decisional Diffie-Hellman assumption, it is impossible to distinguish whether O_j contains a message M or not. The value A_j is computed using the public keys $y = g^x$ of the other participants. For example, n participants $P^{(1)}, ..., P^{(n)}$ compute

$$A_j^{(i)} = e\left(\prod_{k=1}^{i-1} y^{(k)} \prod_{k=i+1}^{n} 1/y^{(k)}, R_j\right),$$

wherein R_j is a public random number for round j. The so obtained values have the property

$$\prod_{k=1}^{n} \left(A_j^{(k)}\right)^{x^{(k)}} = 1.$$

This means that when the ciphertexts $O_j^{(1)}...O_j^{(n)}$ are multiplied, the factors $(A_j^{(1)})^{x^{(1)}}...(A_j^{(n)})^{x^{(n)}}$ cancel and only the messages remain.

The algebraic structure of the ciphertexts makes it possible to prove statements about them using zero-knowledge proofs. Such a zero-knowledge proof allows a prover to prove to a verifier that a given statement holds, without giving the verifier any further information. I.e., the verifier cannot compute anything that he could not have computed before. One can for instance prove the equality of discrete logarithms to different bases, and logical $\wedge$ (and) and $\vee$ (or) combinations thereof [3]. Proofs for the statements used herein are provided as examples in Appendix A.

We propose a new kind of statements to verify the correct execution of collision resolution protocols. So far, zero-knowledge proofs used in dining cryptographers protocols contain statements about individual ciphertexts. E.g., the statement

$$\log_{A_1}(O_1) = \log_g y$$

holds when ciphertext O_1 is empty (i.e., $O_1 = A_1^x$). However, it is also possible to construct statements that hold when there is a relation between two or more ciphertexts. E.g., the statement

$$\log_{A_1/A_2}(O_1/O_2) = \log_g y$$

holds when both ciphertexts O_1 and O_2 encode the same message M (or when both encode no message). It is thus possible to construct more complex statements in order to verify the retransmission of a message.

Example 1. The ciphertext O_2 either contains no message, or the same message as O_1, when the statement

$$\left(\log_{A_2/A_1}(O_2/O_1) = \log_g y\right) \vee \left(\log_{A_2}(O_2) = \log_g y\right)$$

holds.

Example 2. At most one ciphertext out of $O_2, ..., O_k$ contains the same message as O_1 (and the rest of $O_2, ..., O_k$ contain no message), when the statement

$$\left(\log_{A_2...A_j/A_1}(O_2...O_j/O_1) = \log_g y\right) \vee \left(\log_{A_j}(O_j) = \log_g y\right)$$

holds for $j \in \{2, ..., k\}$.

Example 3. Exactly one ciphertext out of $O_2, ..., O_k$ contains the same message as O_1 (and the rest of $O_2, ..., O_k$ contain no message), when the statement

$$\left(\log_{A_2...A_j/A_1}(O_2...O_j/O_1) = \log_g y\right) \vee \left(\log_{A_j}(O_j) = \log_g y\right)$$

holds for $j \in \{2, ..., k-1\}$, and when additionally the statement

$$\log_{A_2...A_k/A_1}(O_2...O_k/O_1) = \log_g y$$

holds.

Remark 1. Note that in examples 2 and 3 it is not sufficient to consider only the last statement, as a participant could encode $O_2 = A_2^x E$ and $O_3 = A_3^x E^{-1}$ instead of $O_2 = A_2^x$ and $O_3 = A_3^x$. In the multiplication $O_2 O_3 ...$, the factors E and E^{-1} would cancel, and the statement would hold. It is therefore necessary to consider each statement.

The statements seen in this section can be used to realize verifiable collision resolution mechanisms, as we see in the next section.

3 Verifiable Collision Resolution

In this section we review the principles of tree based collision resolution and we show how to verify that a participant properly participates in the collision resolution process.

A collision occurs when several participant try to transmit a message in the same round. That is, in a round j, each participant provides a ciphertext O_j. If several of these ciphertexts contain a message, the combination of all these ciphertexts $C_j := \prod_{i=1}^{n} O_j^{(i)}$ only provides a multiplication of all the messages and no meaningful information is transmitted. The purpose of a collision resolution algorithm is to resolve such a collision. In tree based collision resolution algorithms a collision is repeatedly split until all messages are known.

In a simple binary tree collision resolution algorithm, two rounds rounds $2j$ and $2j+1$ are reserved for a collision occurs in round j. Messages involved in the collision are retransmitted randomly in

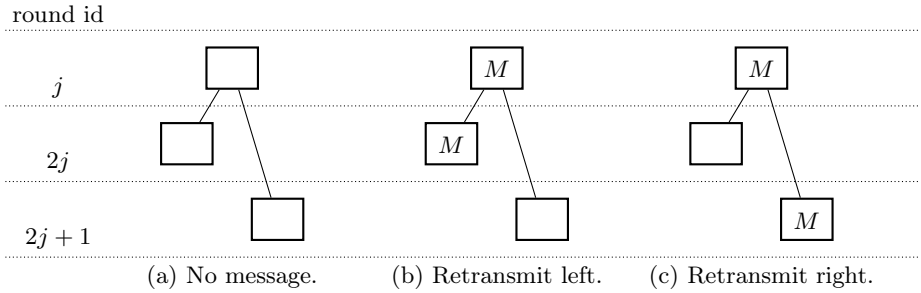

Fig. 3.1. Collision resolution in a binary tree with blocked access. A message involved in the in the collision in round j may be retransmitted either in round $2j$ or round $2j + 1$. No new message may be sent during the collision resolution process.

either of these two rounds. We consider collision resolution algorithms that operate in blocked access mode. This means that no new message may be sent until all collisions are resolved. Figure 3.1 illustrates that only a message M that was sent in round j may be retransmitted. A participant can prove that his ciphertexts O_{2j} and O_{2j+1} are correct, without revealing if any of them contains a message, by proving in zero-knowledge for O_{2j} that

$$\left(\log_{A_j/A_{2j}}(O_j/O_{2j}) = \log_g y\right) \vee \left(\log_{A_{2j}}(O_{2j}) = \log_g y\right)$$

holds and then for O_{2j+1} that

$$\left(\log_{A_j/A_{2j}A_{2j+1}}(O_j/O_{2j}O_{2j+1}) = \log_g y\right)$$

holds. The process is repeated recursively until all collisions are resolved. Note that a technique to implement the proofs is described in Appendix A.

An algorithm with a higher throughput is the SICTA (Successive Inference Cancellation Tree Algorithm) algorithm [12]. It is similar to the simple binary tree algorithm, with the difference that it is not necessary to transfer O_{2i+1}. The result C_{2j+1} of round $2j + 1$ is inferred from C_j and C_{2j}, with $C_{2j+1} = C_j/C_{2j}$. An example of a SICTA collision resolution tree is shown in Figure 3.2. It is again possible to prove in zero-knowledge that a transmitted ciphertext O_{2j} is correct. However, since a corresponding parent O_j may not have been transferred, one must consider two cases:

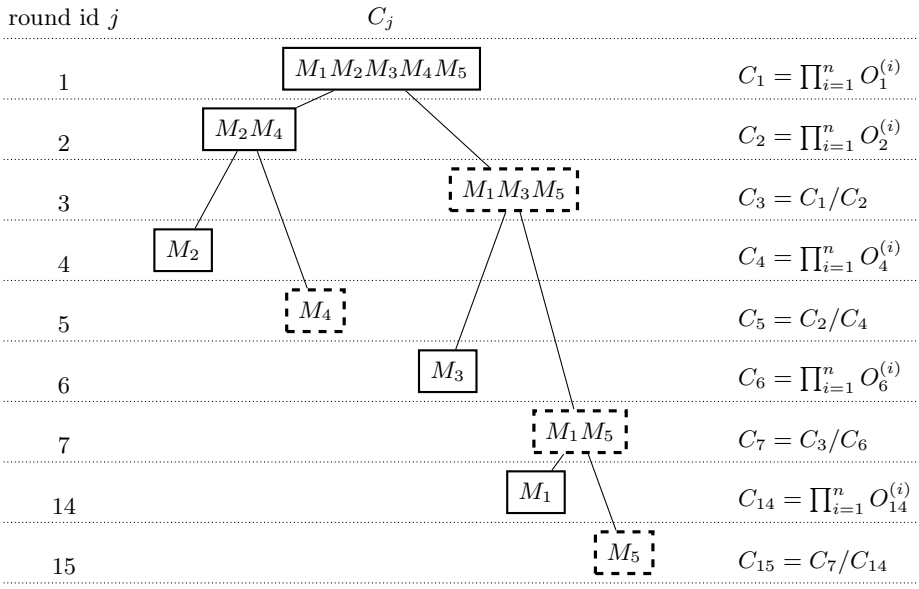

Fig. 3.2. Exemplary binary collision resolution tree with successive inference cancellation (SICTA). In rounds 1,2,4,6 and 14, ciphertexts O_j are transmitted, and C_j is computed using these ciphertexts. In rounds 3,5,7 and 15, no data is transmitted and C_j is computed using data from the parent and the sibling node.

– When a parent O_j exists, a participant can prove that his O_{2j} is correct by proving that

$$\left(\log_{A_j/A_{2j}}(O_j/O_{2j}) = \log_g y\right) \vee \left(\log_{A_{2j}}(O_{2j}) = \log_g y\right)$$

holds.

– When no corresponding O_j has been transmitted, a participant must prove that a message contained in the nearest transmitted parent round was transmitted at most once in all the branches down to O_{2j}. Akin to Example 2, a participant proves for each O_{2j} that

$$\left(\log_{A_{j_t/2}/A_{j_1}\ldots A_{j_t}}(O_{j_t/2}/O_{j_1}\ldots O_{j_t}) = \log_g y\right) \vee \left(\log_{A_{2j}}(O_{2j}) = \log_g y\right)$$

holds, wherein $j_1 := 2j$, $j_k := (j_{k-1}/2) - 1$ and t such that $j_t/2$ is the index of the nearest transmitted parent round of round j.

Example 4. In the collision resolution process shown in Figure 3.2, a participant proves for O_2 that

$$\left(\log_{A_1/A_2}(O_1/O_2) = \log_g y\right) \vee \left(\log_{A_2}(O_2) = \log_g y\right)$$

holds, then for O_4 that

$$\left(\log_{A_2/A_4}(O_2/O_4) = \log_g y\right) \vee \left(\log_{A_4}(O_4) = \log_g y\right)$$

holds, then for O_6 that

$$\left(\log_{A_1/A_2A_6}(O_1/O_2O_6) = \log_g y\right) \vee \left(\log_{A_6}(O_6) = \log_g y\right)$$

holds, then for O_{14} that

$$\left(\log_{A_1/A_2A_6A_{14}}(O_1/O_2O_6O_{14}) = \log_g y\right) \vee \left(\log_{A_{14}}(O_{14}) = \log_g y\right)$$

holds.

We have thus shown that a participant can prove in zero-knowledge that each ciphertext O_j transmitted during the SICTA collision resolution process is correct. In the next section, we investigate the performance of the SICTA algorithm.

4 Performance

We consider the maximum stable throughput (MST), which denotes the maximal input rate (messages/slot) for which all messages have a finite delay. Therefore we define S_k as the average number of slots needed to resolve a collision of k messages, and we consider the throughput k/S_k.

In the SICTA algorithm, a collision of k messages is split into two collisions with i and $k-i$ messages with a probability $\binom{k}{i}2^{-k}$. Thus we have

$$S_k = \sum_{i=0}^{k} \binom{k}{i} 2^{-k}(S_i + S_{k-i}).$$

With $\binom{k}{i} = \binom{k}{k-i}$ this can be written as

$$S_k = \sum_{i=0}^{k} \binom{k}{i} 2^{1-k} S_i$$

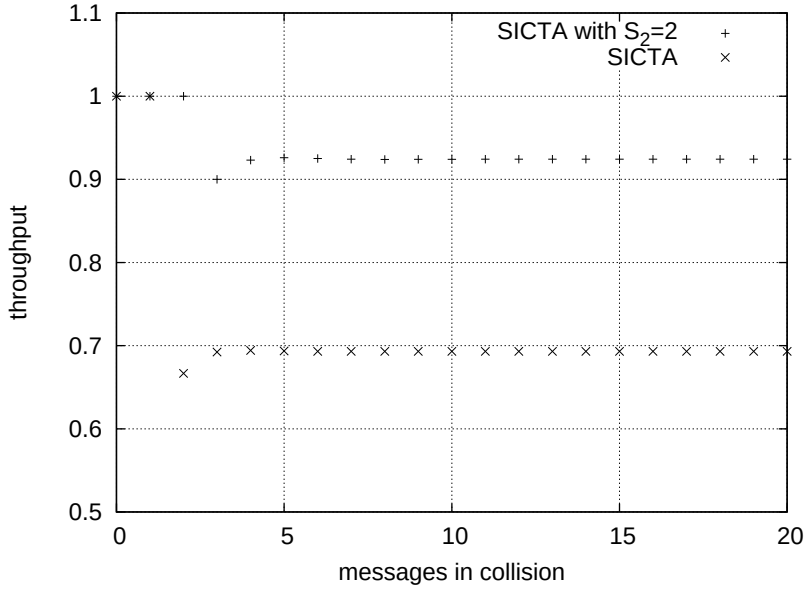

Fig. 4.1. Performance of collision resolution with SICTA.

and after removing the recursion we obtain

$$S_k = \frac{2^{1-k}}{1-2^{1-k}} \sum_{i=0}^{k-1} \binom{k}{i} S_i.$$

As 'collisions' with 0 or 1 messages take only 1 slot, we have $S_0 = S_1 = 1$. The throughput k/S_k for increasing values of k is shown in Figure 4.1. We observe for SICTA the known MST of 0.693.

We can achieve a higher throughput by exploiting the fact that in the dining cryptographers protocol all senders are also receivers. After a collision of two messages, the two respective senders can recover each other's message by removing their own from the collision. They can then avoid a further collision by using a rule that for instance only the numerically smaller message is resent. This way, collisions of two messages are always resolved in two slots. I.e., we have $S_2 = 2$, which leads to a MST of 0.924.

5 Security Considerations

Malicious participants may attempt to prevent the collision resolution process from terminating. For instance,

- colluding participants can always chose the same slot to retransmit their messages, or
- a malicious participant can wait until all other participants have transmitted and then choose to retransmit his message such that a collision occurs, or
- a malicious participant may not send a valid message in the first place.

However, such malicious behavior is easy to detect. In the previously described SICTA algorithm with an MST of 0.924, the probability that a collision does not split is less than or equal to 1/4 (it is exactly 1/4 for collisions with 3 messages). Thus, the probability that a collision does not split k times in a row is less than or equal to $1/4^k$. E.g., the probability that a collision does not split 5 times in a row is below 0.1%. When such malicious activity is detected, one can require commitment before transmission and one can then just skip those branches of the resolution tree that do not split after several attempts. Further it is possible to use zero-knowledge proofs to detect participants that are frequently involved in unresolving collisions.

In many network topologies it is advantageous to have an intermediary collect all the ciphertexts and broadcast the results back to the participants. Such an intermediary can cheat and provide the participants with wrong data. In order to discourage such cheating, one can for instance verify the inputs and the outputs of the intermediary in randomly selected rounds. This way, an intermediary who is cheating repeatedly will eventually be detected. Another possibility to detect such an intermediary is to use verification rounds, in which all participants have to confirm (for instance using an aggregate signature scheme) that they did not detect any unusual activity. If a participant detects a problem (e.g. his message was not transmitted), an investigation phase is started and again the inputs and the outputs of the intermediary are verified.

6 Applications

The described techniques can be used to implement a computationally secure anonymous communication channel with a very low latency. Compared to dining cryptographer protocols that use a reservation phase to avoid collisions, our approach adapts more naturally to situations where participants are frequently joining and leaving the group.

Another application is the realization of secret shuffle algorithms (e.g. [9]). A secret shuffle algorithm is used to obtain a shuffled list of values from a plurality of participants, while keeping it secret which value is coming from which participant. Existing solutions typically require each participant to submit a value. The protocol proposed herein also works efficiently if only a few participants have a value to submit. In particular it may be used to shuffle anonymous public keys for verifiable dining cryptographers protocols wherein rounds are reserved [7,5].

7 Related Work

Superposed receiving [10,11] is a collision resolution technique for the dining cryptographers protocol that achieves throughput of 100%. Therein, messages are elements of an additive group. When a collision occurs, the average of the messages values is computed and only messages whose value is less than this average are retransmitted. Like in SICTA, inference cancellation is used, which leads to the 100% throughput. The problem of this approach is that it is not verifiable, i.e., a malicious participant can disrupt the process.

A fully verifiable dining cryptographers protocol was proposed in [7] and rediscovered in [5]. In this protocol, we have 100% throughput. However, there is the need for a reservation phase which can be lengthy and cumbersome. Current systems are using mixnets to perform the reservations and therefore they are inefficient when only a few reservations are made. Further, they do not easily adapt to situations where participants join or leave frequently.

8 Concluding Remarks

We have shown that it is possible to verify in zero-knowledge that messages are properly retransmitted during a collision resolution process. Using the SICTA collision resolution algorithm, we are able to achieve a maximum stable throughput of 0.924 messages per slot. As possible applications we see the realization of low-latency anonymous communication channels and secret shuffle protocols.

References

1. M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on Computer and communications security*, pages 62–73. ACM New York, NY, USA, 1993.
2. J. Camenisch and M. Stadler. Efficient Group Signature Schemes for Large Groups. *LECTURE NOTES IN COMPUTER SCIENCE*, pages 410–424, 1997.
3. J. Camenisch and M. Stadler. Proof systems for general statements about discrete logarithms. *Technical Report TR 260, Institute for Theoretical Computer Science, ETH Zurich*, Mar. 1997.
4. D. Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1(1):65–75, 1988.
5. H. Corrigan-Gibbs, D. I. Wolinsky, and B. Ford. Proactively accountable anonymous messaging in verdict. In *USENIX Security*, 2013.
6. A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology*, volume 86, pages 186–194. Springer, 1986.
7. C. Franck. New Directions for Dining Cryptographers. Master's thesis, University of Luxembourg, Luxembourg, 2008.
8. P. Golle and A. Juels. Dining Cryptographers Revisited. *Advances in cryptology-EUROCRYPT 2004: International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004: Proceedings*, 2004.
9. C Andrew Neff. A verifiable secret shuffle and its application to e-voting. In *Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 116–125. ACM, 2001.
10. A. Pfitzmann. How to implement ISDNs without user observability – Some remarks. *ACM SIGSAC Review*, 5(1):19–21, 1987.
11. M. Waidner. Unconditional Sender and Recipient Untraceability in spite of Active Attacks. *Lecture Notes in Computer Science*, 434:302, 1990.
12. Yingqun Yu and Georgios B Giannakis. Sicta: a 0.693 contention tree algorithm using successive interference cancellation. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 3, pages 1908–1916. IEEE, 2005.

A Proofs

The notation for zero-knowledge proofs used herein is based on the notation introduced in [2]. Secrets of a prover are represented by greek symbols, and values known to a verifier are represented by non-greek symbols. For instance, a proof of knowledge of the discrete logarithm of y to the base g can be written as $\mathcal{PK}\{\alpha : y = g^\alpha\}$, and a proof of equivalence of the discrete logarithms of w_1 to the base g_1 and of w_2 to the base g_2 can be written as $\mathcal{PK}\{\alpha : (w_1 = g_1^\alpha) \wedge (w_2 = g_2^\alpha)\}$. We assume a secure hash function $\mathcal{H}(.)$ and generate non-interactive zero-knowledge proofs according to the Fiat-Shamir heuristic [6,1].

Example 5. A statement of the form

$$\log_{g_1}(w_1) = \log_{g_2}(w_2)$$

corresponds to the zero-knowledge proof

$$\mathcal{PK}\{\alpha : (w_1 = g_1^\alpha) \wedge (w_2 = g_2^\alpha)\}.$$

To construct this proof, a prover randomly choses $b \in \mathbb{Z}$, computes $t_1 = g_1^b$, $t_2 = g_2^b$, $c = \mathcal{H}(t_1, t_2)$, $r = b + c\alpha$, and obtains the proof $Z = (c, r)$. To verify this proof, a verifier computes $t_1' = g_1^r w_1^{-c}$, $t_2' = g_2^r w_2^{-c}$ and accepts if $\mathcal{H}(t_1', t_2') = c$.

Example 6. A statement of the form

$$\big(\log_{g_1}(w_1) = \log_{g_2}(w_2)\big) \vee \big(\log_{g_3}(w_3) = \log_{g_4}(w_4)\big)$$

corresponds to the zero-knowledge proof

$$\mathcal{PK}\{\alpha : ((w_1 = g_1^\alpha) \wedge (w_2 = g_2^\alpha)) \vee ((w_3 = g_3^\alpha) \wedge (w_4 = g_4^\alpha))\}.$$

To construct this proof, a prover knowing α, such that $w_1 = g_1^\alpha$ and $w_2 = g_2^\alpha$, randomly choses $b, r_2, c_2 \in \mathbb{Z}$ and computes $t_1 = g_1^b$, $t_2 = g_2^b$, $t_3 = w_3^{r_2} g_3^{-c_2}$, $t_4 = w_4^{r_2} g_4^{-c2}$, $c = \mathcal{H}(t_1, t_2, t_3, t_4)$, $c_1 = c - c_2$, $r_1 = b + c_1\alpha$ and obtains the proof $Z = (c_1, c_2, r_1, r_2)$. A prover knowing α, such that $w_3 = g_3^\alpha$ and $w_4 = g_4^\alpha$, randomly choses $b, r_1, c_1 \in \mathbb{Z}$ and computes $t_1 = w_1^{r_1} g_1^{-c_1}$, $t_2 = w_2^{r_1} g_2^{-c_1}$, $t_3 = g_3^b$, $t_4 = g_4^b$, $c = \mathcal{H}(t_1, t_2, t_3, t_4)$, $c_2 = c - c_1$, $r_2 = b + c_2\alpha$ and obtains the proof $Z = (c_1, c_2, r_1, r_2)$. To verify the proof Z, a verifier computes $t_1' = g_1^{r_1} w_1^{-c_1}$, $t_2' = g_2^{r_1} w_2^{-c_1}$, $t_3' = g_3^{r_2} w_3^{-c_2}$, $t_4' = g_4^{r_2} w_4^{-c_2}$ and accepts if $\mathcal{H}(t_1', t_2', t_3', t_4') = c$.