國立臺灣大學電機資訊學院資訊工程學系

碩士論文

Department of Computer Science and Information Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Master Thesis

CRAB: 使用者可反駁的匿名廣播

CRAB: Client-Rebuttable Anonymous Broadcast

許育銘

Yu-Ming Hsu

指導教授: 蕭旭君 博士

Advisor: Hsu-Chun Hsiao, Ph.D.

中華民國 112 年 6 月

June, 2023

doi:10.6342/NTU202301028

國立臺灣大學碩士學位論文
口試委員會審定書
MASTER'S THESIS ACCEPTANCE CERTIFICATE
NATIONAL TAIWAN UNIVERSITY

CRAB: 使用者可反駁的匿名廣播

CRAB: Client-Rebuttable Anonymous Broadcast

本論文係許育銘君（學號 R10922003）在國立臺灣大學資訊工程學系完成之碩士學位論文，於民國 112 年 6 月 14 日承下列考試委員審查通過及口試及格，特此證明。

The undersigned, appointed by the Department of Computer Science and Information Engineering on 14 June 2023 have examined a Master's thesis entitled above presented by HSU,YU-MING (student ID: R10922003) candidate and hereby certify that it is worthy of acceptance.

口試委員 Oral examination committee:

（指導教授 Advisor）

系主任/所長 Director:

doi:10.6342/NTU202301028

誌謝

首先，我要向我的指導教授蕭旭君獻上最誠摯的感謝。我在大學四年級加入實驗室，教授每個禮拜都花很多時間跟各組、各專案開會，把大部分的時間都奉獻給學生了。我在寫論文的期間，有時候會臨時想找教授討論，都沒有發生找不到教授的情況。除此之外，教授也提供給我很多機會，像是跟國外的學者合作，還有去 ETHZ 移地研究，讓我學習到非常多。我會好好珍惜這些經驗，希望未來能跟教授一樣貢獻給其他人。

再來，我要感謝實驗室的同學們。首先要感謝王靖傑同學，他是我們 crypto 組的組長，也是我的好戰友，不僅給我很多研究上的建議，也給我很多現實面的幫助，比如說口試期間我在 ETHZ，都是他負責聯絡口試委員以及幫我跟系辦傳遞資料。總而言之，在各方面都幫助我很多。再來我要感謝李洵同學，我在實驗室第一個加入的就是他的專案，讓我學習到了整個做研究的流程、甚至是之後的申請專利、推廣等等，也讓我有幸得到人生第一篇 paper 並且去 NDSS 大開眼界。即便他在離開實驗室之後還有來我口試的彩排給我建議，真的非常感謝。還有很多很值得感謝的同學，這裡就不一一列舉了。

最後，我要感謝李彥寰教授，他算是我做研究的啟蒙老師。我大學時最先找他做專題研究，當時我完全不懂做研究，是他手把手的教我怎麼找 paper、怎麼好好紀錄研究結果，以及展示給別人看。雖然我後來覺得蕭教授的實驗室研究比較

doi:10.6342/NTU202301028

有趣而離開了李教授的實驗室，但是我從他身上學到的研究方法至今仍讓我受益無窮。也許他不知道，但他是我在臺大最感謝的教授之一。

doi:10.6342/NTU202301028

Acknowledgements

First of all, I would like to express my sincere thanks to my advisor, Prof. Hsu-Chun Hsiao. I joined NSLAB since I was a senior undergraduate student. Prof. Hsiao spends a lot of time every week meeting with each group and project. She dedicates most of her time to the students. When I was writing my thesis, sometimes I wanted to discuss with the professor on the fly, and I can always find her. In addition, Prof. Hsiao also provides many opportunities for me, such as collaboration with foreign researchers and research exchange to ETH Zurich, which makes me learn a lot. I will treasure these experiences, and hope that I can contribute to others in the future like my professor.

Next, I would like to thank my labmates. First, I would like to thank Jay. He is the leader of our crypto group, and also my research fellow. He not only gave me useful suggestions for my research, but also gave me a lot of practical help. For example, I was in ETH Zurich during our thesis defense, so he took charge of contacting the committees and helped me to pass on information to the department office. In short, he helped me a lot in many aspects. I would also like to thank Leexun. The first project I joined in NSLAB is Leexun's project, which allowed me to learn the whole process of doing research, and even the subsequent patent application, promotion, etc. I was also fortunate to get my first paper in life and had an eye-opening experience in NDSS. Even after he left the lab, he still came to the rehearsal of my defense and gave me valuable advice. I can't thank him

doi:10.6342/NTU202301028

enough. There are many other labmates who deserve my thanks, so I won't list them all here.

Last but not least, I would like to thank Prof. Yen-Huan Li, my first teacher in doing research. He is my first advisor of special research in bachelor. I was a complete beginner in research, so he taught me all the things like how to find papers, how to document the research results properly, and how to show them to others, etc. Although I later left his lab because I found the research at NSLAB more interesting, I still benefit from the research methods I learned from him. He may not know, but he is one of my most grateful professors at NTU.

doi:10.6342/NTU202301028

摘要

匿名廣播有很多應用，如吹哨者和電子投票。一些研究是基於混合網路，它們往往需要高成本的混合證明。一些研究是基於多方運算，混合成本較低，但使用者無法驗證其訊息的完整性。在這篇論文中，我們提出了 CRAB，一個基於多方混合協定的匿名廣播系統，為使用者提供驗證機制。該系統是「使用者可反駁」的，意味著使用者可以證明伺服器的錯誤行為。我們證明了 CRAB 滿足安全性的需求。我們也顯示了 CRAB 可以應用於電子投票系統。最後，我們實作並評估了我們的系統。我們的系統可以比使用混合證明的系統快 23～33 倍，並且有良好的可擴展性。因此，我們的系統適用於大規模的電子投票。

關鍵字：匿名廣播、多方運算、匿名性、可驗證性

doi:10.6342/NTU202301028

doi:10.6342/NTU202301028

Abstract

There are many applications of anonymous broadcast, such as whistleblowing and electronic voting. Some works are based on mixnet, and they often require high cost on proof of shuffle. Some works are based on multiparty computation (MPC), where the shuffle cost is lower but the clients cannot verify the integrity of their messages. In this thesis, we propose CRAB, an anonymous broadcast system based on a multiparty shuffling protocol that provides a verification mechanism for the clients. The system is ”client-rebuttable”, which means that a client can prove the misbehavior of the servers. We prove that CRAB satisfies the security requirements. We also show that CRAB can be applied to electronic voting systems. Finally, we implement and evaluate our system. Our system can be 23∼33 times faster than those using proof of shuffle, and our system has good scalability. Thus, our system is suitable for large-scale electronic voting.

Keywords: Anonymous broadcast, MPC, anonymity, verifiability

doi:10.6342/NTU202301028

doi:10.6342/NTU202301028

Contents

Page

口試委員審定書 **i**

誌謝 **iii**

Acknowledgements **v**

摘要 **vii**

Abstract **ix**

Chapter 1 Introduction **1**

Chapter 2 Design Goal **5**

2.1 Threat Model and Assumptions 5

2.2 Security Goals 6

Chapter 3 Background **9**

3.1 Notation 9

3.2 Multiparty Computation 9

3.3 Multiparty Shuffling Protocol 12

Chapter 4 The CRAB System **15**

4.1 Generate Backdoor 15

4.2 Validation Check 17

4.3 The Entire System 22

doi:10.6342/NTU202301028

4.3.1 Generate backdoor 22
4.3.2 Send request 22
4.3.3 Validation check 22
4.3.4 Process 23
4.3.5 Verify 23
4.4 Remove the Trusted Party 23
Chapter 5 Security Proof **25**
5.1 Correctness 26
5.2 Anonymity 26
5.3 Client-rebuttability 28
5.3.1 ”If” direction 29
5.3.2 ”Only if” direction 32
Chapter 6 Application: Electronic Voting **35**
6.1 The E-voting System 36
6.2 Security Analysis 37
Chapter 7 Implementation and Evaluation **39**
7.1 Implementation on Group Elements 39
7.2 Complexity analysis 41
7.3 Experiment Result 41
7.3.1 Client performance 41
7.3.2 Validation check 41
7.3.3 Process 42
7.3.4 Rebuttal check 44

doi:10.6342/NTU202301028

Chapter 8 Related Work **47**

Chapter 9 Conclusion **51**

References **53**

doi:10.6342/NTU202301028

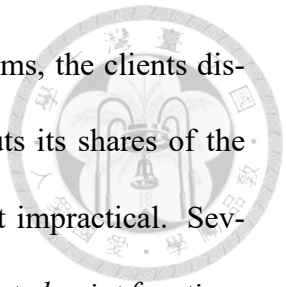
doi:10.6342/NTU202301028

Chapter 1 Introduction

There are many scenarios where people want to broadcast their messages anonymously, such as whistleblowing and electronic voting. In these scenarios, clients send their messages to the servers, and the servers are supposed to output these messages in a random order to break the link between a message and its sender. In addition to anonymity, integrity of the messages is also important. The censored servers may modify or discard some of the messages, and the faulty servers may ignore errors to avoid trouble. Therefore, we need a mechanism to let the clients point out the error if the output is incorrect. Finally, the system should be efficient. Both the generation of the output and the mechanism for the clients should have low costs.

Many anonymous broadcast systems make use of mixnet [13] to mix the messages. In these systems, each client sends the full (encrypted) message to the servers, and the servers in turn shuffle the messages. To prevent any server from altering the messages, the servers are required to provide a zero-knowledge proof for each shuffle [5, 28]. If the output is incorrect, it is difficult for the server to generate a valid proof, so the error can be easily detected by checking the proof. The main problem with mixnet-based approaches is efficiency, because generating the proof of shuffle is costly. Several recent systems [31, 32, 34, 45] divide the shuffle into smaller shuffles to improve performance, but the overall cost is still high.

doi:10.6342/NTU202301028



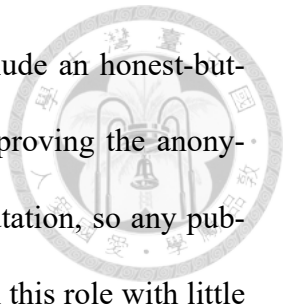
Another type of system is based on DC-net [12]. In these systems, the clients distribute their messages among several servers, and each server outputs its shares of the shuffled messages. However, the high communication cost makes it impractical. Several recent works [1, 16, 38, 46] improve the performance with *distributed point functions* (DPF) [25], where the order of output messages is randomised with ”position” information chosen by the clients. One problem with DPF-based approaches is the collision of the positions, which leads to the problem that some messages may not be published correctly.

Recently, Clarion [23] proposed a protocol in which the clients also distribute their messages among multiple servers, and the servers shuffle the messages together using a multiparty shuffling protocol. Compared to DPF-based approaches, this system has no correctness problem and has lower communication costs. However, a disadvantage of Clarion (and DPF-based approaches) is that they require at least one honest server to ensure the correctness of the multiparty computation. If a client finds that his message is not included in the output, there is no mechanism for him to show that the computation is incorrect.

Therefore, this work presents CRAB, an anonymous broadcast system, to deal with the above problem. In this system, each client secret-shared his message with additional information, called a *backdoor*, to the servers. The servers then run a multiparty shuffling protocol and output the shuffled messages. If a client finds that the output does not contain his message, he can *rebut* it by revealing the backdoor to show that the output is wrong, while keeping his message undisclosed. We define that the systems with such mechanism are *client-rebuttable*.

However, there is an anonymity problem if the client reveals his backdoor directly.

doi:10.6342/NTU202301028



We offer two solutions to this problem. The first solution is to include an honest-but-curious rebuttal authority that is responsible for approving or disapproving the anonymous broadcast. The rebuttal authority is not involved in the computation, so any publicly trusted party, such as a national regulatory authority, can take on this role with little overhead. This scheme is provably secure (see Chapter 5) and can be extended to an electronic voting system (see Chapter 6). The other solution is to allow the clients send ”trap” messages. This scheme is less secure, but the client does not have to trust any other party.

In conclusion, this thesis makes the following contributions:

- We define the security property of client-rebuttability for an anonymous broadcast system.
- We propose a system that achieves correctness, anonymity and client-rebuttability. We also show its application to electronic voting.
- We provide a rigorous proof of the security of our system.
- We implement our system and show that it is efficient and practical to use.

doi:10.6342/NTU202301028

doi:10.6342/NTU202301028

Chapter 2 Design Goal

In the CRAB system, there are k servers and a rebuttal authority to provide the anonymous broadcast service to many clients. The clients can send requests with their messages to the servers, and the servers should help them publish these messages anonymously. After receiving n messages, the servers publish the messages in random order. If a client's message is not published correctly, he can prove the misbehavior of the servers without sacrificing the anonymity of his message.

2.1 Threat Model and Assumptions

We assume that the the clients and the servers can communicate over a secure channel with encryption such as TLS. However, the clients and the servers do not need to trust each other. An adversary can observe the network traffic and control up to all but one server. The adversary can try to find the source of the messages or modify some of the messages. A malicious client can try to prove that the servers are wrong, even if the servers are not.

We also assume that there is a rebuttal authority that is honest-but-curious. The rebuttal authority performs every functionality correctly and does not collude with the adversary, but it can try to find the source of the messages. Communication between the rebuttal authority and the clients is also secure and encrypted. We assume that each party

doi:10.6342/NTU202301028

is polynomial bounded.

Finally, we assume that there is a public bulletin board on which the servers and the rebuttal authority can post data to. The data on the bulletin board cannot be modified or removed. Such a bulletin board can be implemented by a public database or a blockchain.

CRAB does not guarantee availability. Our system cannot defend against denial-of-service attacks or if the servers refuse to accept any message.

2.2 Security Goals

First, we define the security properties of correctness and anonymity:

- **Correctness**: The servers should output a shuffle of all the messages if everyone behaves properly.
- **Anonymity**: No one can learn the source of others' messages.

An anonymous broadcast system should ensure the correctness and anonymity of the messages to some extent. Anonymity can never be satisfied if all the servers collude, so it is common to assume that the adversary can only control some of the servers. We follow Clarion that our system achieves anonymity if at least one server is not controlled by the adversary.

Correctness is ensured if the servers and the clients follow the protocol. However, the malicious or flawed servers may violate the protocol, leading to incorrect output or abortion of the system. Prior works [1] have studied on robustness or censorship-resistance, which mitigates the misbehavior of the servers at additional cost. In contrast, we address

doi:10.6342/NTU202301028

this problem from the client side. A client can rebut the output when he finds that his message is not included. We called this property *client-rebuttability*:

- **Client-rebuttability**: A client can prove the misbehavior of the servers without revealing the ownership of his message.

We mentioned here that the client does not have to reveal his message, so anonymity would not be broken here. We also need to make sure that a malicious client cannot make such a proof if the servers behave properly.

doi:10.6342/NTU202301028

doi:10.6342/NTU202301028

Chapter 3 Background

3.1 Notation

A summary of the notations is presented in Table 3.1. Denote the k servers as $S_1, \ldots, S_k$. Given a set S, denote $s \xleftarrow{r} S$ as uniformly selecting s from S. Let λ be the security parameter. Let $p = 2q + 1$ be a λ-bit safe prime, i.e., q is also a prime. Let $\mathbb{G}$ be the subgroup of $\mathbb{Z}_p^*$ of order q. We assume that the Diffie-Hellman problems on $\mathbb{G}$ are hard, and g_1, g_2, g_3 are public generators of $\mathbb{G}$ where the discrete logarithm problem between them is also hard. Given $x \in \mathbb{Z}_p$, let $[x]$ be a linear secret sharing of x over $\mathbb{Z}_p$. Given a vector $x = (x_1, \ldots, x_n) \in \mathbb{Z}_p^n$, denote by $[x] = ([x_1], \ldots, [x_n])$. Specifically, let $[x]_i$ be the share for server S_i. Let Rec be the reconstruction of the shares, i.e. $x = \mathsf{Rec}([x])$. Our system adopts Clarion, which uses additive secret sharing where $\mathsf{Rec}([x]) = \sum_{i=1}^{k} [x]_i$. However, it is worth noting that our system is compatible with the multiparty shuffling protocol using other linear secret sharing schemes such as Shamir's secret sharing.

3.2 Multiparty Computation

In many privacy-preserving systems, the clients secret-shared their data to multiple servers. The servers can perform some computation on the data, while none of the servers

doi:10.6342/NTU202301028

Table 3.1: Notations.

Notation	Description
n	The number of messages
k	The number of servers
S_i	The i^{th} server, $i = 1, \ldots, k$
λ	Security parameter
p	A λ-bit prime
q	$(p-1)/2$, also a prime
$\mathbb{G}$	The subgroup of $\mathbb{Z}_p^*$ of order q
g_1, g_2, g_3	Public generators of $\mathbb{G}$
$[x]$, $x \in \mathbb{Z}_p$	A share of x
$[x]$, $x \in \mathbb{Z}_p^n$	$([x_1], \ldots, [x_n])$
$[x]_i$	The share for server S_i
$\mathsf{Rec}([x])$	The reconstruction of $[x]$
$s \xleftarrow{r} S$	Uniformly select s from the set S

learn the data during the computation. If less than a threshold number of servers collude, the privacy of the data is guaranteed.

Here we consider additive secret sharing and Shamir's secret sharing. A computation can be represented as an arithmetic circuit with a series of addition and multiplication gates. The servers evaluate one gate at a time, from the input gate to the output gate. For each gate, the value of the input and output wires should be secret-shared between the servers. For example, if a gate has input x, y and should output z, then each server should input $[x], [y]$ and output $[z]$. Finally, the servers collect the shares of the output gate, and reconstruct the output value.

This is easy to do for an addition gate: upon receiving the inputs $[x], [y]$, each server simply outputs $[x] + [y]$ because $\mathsf{Rec}([x] + [y]) = \mathsf{Rec}([x]) + \mathsf{Rec}([y]) = x + y$. For a multiplication gate, however, it is not so simple. We can use the technique proposed by Beaver [6]. Each server holds a *Beaver triple*, which is a group of shares $[a], [b], [c]$ such

doi:10.6342/NTU202301028

that $a \cdot b = c$. Note that a, b, c should remain unknown to the servers, and a triple should not be used twice. We can generate the Beaver triples by random double sharing [21] or somewhat homomorphic encryption [22] in advance. Then we can compute $x \cdot y$ with a Beaver triple. There is a small difference between the computation in additive secret sharing and Shamir's secret sharing:

1. Each server computes $[x - a] = [x] - [a], [y - b] = [y] - [b]$, and sends the values to other servers.
2. The servers reconstruct $d = x - a, e = y - b$.
3. Each server outputs $[z] = [c] + d[y] + e[x] - \frac{de}{k}$ for additive secret sharing and $[z] = [c] + d[y] + e[x] - de$ for Shamir's secret sharing.

Then we have $[z]$ be a secret-share of xy. Indeed, In the case of additive secret sharing, we have:

$$\begin{aligned}
\mathsf{Rec}([z]) &= \mathsf{Rec}([c]) + \mathsf{Rec}(d[y]) + \mathsf{Rec}(e[x]) - \mathsf{Rec}(\frac{de}{k}) \\
&= c + dy + ex - de \\
&= ab + (x - a)y + (y - b)x - (x - a)(y - b) \\
&= xy
\end{aligned}$$

In the case of Shamir's secret sharing, we have $\mathsf{Rec}(de) = de$, so similarly we can show that $\mathsf{Rec}(z) = xy$.

Several techniques are then proposed for different operations, such as exponentiation [3, 19, 39] and non-interactive proof [8, 15, 48]. We will describe some of them in Chapter 4.2.

doi:10.6342/NTU202301028

3.3 Multiparty Shuffling Protocol

The multiparty shuffling protocol is based on multiparty computation. Each server owns a share of the list of n messages, and the servers would eventually output a shuffle of the messages without learning the source of each output message. We can use Clarion [23] as a secure multiparty shuffling protocol.

In Clarion, the servers require more complicated setup, which is called *share translation protocol* [11]. Let $G = \mathbb{Z}_p^\ell$ for some constant integer ℓ. Each server S_i holds a permutation π_i, and it runs share translation protocols with all the other servers S_j. After the translation protocol, the server S_j receives two random vectors $a_{i,j}, b_{i,j} \in G^n$, and server S_i receives a vector $\delta_{i,j} = \pi_i(a_{i,j}) - b_{i,j}$. The translation protocol can be done by oblivious transfer. The servers then generate the *k-party shuffle correlation* using multiparty computation, the details of which are omitted here for simplicity. The k-party shuffle correlation is defined as follows:

- Each server gets random vectors $a_i, b_i, a'_i \in G^n$ and a permutation π_i.
- S_k gets an additional vector:

  $\Delta = \pi_k(\ldots \pi_2(\pi_1(\sum_{i=2}^{k} a_i) + a'_1) \ldots + a'_{k-1}) - \sum_{i=1}^{k-1} b_i$.
- The servers know nothing about each other's value.

Now the servers can shuffle the messages. Let $x \in G^n$ be the input messages. Each server collects the share of the messages $[x]$, then they run the following shuffling protocol:

1. Server S_i (except $i = 1$) computes $z_i = [x]_i - a_i$ and sends it to S_1.

doi:10.6342/NTU202301028

2. S_1 computes $z'_1 = \pi_1(\sum_{i=2}^{k} z_i + [x]_1) - a'_1$ and sends it to S_2. Then he sets his output $[s]_1 = b_1$.

3. For $i = 2, \ldots, k-1$, S_i computes $z'_i = \pi_i(z'_{i-1}) - a'_i$, sends it to S_{i+1}, and sets his output $[s]_i = b_i$.

4. S_k sets his output $[s]_k = \pi_k(z'_{k-1}) + \Delta$.

Then the output $s = \pi_k(\ldots \pi_2(\pi_1(x)) \ldots)$ would be a permutation of x, where none of the servers know the permutation. The protocol does not contain any public-key cryptography, so the computation is very fast.

Clarion requires at least one honest server to ensure the correctness of the protocol. The servers perform several *blind MAC verifications* to check the integrity of the messages. Since CRAB does not use this mechanism, only a simplified introduction is given here. In Clarion, each submitted message is secret-shared with a Carter-Wegman one-time MAC [47] and the corresponding key. The MAC is linearly computed, so the servers can verify the MAC blindly using multiparty computation. The servers perform a blind MAC verification before accepting a message. Then the MAC is shuffled with the message. After the shuffle, the servers perform another blind MAC verification on the entire output to ensure the correctness of the shuffle.

doi:10.6342/NTU202301028

doi:10.6342/NTU202301028

Chapter 4 The CRAB System

In this section, we introduce the CRAB system. Figure 4.1 shows the overall workflow of CRAB. First, a client generates the backdoor and the proof. The backdoor is reserved for rebuttal later, and the details of the backdoor will be presented in Section 4.1. The proof is used for validation check. Then the client sends a request to the servers, and the servers perform a validation check for each request. The details of the request and validation check will be discussed in Section 4.2. After checking all the inputs, the servers process the accepted messages using the multiparty shuffling protocol and output a shuffle of the messages. Finally, the clients verify if their messages are included and report it to the rebuttal authority. The rebuttal authority would then approve or disapprove the anonymous broadcast.

4.1 Generate Backdoor

Recall that $\mathbb{G}$ is the subgroup of $\mathbb{Z}_p^*$ of prime order $q = \frac{p-1}{2}$. To broadcast a message $m \in \mathbb{G}$ anonymously, a client randomly chooses his backdoor secrets $s_1, s_2 \xleftarrow{r} \mathbb{Z}_q$, computes the backdoor $M = m^{s_1} g_3^{s_2}$, and secret-share (m, M) to the servers. More specifically, the clients chooses $[m]_1, \ldots, [m]_k, [M]_1, \ldots, [M]_k \in \mathbb{Z}_p$ such that $\mathsf{Rec}([m]) = m, \mathsf{Rec}([M]) = M$, and sends $([m]_i, [M]_i)$ to server S_i.

doi:10.6342/NTU202301028

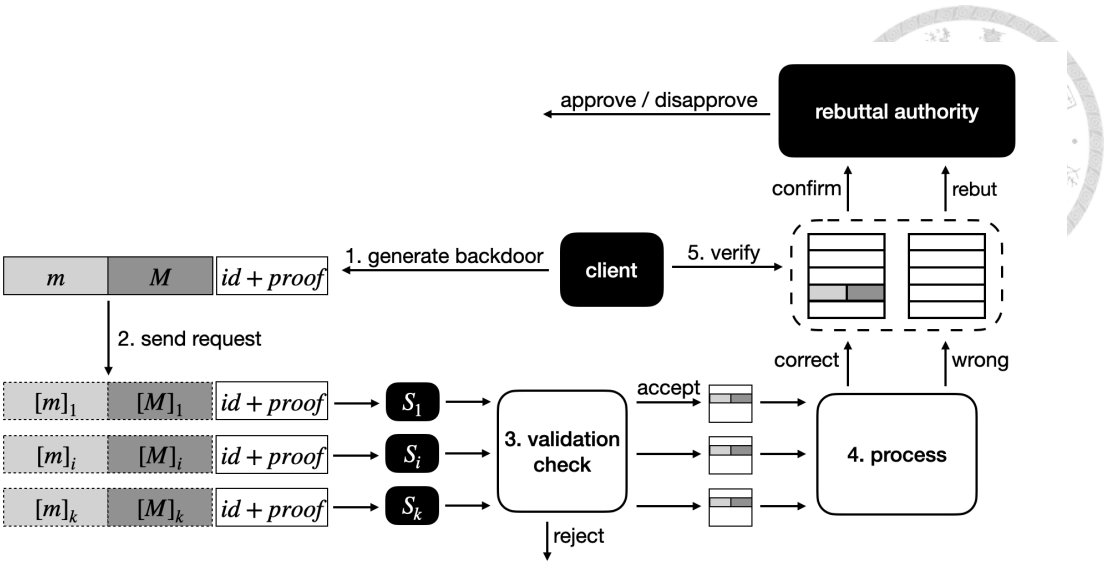
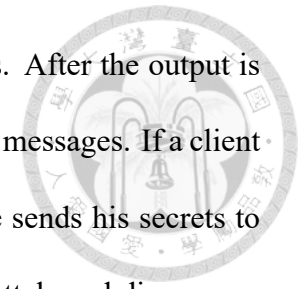


Figure 4.1: The flowchart of CRAB.

The servers execute the multiparty shuffling protocol and output the shuffled messages $(m_1, M_1), (m_2, M_2), \ldots, (m_n, M_n)$. If the client's message $(m, M = m^{s_1} g_3^{s_2})$ does not exist in the output, then with overwhelming probability $M_i \neq {m_i}^{s_1} g_3^{s_2}, \forall i = 1, \ldots, n$. Thus, the client can publish his backdoor secrets, and the others can verify that his message is not included.

However, if the clients publish their secrets directly, the adversary can learn the ownership of their messages. A malicious server can modify some of the output shares and covertly recover the original messages from the output. For example, the server can modify the first j entries of his output shares from $(a_1, A_1), \ldots, (a_j, A_j)$ to $(a_1 + b_1, A_1 + B_1), \ldots, (a_j + b_j, A_j + B_j)$. After seeing the output messages $(m_1, M_1), \ldots, (m_n, M_n)$, the adversary can recover the first j messages $(m_1 - b_1, M_1 - B_1), \ldots, (m_j - b_j, M_j - B_j)$, and the clients with these messages may rebut. When a client publishes his secrets, the adversary searches through $(m_1 - b_1, M_1 - B_1), \ldots, (m_j - b_j, M_j - B_j)$ to find the one that matches the client's secrets. In this way, the malicious server knows what the client intends to publish.

doi:10.6342/NTU202301028



Therefore, we need a rebuttal authority to manage the rebuttals. After the output is published, the rebuttal authority asks all the clients if they found their messages. If a client found his message, he responds with a confirmation. Otherwise, he sends his secrets to the rebuttal authority. The rebuttal authority checks the clients' rebuttals and disapprove the anonymous broadcast if any of them are valid. Note that the clients do not reveal the ownership of their messages to the rebuttal authority.

4.2 Validation Check

In the previous section, we showed that a client can use his secrets to prove that the output is incorrect. However, a malicious client can use false secrets s_1', s_2' to claim the incorrectness. Therefore, we need a mechanism for the servers to check the validity of a client's request before accepting it.

When a client submits his message, he should also provide his identifier $id = g_1^{s_1} g_2^{s_2}$ and a zero-knowledge proof of the consistency of his identifier and message. Specifically, the client proves that he knows the backdoor secrets s_1, s_2 such that $id = g_1^{s_1} g_2^{s_2} \wedge M = m^{s_1} g_3^{s_2}$. Note that id and M follow the form of Pedersen commitment [41], and a zero-knowledge protocol for id is proposed by Okamoto [40]:

- Okamoto's protocol

  1. The prover (client) randomly chooses $r_1, r_2 \xleftarrow{r} \mathbb{Z}_q$ and sends $a = g_1^{r_1} g_2^{r_2}$ to the verifier (server).
  2. The verifier chooses $ch \xleftarrow{r} \mathbb{Z}_q$ and sends it to the prover.
  3. The prover computes $z_1 = ch \cdot s_1 + r_1, z_2 = ch \cdot s_2 + r_2$ and sends them to

doi:10.6342/NTU202301028

Client (input message m):

1. Choose $s_1, s_2, \xleftarrow{r} \mathbb{Z}_q$.
2. Compute $M = m^{s_1} g_3^{s_2}$ and generate shares $[m]_1, \ldots, [m]_k, [M]_1, \ldots [M]_k$.
3. Compute $id = g_1^{s_1} g_2^{s_2}$.
4. Choose $r_1, r_2, \xleftarrow{r} \mathbb{Z}_q$, compute $a = g_1^{r_1} g_2^{r_2}, b = m^{r_1} g_3^{r_2}$ and generate shares $[b]_1, \ldots, [b]_k$.
5. Compute $com_i = H(id, [m]_i, [M]_i, a, [b]_i), \forall i = 1, \ldots, k$, $ch = H'(com_1, \ldots, com_k)$, and $z_1 = ch \cdot s_1 + r_1, z_2 = ch \cdot s_2 + r_2$.
6. Send $id, [m]_i, [M]_i, a, [b]_i, z_1, z_2$ to server S_i.

Server S_i (output *accept* or *reject*):

1. Check if they receive the same id, a, z_1, z_2.
2. Compute $com'_i = H(id, [m]_i, [M]_i, a, [b]_i)$ and send it to all other servers.
3. Compute $ch' = H'(com'_1, \ldots, com'_k)$ and check that $g_1^{z_1} g_2^{z_2} = id^{ch'} a$. If not, output *reject*.
4. Compute $[m^{z_1}]_i$ and $[M^{ch'}]_i$ using multiparty exponentiation.
5. Compute $[d]_i = [m^{z_1}]_i \cdot g_3^{z_2} - [M^{ch'} \cdot b]_i$ and send it to all other servers.
6. Reconstruct d. If $d = 0$, output *accept*. Otherwise, output *reject*.

Figure 4.2: The protocol of validation check.

the verifier.

4. The verifier checks that $g_1^{z_1} g_2^{z_2} = id^{ch} a$.

We start from Okamoto's protocol and finally reach our non-interactive multi-verifier zero-knowledge (NIMVZK) protocol version 4. Protocol version 1 includes the message (m, M) to the proof, while protocol version 2 makes it non-interactive. Protocol version 3 is a straight-forward multi-verfier transform from protocol version 2, and protocol version 4 fixes some bugs in protocol version 3. A clearer description of the final protocol (version 4) can be found on Figure 4.2.

doi:10.6342/NTU202301028

First, to prove that id and M are generated from the same secrets s_1, s_2, we modify Okamoto's protocol to protocol version 1:

- Protocol version 1

  1. The prover randomly chooses $r_1, r_2 \xleftarrow{r} \mathbb{Z}_q$ and sends $a = g_1^{r_1} g_2^{r_2}, b = m^{r_1} g_3^{r_2}$ to the verifier.
  2. The verifier chooses $ch \xleftarrow{r} \mathbb{Z}_q$ and sends it to the prover.
  3. The prover computes $z_1 = ch \cdot s_1 + r_1, z_2 = ch \cdot s_2 + r_2$ and sends them to the verifier.
  4. The verifier checks that $g_1^{z_1} g_2^{z_2} = id^{ch} a \wedge m^{z_1} g_3^{z_2} = M^{ch} b$.

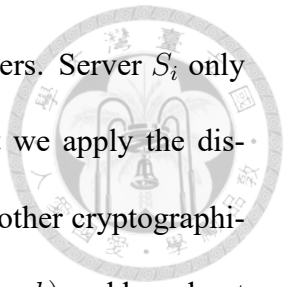
Note that the discrete logarithm problem between g_3 and m might be easy for the prover. However, we show in Theorem 5.4 that the prover cannot use $M \neq m^{s_1} g_3^{s_2}$ to convince the verifier with non-negligible probability.

Next, we want to make the protocol non-interactive. Applying the random oracle assumption to a cryptographically secure hash function H, we can use Fiat-Shamir heuristic to construct protocol version 2:

- Protocol version 2

  1. The prover randomly chooses $r_1, r_2 \xleftarrow{r} \mathbb{Z}_q$ and computes $a = g_1^{r_1} g_2^{r_2}, b = m^{r_1} g_3^{r_2}, ch = H(id, m, M, a, b)$. Then he sends $(a, b, z_1 = ch \cdot s_1 + r_1, z_2 = ch \cdot s_2 + r_2)$ to the verifier.
  2. The verifier computes $ch' = H(id, m, M, a, b)$ and checks that $g_1^{z_1} g_2^{z_2} = id^{ch'} a \wedge m^{z_1} g_3^{z_2} = M^{ch'} b$.

doi:10.6342/NTU202301028



In our case, however, the message is distributed among the servers. Server S_i only knows $[m]_i, [M]_i, g_1, g_2, g_3, id$. To make the distributed proof, first we apply the distributed Fiat-Shamir proposed by Yang and Wang [48]. Let H' be another cryptographically secure hash function. S_i can compute $com_i = H(id, [m]_i, [M]_i, a, b)$ and broadcast to all other servers. Then we can set the challenge as $ch = H'(com_1, \ldots, com_k)$, as we show in protocol version 3.

- Protocol version 3

  1. The client randomly chooses $r_1, r_2 \xleftarrow{r} \mathbb{Z}_q$. Then he computes $a = g_1^{r_1} g_2^{r_2}, b = m^{r_1} g_3^{r_2}, \{com_i = H(id, [m]_i, [M]_i, a, b)\}_{i=1}^k, ch = H'(com_1, \ldots, com_k)$ and sends $(a, b, z_1 = ch \cdot s_1 + r_1, z_2 = ch \cdot s_2 + r_2)$ to each server.
  2. Each server computes $com'_i = H(id, [m]_i, [M]_i, a, b)$ and sends to other servers. Then they compute $ch' = H'(com'_1, \ldots, com'_k)$ and check that $g_1^{z_1} g_2^{z_2} = id^{ch'} a$.
  3. All the servers jointly check that $m^{z_1} g_3^{z_2} = M^{ch'} b$.

However, there is a drawback in protocol version 3. A server can store b, z_1, z_2, ch' for an id. When the list of shuffled messages $(m_1, M_1), \ldots, (m_n, M_n)$ is published, the server can compute $d_i = m_i^{z_1} g_3^{z_2} - M_i^{ch'} b$ for each $i = 1, \ldots, n$. If $d_i = 0$, then the server can match id and (m_i, M_i). To fix this problem, we make version 4, where the client also secret-shares b to the servers. Specifically, the client chooses $[b]_1, \ldots, [b]_k \in \mathbb{Z}_p$ such that $\mathsf{Rec}([b]) = b$, and computes $com_i = H(id, [m]_i, [M]_i, a, [b]_i)$.

- Protocol version 4 (final)

  1. The client randomly chooses $r_1, r_2 \xleftarrow{r} \mathbb{Z}_q$. Then he computes $a = g_1^{r_1} g_2^{r_2}, b = m^{r_1} g_3^{r_2}$ and generates shares $[b]_1, \ldots, [b]_k$. Finally, he computes $\{com_i =$

doi:10.6342/NTU202301028

$H(id, [m]_i, [M]_i, a, [b]_i)\}_{i=1}^k, ch = H'(com_1, \ldots, com_k)$ and sends $(a, [b]_i, z_1 = ch \cdot s_1 + r_1, z_2 = ch \cdot s_2 + r_2)$ to server S_i.

2. Each server computes $com'_i = H(id, [m]_i, [M]_i, a, [b]_i)$ and sends to other servers. Then they compute $ch' = H'(com'_1, \ldots, com'_k)$ and check that $g_1^{z_1} g_2^{z_2} = id^{ch'} a$.

3. All the servers jointly check that $m^{z_1} g_3^{z_2} = M^{ch'} b$.

We remain to show the final step, where the servers perform multiparty computation while none of the servers learn m or M. First, we need to compute the exponentiation with the public exponent and the secret-shared base (m^{z_1} and $M^{ch'}$). We can compute the exponential by square-and-multiply, which requires $O(\lambda)$ multiplications, but the computation can be faster. Here we present a technique proposed by Ning and Xu [39]. The servers execute the following protocol to compute m^{z_1}, where z_1 is public and m is secret-shared:

1. Each server randomly chooses $r_i \xleftarrow{r} \mathbb{Z}_p$ and sends the shares $[r_i], [r_i^{-z_1}]$ to other servers.

2. Each server has the share $[r_1], \ldots, [r_k]$ and $[r_1^{-z_1}], \ldots, [r_k^{-z_1}]$. Let $r = \prod_{i=1}^k r_i$, they can use multiparty multiplication to compute $[r]$ and $[r^{-z_1}]$.

3. The servers use multiparty multiplication to compute $[mr]$ and reconstruct mr.

4. Each server outputs $(mr)^{z_1}[r^{-z_1}]$.

The protocol requires $2k - 1$ multiparty multiplications, which has much lower cost than square-and-multiply when $k \ll \lambda$. Correctness of the protocol can be easily verified:

$$\mathsf{Rec}((mr)^{z_1}[r^{-z_1}]) = (mr)^{z_1}\mathsf{Rec}(r^{-z_1}) = (mr)^{z_1} r^{-z_1} = m^{z_1}$$

doi:10.6342/NTU202301028

Similarly, the servers can obtain $[M^{ch'}]$ and compute $[M^{ch'} \cdot b]$ using multiparty multiplication. Finally, each server computes $[d] = [m^{z_1}] \cdot g_3^{z_2} - [M^{ch'} \cdot b]$, and they jointly reconstruct $d = \mathsf{Rec}([d])$. If $d = 0$, then the servers accept the proof and the message (m, M).

4.3 The Entire System

Here we introduce the entire CRAB system, which uses Clarion as the underlying multiparty shuffling protocol.

4.3.1 Generate backdoor

To broadcast a message m, the client generates backdoor secrets s_1, s_2 and computes the backdoor $M = m^{s_1} g_3^{s_2}$ and the identifier $id = g_1^{s_1} g_2^{s_2}$. Then he shares his message $([m], [M])$ and generate the NIMVZK proof $(a, [b], z_1, z_2)$, where b is shared.

4.3.2 Send request

The client sends the identifier, the shares of his message, and the proof to the servers. More specifically, the client sends $(id, [m]_i, [M]_i, a, [b]_i, z_1, z_2)$ to server S_i for $i = 1, \ldots, k$.

4.3.3 Validation check

When the servers receive a request from a client, they first check if they receive the same identifier and proof (except $[b]$), then they verify the NIMVZK proof as described in Section 4.2. If all checks pass, then the servers accept the message (m, M) and publish

doi:10.6342/NTU202301028

the identifier to the bulletin board.

4.3.4 Process

The servers run the secret-shared shuffling protocol described in Section 3.3. Finally, they publish a shuffle of the accepted messages $(m_1, M_1), (m_2, M_2), \ldots, (m_n, M_n)$.

4.3.5 Verify

After the the list of shuffled messages is published, the rebuttal authority asks each client if his message is on the list. A client responses with a confirmation if his message is included, otherwise he sends his secrets to the rebuttal authority. For each received secrets s_1, s_2, the rebuttal authority checks that (1) $id = g_1^{s_1} g_2^{s_2}$ is on the bulletin board (2) $M_i \neq m_i^{s_1} g_3^{s_2}, \forall i = 1, \ldots, n$. If any of the secrets passes the check, the rebuttal authority disapprove the anonymous broadcast.

4.4 Remove the Trusted Party

The CRAB system introduced above requires an honest-but-curious rebuttal authority. Clients must trust that the rebuttal authority will help them disapprove the anonymous broadcast. However, this trust requirement may be difficult to achieve in some scenarios.

Here we present another scheme for these scenarios. Similar to the idea in previous work [31], a client can send several ”trap” messages with different secrets. These trap messages should be designed in such a way that the adversary cannot distinguish between the trap messages and the real messages. If any of his trap messages is not in the output,

doi:10.6342/NTU202301028

he can rebut with the corresponding secrets. Everyone can check that the message is not included, so the misbehavior of the servers is proven. The adversary can learn the source of the trap messages, but the real messages remain anonymous.

The adversary may be lucky to have only modified the real messages but not any of the client's trap messages. In this case, the client may have to sacrifice the anonymity of his message in order to rebut. So there is a trade-off between security and trust-less requirement.

doi:10.6342/NTU202301028

Chapter 5 Security Proof

In this section, we prove that CRAB satisfies the security requirements described in Chapter 2.2.

First we analyze the information held by each party. Each server knows $(a = g_1^{r_1} g_2^{r_2}, ch', z_1 = ch' \cdot s_1 + r_1, z_2 = ch' \cdot s_2 + r_2)$ for each identifier id. The adversary can output the modified messages while knowing the original messages. The rebuttal authority knows the output messages, some of which may be modified. The rebuttal authority also receives the secrets from the rebuttal, where the secrets do not match any of the output messages.

Before we prove the security of CRAB, we introduce the assumption we need for the proof:

Assumption 5.1. *Let $g_3 = g_2^{t_3}$ for some $t_3 \in \mathbb{Z}_q$ and $s_2, s_3 \xleftarrow{r} \mathbb{Z}_q$. There is no probabilistic polynomial time (PPT) adversary that can distinguish between $(g_2, g_2^{s_2}, g_3 = g_2^{t_3}, g_3^{s_2} = g_2^{s_2 \cdot t_3})$ and $(g_2, g_2^{s_2}, g_3, g_2^{s_3})$ with non-negligible probability.*

This assumption is similar to the decisional Diffie-Hellman assumption, while we fix $t_3 \in \mathbb{Z}_q$ that is unknown to the adversary instead of randomly choosing $t_3 \xleftarrow{r} \mathbb{Z}_q$. So this assumption is reasonable if the generation of g_3 is uniformly random.

doi:10.6342/NTU202301028

5.1 Correctness

If the clients and servers behave properly, the system should accept and output a shuffle of all messages. The correctness of CRAB is achieved when the validation check and the processing are correct. For the validation check, it is easy to see that if a client generates the backdoor and the proof correctly, then the servers should accept his message. The correctness of processing is guaranteed by the underlying multiparty shuffling protocol, such as Clarion [23] in our case.

5.2 Anonymity

Anonymity requires that the source of each message cannot be learned by others. In CRAB, we want to prevent the adversary and the rebuttal authority from linking a message to an identifier. We assume that the adversary does not learn the messages during processing, and the order of the output messages is independent of the order of the identifiers. Then we can show that both the adversary and the rebuttal authority cannot guess the message of each identifier with non-negligible advantage over random guessing. First, we prove that under Assumption 5.1, no one can decide if an identifier matches a message without knowing the underlying secrets:

Theorem 5.2. *Let* $id = g_1^{s_1} g_2^{s_2}$ *where* $s_1, s_2 \xleftarrow{r} \mathbb{Z}_q$. *For any* $m \in \mathbb{G}$, *let* $M = m^{s_1} g_3^{s_2}$ *and* $M' \xleftarrow{r} \mathbb{G}$, *there is no PPT adversary that can distinguish* $(id, (m, M))$ *and* $(id, (m, M'))$ *with non-negligible probability.*

Proof. Suppose that such adversary exists, i.e., there is a PPT adversary Adv and non-negligible function $\epsilon(\cdot)$ s.t. $|\Pr[\mathsf{Adv}(id, (m, M)) = 1] - \Pr[\mathsf{Adv}(id, (m, M')) = 1]| \geq$

doi:10.6342/NTU202301028

$\epsilon(\lambda)$. Then there is another adversary Adv' that can break Assumption 5.1 as follows:

1. The challenger chooses $b \xleftarrow{r} \{0,1\}$ and sends

$$(h_1, h_2, h_3, h_4) = \begin{cases} (g_2, g_2^{s_2}, g_3, g_3^{s_2}) & b = 0 \\ (g_2, g_2^{s_2}, g_3, g_2^{s_3}) & b = 1 \end{cases}$$

   to Adv'.

2. Adv' chooses $s_1 \xleftarrow{r} \mathbb{Z}_q$, $m \in \mathbb{G}$, and sets $id = g_1^{s_1} h_2, M = m^{s_1} h_4$. Then Adv' sends $id, (m, M)$ to Adv.

3. Adv outputs $b' \in \{0,1\}$.

4. Adv' outputs b'.

It is trivial that $\Pr[b' = 1|b = 0] = \Pr[b' = 1|h_4 = g_3^{s_2}] = \Pr[\mathsf{Adv}(id, (m, M)) = 1|M = m^{s_1} g_3^{s_2}]$. Also we know that for random $s_3 \xleftarrow{r} \mathbb{Z}_q$, $g_2^{s_3}$ is uniformly distributed over $\mathbb{G}$, and so is $M = m^{s_1} g_2^{s_3}$. So $\Pr[b' = 1|b = 1] = \Pr[b' = 1|h_4 \xleftarrow{r} \mathbb{G}] = \Pr[\mathsf{Adv}(id, (m, M)) = 1|M \xleftarrow{r} \mathbb{G}]$. Overall, we have

$$\begin{aligned} &\Pr[b' = 1|b = 0] - \Pr[b' = 1|b = 1] \\ =& \Pr[\mathsf{Adv}(id, (m, M)) = 1|M = m^{s_1} g_3^{s_2}] - \Pr[\mathsf{Adv}(id, (m, M)) = 1|M \xleftarrow{r} \mathbb{G}] \\ \geq& \epsilon(\lambda) \end{aligned}$$

, which is non-negligible. So we construct a PPT adversary Adv' that distinguishes $b = 0, 1$ with non-negligible probability, which contradicts Assumption 5.1. So such adversary Adv does not exist. □

Next, we consider the ability of the adversary. By Theorem 5.2, we show that the

doi:10.6342/NTU202301028

adversary cannot find the source of a message (m, M) with only id. Also, according to the security of Okamoto's identifier scheme [40], (id, a, ch', z_1, z_2) does not leak the information about s_1, s_2. Now the only thing that the adversary can make use of is the fact that $m^{z_1} g_3^{z_2} = M^{ch'} b$, where the distribution of b over $\mathbb{G}$ is uniform in the adversary's perspective. However, given any (id, ch', z_1, z_2) and (m, M), there always exists $b = m^{z_1} g_3^{z_2} M^{-ch'}$, so the adversary still cannot decide if id matches (m, M). In summary, the adversary cannot learn if an identifier matches a message, so the adversary can only randomly guess the message of each identifier from the set of the original messages.

Finally, we consider the ability of rebuttal authority. For the identifiers that the rebuttal authority knows the secrets of, the rebuttal authority does not know the original messages of these identifiers. Thus, it cannot find the messages of these identifiers with non-negligible probability. For the other identifiers, the rebuttal authority does not know their secrets, and by Theorem 5.2 it cannot decide if an identifier matches a message. Also, it cannot distinguish whether a message has been modified or not. So the rebuttal authority can only guess the messages of these identifiers randomly from the set of the output messages. Overall, we can also conclude that the rebuttal authority does not have a better strategy than random guessing.

5.3 Client-rebuttability

In CRAB, a client should be able to rebut if and only if the servers modify his message. We will show that both directions cannot be violated with non-negligible probability.

doi:10.6342/NTU202301028

5.3.1 "If" direction

A client can successfully rebut if none of the output messages match his secrets. There are two ways in which the client cannot successfully rebut:

1. The adversary modifies the client's message, but the message still matches the client's secrets.
2. The other messages inadvertently match the client's secret.

In the first case, the adversary knows $id = g_1^{s_1} g_2^{s_2}$ and a message $(m, M = m^{s_1} g_3^{s_2})$ without knowing s_1, s_2. Now we show that the adversary cannot generate $(m', M') \neq (m, M)$ such that $M' = m'^{s_1} g_3^{s_2}$ with non-negligible probability under Assumption 5.1.

Theorem 5.3. *Let* $id = g_1^{s_1} g_2^{s_2}, M = m^{s_1} g_3^{s_2}$ *where* $s_1, s_2 \xleftarrow{r} \mathbb{Z}_q$ *and* $m \in \mathbb{G}$. *There is no PPT adversary that can generate* $(m', M') \neq (m, M)$ *where* $M' = m'^{s_1} g_3^{s_2}$ *with non-negligible probability.*

Proof. Suppose that such adversary exists, i.e., there is a PPT adversary Adv and non-negligible function $\epsilon(\cdot)$ s.t. $\Pr[(m', M') \leftarrow \mathsf{Adv}(id, (m, M)), M' = m'^{s_1} g_3^{s_2} \wedge (m', M') \neq (m, M)] \geq \epsilon(\lambda)$. Then there is another adversary Adv' that can break Assumption 5.1 as follows:

1. The challenger chooses $b \xleftarrow{r} \{0, 1\}$ and sends

$$(h_1, h_2, h_3, h_4) = \begin{cases} (g_2, g_2^{s_2}, g_3, g_3^{s_2}) & b = 0 \\ (g_2, g_2^{s_2}, g_3, g_2^{s_3}) & b = 1 \end{cases}$$

   to Adv'.

doi:10.6342/NTU202301028

2. Adv′ chooses $s_1 \xleftarrow{r} \mathbb{Z}_q$, $m \in \mathbb{G}$, and sets $id = g_1^{s_1} h_2, M = m^{s_1} h_4$. Then Adv′ sends $id, (m, M)$ to Adv.

3. Adv outputs m', M'.

4. Adv′ checks if $(m'm^{-1})^{s_1} = M'M^{-1} \wedge (m', M') \neq (m, M)$. If *true*, Adv′ outputs $b' = 1$. Otherwise, Adv′ chooses $b' \xleftarrow{r} \{0, 1\}$ and output b'.

First we consider the case that $b = 0$, i.e., $M = m^{s_1} g_3^{s_2}$. It is trivial that $(m'm^{-1})^{s_1} = M'M^{-1}$ if and only if $M' = m'^{s_1} g_3^{s_2}$. So the probability that Adv′ outputs 1 is

$$
\begin{aligned}
&\Pr[b' = 1 | b = 0] \\
&= \Pr[b' = 1 | \textit{true}, b = 0] \cdot \Pr[\textit{true} | b = 0] + \Pr[b' = 1 | \textit{false}, b = 0] \cdot \Pr[\textit{false} | b = 0] \\
&= 1 \cdot \Pr[M' = m'^{s_1} g_3^{s_2} \wedge (m', M') \neq (m, M) | b = 0] \\
&\quad + \frac{1}{2} \cdot (1 - \Pr[M' = m'^{s_1} g_3^{s_2} \wedge (m', M') \neq (m, M) | b = 0]) \\
&= \frac{1}{2} + \frac{1}{2} \Pr[M' = m'^{s_1} g_3^{s_2} \wedge (m', M') \neq (m, M) | b = 0] \\
&\geq \frac{1}{2} + \frac{\epsilon(\lambda)}{2}
\end{aligned}
$$

Then we consider the case that $b = 1$, i.e., $M = m^{s_1} g_2^{s_3}$. The probability that Adv′ outputs 1 is

$$
\begin{aligned}
&\Pr[b' = 1 | b = 1] \\
&= \Pr[b' = 1 | \textit{true}, b = 1] \cdot \Pr[\textit{true} | b = 1] + \Pr[b' = 1 | \textit{false}, b = 1] \cdot \Pr[\textit{false} | b = 1] \\
&\leq \Pr[\textit{true} | b = 1] + \Pr[b' = 1 | \textit{false}, b = 1] \\
&= \Pr[\textit{true} | b = 1] + \frac{1}{2}
\end{aligned}
$$

Now we analyze $\Pr[\textit{true} | b = 1]$, i.e., $\Pr[(m'm^{-1})^{s_1} = M'M^{-1} \wedge (m', M') \neq (m, M) | id =$

doi:10.6342/NTU202301028

$g_1^{s_1} g_2^{s_2}, M = m^{s_1} g_2^{s_3}]$. Let $m = g_2^t$ for some $t \in \mathbb{Z}_q$. In the view of Adv, there are q possible solutions $(s_1^i = s_1 + i, s_2^i = s_2 - t_1 \cdot i, s_3^i = s_3 - t \cdot i), \forall i \in \mathbb{Z}_q$ such that $id = g_1^{s_1^i} g_2^{s_2^i} \wedge M = m^{s_1^i} g_2^{s_3^i}$. So s_1 is uniformly distributed over $\mathbb{Z}_q$, which means that Adv does not learn anything about s_1.

It is trivial that when $m' = m, M' \neq M$, $\Pr[(m'm^{-1})^{s_1} = M'M^{-1}] = 0$. When $m' \neq m$, we have $m'm^{-1} \in \mathbb{G}$ and $m'm^{-1} \neq 1$, i.e., $m'm^{-1}$ has order q. So in the view of Adv where s_1 is uniformly distributed over $\mathbb{Z}_q$, $(m'm^{-1})^{s_1}$ is uniformly distributed over $\mathbb{G}$, and $\Pr[(m'm^{-1})^{s_1} = M'M^{-1}] = \frac{1}{q}$. Overall, we have

$$\Pr[b' = 1|b = 0] - \Pr[b' = 1|b = 1] \geq \frac{1}{2} + \frac{\epsilon(\lambda)}{2} - (\frac{1}{q} + \frac{1}{2}) = \frac{\epsilon(\lambda)}{2} - \frac{1}{q}$$

, which is non-negligible. So we construct a PPT adversary Adv′ that distinguish $b = 0, 1$ with non-negligible probability, which contradicts Assumption 5.1. So such adversary Adv does not exist. □

In fact, the adversary also knows a, ch', z_1, z_2. However, according to the security of Okamoto's identifier scheme, (a, ch', z_1, z_2) do not provide any additional information about s_1, s_2 and cannot help the adversary to output (m', M'). In summary, the first case occurs with negligible probability.

In the second case, each message matches the secrets with probability $\frac{1}{q}$, so the probability that at least one of them matches the secrets is

$$1 - (1 - \frac{1}{q})^{n-1} < 1 - (1 - \frac{n-1}{q}) = \frac{n-1}{q}$$

, which is negligible. In conclusion, both cases occur with negligible probability, so the probability that the client cannot successfully rebut is also negligible.

doi:10.6342/NTU202301028

5.3.2 ”Only if” direction

If the servers do not modify a client's message (m, M), then the client should not be able to provide s_1, s_2 such that $id = g_1^{s_1} g_2^{s_2} \wedge M \neq m^{s_1} g_3^{s_2}$. More specifically, such $(id, (m, M))$ cannot pass the validation check with non-negligible probability. First we review Okamoto's protocol and protocol version 1:

- Okamoto's protocol

  1. The client randomly chooses $r_1, r_2 \xleftarrow{r} \mathbb{Z}_q$ and sends $a = g_1^{r_1} g_2^{r_2}$ to the servers.
  2. The servers choose $ch \xleftarrow{r} \mathbb{Z}_q$ and sends it to the client.
  3. The client computes $z_1 = ch \cdot s_1 + r_1, z_2 = ch \cdot s_2 + r_2$ and sends them to the servers.
  4. The servers check that $g_1^{z_1} g_2^{z_2} = id^{ch} a$.

- Protocol version 1

  1. The client randomly chooses $r_1, r_2 \xleftarrow{r} \mathbb{Z}_q$ and sends $a = g_1^{r_1} g_2^{r_2}, b = m^{r_1} g_3^{r_2}$ to the servers.
  2. The servers choose $ch \xleftarrow{r} \mathbb{Z}_q$ and sends it to the client.
  3. The client computes $z_1 = ch \cdot s_1 + r_1, z_2 = ch \cdot s_2 + r_2$ and sends them to the servers.
  4. The servers checks that $g_1^{z_1} g_2^{z_2} = id^{ch} a \wedge m^{z_1} g_3^{z_2} = M^{ch} b$.

Okamoto defined that a ”good” client (prover) is a client that follows Okamoto's protocol. Let T be the condition of a ”good” client: $id = g_1^{s_1} g_2^{s_2} \wedge a = g_1^{r_1} g_2^{r_2} \wedge z_1 =$

doi:10.6342/NTU202301028

$ch \cdot s_1 + r_1 \wedge z_2 = ch \cdot s_2 + r_2$. Okamoto proved that if a client is not "good", he cannot pass the check is with non-negligible probability, i.e., $\Pr[g_1^{z_1} g_2^{z_2} = id^{ch} a | \neg T]$ is negligible. Now we prove that in protocol version 1, a client cannot pass the check with non-negligible probability if $M \neq m^{s_1} g_3^{s_2}$.

Theorem 5.4. *Suppose that the servers behave properly and the challenge ch is uniformly chosen from $\mathbb{Z}_q$. There is no PPT adversary that can generate $M \neq m^{s_1} g_3^{s_2}$ and pass the check with non-negligible probability.*

Proof. Suppose that such adversary exists, i.e., there is a PPT adversary Adv and non-negligible function $\epsilon(\cdot)$ s.t. $\Pr[g_1^{z_1} g_2^{z_2} = id^{ch} a \wedge m^{z_1} g_3^{z_2} = M^{ch} b | M \neq m^{s_1} g_3^{s_2}] \geq \epsilon(\lambda)$. Then there is another adversary Adv′ that is not "good" but pass the check in Okamoto's protocol as follows:

1. Adv′ receives id, m, M, a, b from Adv and sends id, a to the challenger.
2. The challenger chooses $ch \xleftarrow{r} \mathbb{Z}_q$ and sends to Adv′.
3. Adv′ sends ch to Adv.
4. Adv′ receives z_1, z_2 from Adv and sends z_1, z_2 to the challenger.

Now we analyze the success probability for Adv. We have

$$
\begin{aligned}
\epsilon(\lambda) &\leq \Pr[g_1^{z_1} g_2^{z_2} = id^{ch} a \wedge m^{z_1} g_3^{z_2} = M^{ch} b | M \neq m^{s_1} g_3^{s_2}] \\
&= \Pr[g_1^{z_1} g_2^{z_2} = id^{ch} a \wedge m^{z_1} g_3^{z_2} = M^{ch} b | M \neq m^{s_1} g_3^{s_2} \wedge T] \cdot \Pr[M \neq m^{s_1} g_3^{s_2} \wedge T] \\
&\quad + \Pr[g_1^{z_1} g_2^{z_2} = id^{ch} a \wedge m^{z_1} g_3^{z_2} = M^{ch} b | M \neq m^{s_1} g_3^{s_2} \wedge \neg T] \cdot \Pr[M \neq m^{s_1} g_3^{s_2} \wedge \neg T] \\
&\leq \Pr[m^{z_1} g_3^{z_2} = M^{ch} b | M \neq m^{s_1} g_3^{s_2} \wedge T] + \Pr[g_1^{z_1} g_2^{z_2} = id^{ch} a | M \neq m^{s_1} g_3^{s_2} \wedge \neg T] \\
&= \Pr[m^{z_1} g_3^{z_2} = M^{ch} b | M \neq m^{s_1} g_3^{s_2} \wedge T] + \Pr[g_1^{z_1} g_2^{z_2} = id^{ch} a | \neg T]
\end{aligned}
$$

doi:10.6342/NTU202301028

First we consider $\Pr[m^{z_1} g_3^{z_2} = M^{ch} b | M \neq m^{s_1} g_3^{s_2} \wedge T]$.

$$\begin{aligned}
&\Pr[m^{z_1} g_3^{z_2} = M^{ch} b | M \neq m^{s_1} g_3^{s_2} \wedge T] \\
=&\Pr[m^{ch \cdot s_1 + r_1} g_3^{ch \cdot s_2 + r_2} = M^{ch} b | M \neq m^{s_1} g_3^{s_2}] \\
=&\Pr[(m^{s_1} g_3^{s_2} M^{-1})^{ch} = m^{-r_1} g_3^{-r_2} b | M \neq m^{s_1} g_3^{s_2}]
\end{aligned}$$

Since $m^{s_1} g_3^{s_2} M^{-1} \in \mathbb{G}$ and $m^{s_1} g_3^{s_2} M^{-1} \neq 1$, $m^{s_1} g_3^{s_2} M^{-1}$ has order q. Since b, r_1, r_2 are chosen before ch, we have $\Pr[(m^{s_1} g_3^{s_2} M^{-1})^{ch} = m^{-r_1} g_3^{-r_2} b] = \frac{1}{q}$. Then we can get

$$\Pr[g_1^{z_1} g_2^{z_2} = id^{ch} a | \neg T] \geq \epsilon(\lambda) - \frac{1}{q}$$

is non-negligible, which contradicts Okamoto's result. So such adversary Adv does not exist. □

The actual validation check is protocol version 4, which includes distributed Fiat-Shamir and secure multiparty computation to protocol version 1. However, a malicious client should not be able to gain non-negligible advantage from these transformations. Thus, the probability that the client can rebut is negligible.

doi:10.6342/NTU202301028

Chapter 6 Application: Electronic Voting

An application of CRAB is to build up a robust and efficient electronic voting system. In electronic voting, or e-voting, the voters submit their ballots to the servers, and the servers count the anonymous ballots after the voting period is over. To prevent malicious servers, the ballots must be protected, e.g., encrypted or secret-shared (or a mixture of both), and recovered (decrypted or reconstructed) later.

Many e-voting systems aggregate the protected ballots and recover the tally later [7, 18, 20, 27, 30, 35, 43]. This is usually achieved with homomophism, where the voters send the ballots as numbers, and the servers count the ballots in the protected manner with homomorphic operations. The servers then recover only the tally but not each individual ballot, so the privacy of each vote is preserved. The main problem with these systems is that the servers must check that the ballots are valid numbers. If a malformed ballot is accepted (accidentally or intentionally), the election result would be significantly affected.

Many other systems output a mix of the recovered ballots and tally them later. Anyone can verify the tally by counting the ballots on his own. This scenario is analogous to anonymous broadcast. Many e-voting systems [2, 14, 26, 29, 36, 44] also use mixnet, and they have the same problem that it is expensive to generate a zero-knowledge proof

doi:10.6342/NTU202301028

of shuffle. Therefore, we can use CRAB to build up an e-voting system with better performance.

6.1 The E-voting System

E-voting systems require eligibility and fairness: only eligible voters can vote, and each voter can vote only once. We assume that there is a *registrar* that can check the eligibility of each person. Before the voting period, the voters generate their secrets and identifiers, and send their identifiers to the registrar. The registrar would publish the identifiers of the eligible voters on a bulletin board.

During the voting period, each voter uses his vote and his secrets to generate the backdoor. His ballot (message) is the pair of the vote and the backdoor. Then he sends the identifier, the ballot and a zero-knowledge proof to the servers as described in Chapter 4.3.2. The servers them perform the validation check. Before they check the proof, they check if the identifier is on the bulletin board and if the identifier has not already been accepted. If all checks pass, the servers accept the vote and mark that the identifier is accepted.

After the voting period, the servers process the ballots using multiparty shuffle and publish a list of the shuffled ballots. Then the servers can count the ballots and publish the result. Voters can verify that their votes are included and that the result matches the published ballots.

doi:10.6342/NTU202301028

6.2 Security Analysis

It is clear that the e-voting system based on CRAB also achieve correctness and anonymity. Eligibility and fairness are also satisfied since that each identifier must be eligible and would be accept only once.

In addition to the above security requirements, verifiability is also an important security requirement. Verifiability allows the voters to verify that their votes have been counted correctly. It can be divided into *individual* and *universal* verifiability:

- **Individual verifiability**: Each voter can verify that his vote is included in the tally.
- **Universal verifiability**: Anyone can verify that all the votes are counted correctly and not modified.

CRAB is client-rebuttable, which allows a voter to verify that his vote is included or otherwise disprove it. Thus, CRAB also achieves individual verifiability. However, CRAB does not provide the integrity guarantee of others' messages, so the universal verifiability is not achieved. We leave the construction of a universally verifiable e-voting system without mixnet for future work.

doi:10.6342/NTU202301028

doi:10.6342/NTU202301028

Chapter 7 Implementation and Evaluation

We implement the CRAB system using Clarion as the multiparty shuffling protocol. Clarion is open source on [42], and our implementation is based on it. We also use goff [9] for fast field operation, and we choose $p = 2^{1017} + 422487$ for 127-bit messages (we will explain the choice of p in Chapter 7.1). We implement the parts with group operations, including the validation check and rebuttal check. However, we do not make much modification on the process part. We evaluate the performance of CRAB on the CSIE workstation. We do not compare our work with Clarion because the processing is quite similar.

7.1 Implementation on Group Elements

In Chapter 5, we prove the security of the CRAB system under Assumption 5.1. To make our assumptions consistent with the decisional Diffie-Hellman assumption, the public generators $g_1, g_2, g_3 \in \mathbb{G}$ should be randomly chosen from $\mathbb{G}$. A naive way to generate the random group elements is that we choose a generator g and random exponents $r_1, r_2, r_3 \xleftarrow{r} \mathbb{Z}_q^*$, and output $g_i = g^{r_i}, i = 1, 2, 3$. However, the party who knows r_1, r_2, r_3

doi:10.6342/NTU202301028

also knows the discrete logarithm relation between them, which violates our requirement in Chapter 3.1.

We propose a secure way to generate the group generator. First, we choose random elements in $\{2, \ldots, p-2\}$. For any $g \in \mathbb{Z}_p^*$, the following three statements are equivalent:

1. $g^q = 1$
2. $g \in \mathbb{G}$
3. $p - g \notin \mathbb{G}$

We can randomly choose $g_1', g_2', g_3' \in \{2, \ldots, p-2\}$ and determine if they belong to the group $\mathbb{G}$ by checking $(g_i')^q \equiv 1 \pmod{p}$. If $g_i' \in \mathbb{G}$, we have $g_i = g_i'$. Otherwise, we have $g_i = p - g_i'$.

We also require that the message m and the backdoor M belong to $\mathbb{G}$. However, it is unrealistic to restrict the client to send only the message that belongs to $\mathbb{G}$. Here is our workaround:

- We accept messages up to $\lambda - 2$ bits. Recall that p is a λ-bit prime number, which means that $p > 2^{\lambda-1}$.
- To broadcast a message, first we map the message to a number $m (< 2^{\lambda-2})$. If $m^q \neq 1$, the client sets $m \leftarrow p - m (> 2^{\lambda-2})$ and sends the message shares and the proof according to m.
- During the validation, the servers should also check if $m^q = 1 \wedge M^q = 1$.
- Finally, we convert each output (m_i, M_i) back into a message. If $m_i < 2^\ell$, we map the number m_i to a message. Otherwise, we map the number $(p - m_i)$.

doi:10.6342/NTU202301028

7.2 Complexity analysis

The communication cost between a client and a server is $O(1)$. For each server, the validation check of a message requires $O(k)$ multiparty computation, and the processing cost is $O(kn)$ for computation and communication. The total cost for a server is $O(kn)$. Finally, it takes the rebuttal authority $O(n)$ to verify a rebuttal.

7.3 Experiment Result

7.3.1 Client performance

The computation time for a client is about 50ms. This includes the time to generate the id, the backdoor, and the proof. In environments with lower computation power, such as mobile devices and browsers, the computation time may be longer. However, we believe it is within acceptable limits.

The total number of bytes that a client sends to a server is about 7 times than the length of the message. In our case, a client sends 864 bytes to each server to broadcast a 127-byte message.

7.3.2 Validation check

In CRAB and most DPF-based systems, the servers have to check the validity of each request. It is called ”format-verification” in Blinder [1] and ”audit” in Spectrum [38] and Sabre [46]. The validation check on the servers should be fairly fast. Otherwise, the clients may have to wait a long time to know if their messages are accepted. Moreover, if it takes

doi:10.6342/NTU202301028

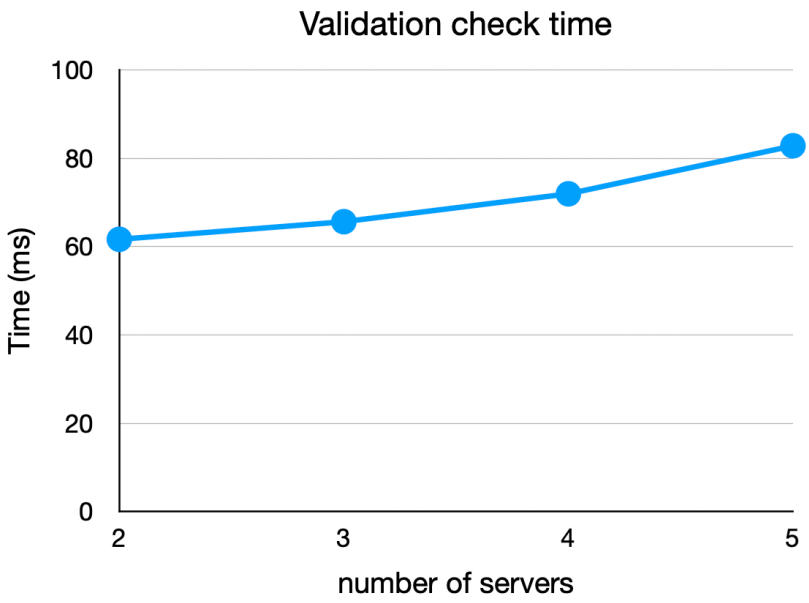


Figure 7.1: The time of validation for 2∼5 servers.

a lot of effort for the servers to validate each request, the adversary can launch a denial-of-service attack by continuing to send requests to the servers. Most of the DPF-based systems suffer from a long validation time that grows with the number of messages.

Figure 7.1 shows the validation check time for different numbers of servers. The validation check time increases slowly with the number of servers, and the time is less than 100 ms for 5 servers. Compared to DPF-based systems, Blinder [1] takes a few seconds while Spectrum [38] and Sabre [46] take a few hundred milliseconds for a total of 10^5 messages. Also, the time grows with the total number of messages in DPF-based systems, while the time is not affected in CRAB. Thus, CRAB has better scalability in the validation aspect.

7.3.3 Process

We compare the processing time of CRAB with the zero-knowledge proof of shuffle. The zero-knowledge proof of shuffle is required in most mixnet-based systems. The state-

doi:10.6342/NTU202301028

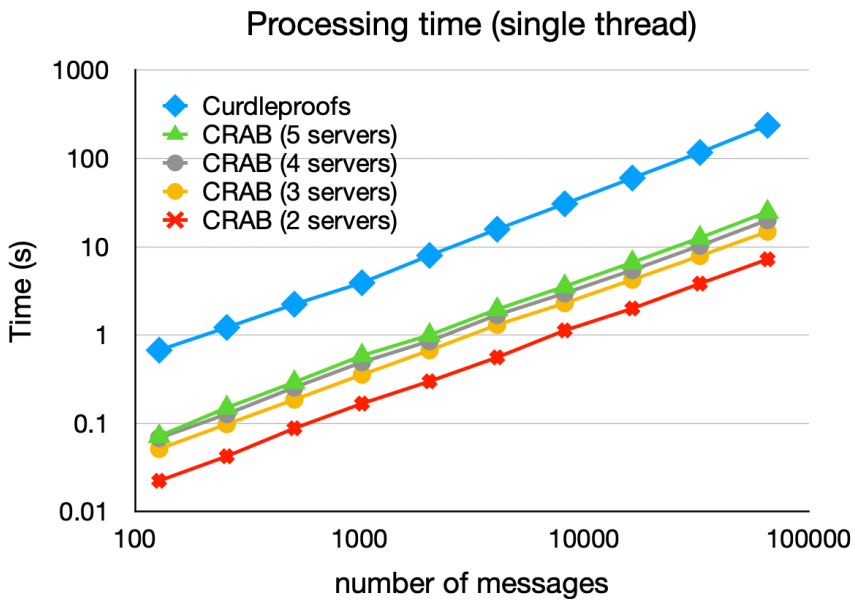


Figure 7.2: The time of processing for $10^2 \sim 10^5$ messages with single thread.

of-the-art zero-knowledge proof is proposed by Bayer and Groth [5], which has relatively lower proof time among the proofs with sublinear proof size. Some recent mixnet-based anonymous broadcast systems such as Stadium [45] and Atom [31] use this proof. The proof we compare to is Curdleproofs [4], which is inspired by Bayer and Groth's proof. Curdleproofs is recently proposed and implemented by the Ethereum research team, so it should be a reasonable comparison. The evaluated time of Curdleproofs includes the shuffle and the proof.

Figure 7.2 shows the performance of CRAB for different numbers of servers and Curdleproofs. For CRAB with 2 servers, the processing is $23 \sim 33\times$ faster than Curdleproofs. For CRAB with 5 servers, the processing is still $6.6 \sim 9.6\times$ faster than Curdleproofs.

We also show that the computation time of CRAB can be significantly reduced by parallel computing. We evaluate the performance of CRAB with 16 threads. In Figure 7.3, we can see that the estimated time for 5 servers and about 10^5 messages can be reduced to $1/4$. The time for 10^6 messages is only several tens of seconds.

doi:10.6342/NTU202301028

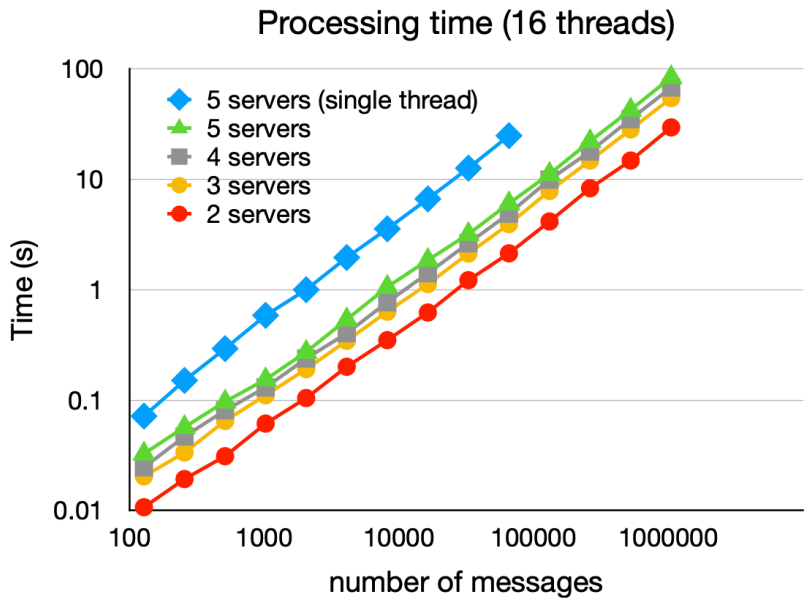


Figure 7.3: The time of processing for $10^2 \sim 10^6$ messages with 16 threads.

7.3.4 Rebuttal check

To check a rebuttal, the rebuttal authority has two steps. The first step is to check that the given identity is on the bulletin board and matches the given secrets. The second step is to iterate through all the messages and check that the given secrets do not match the messages. The first step is fast while the second step is a bit time-consuming. Fortunately, the rebuttal authority usually only need to run the second step once to disapprove the anonymous broadcast. Moreover, the second step can easily be parallelized.

Figure 7.4 shows the time of the second step of the rebuttal check with 16 threads. The rebuttal check time for 10^6 messages is about 440 seconds. Considering an election for a million people, it often takes several hours and a lot of money to verify the result. So the rebuttal check of CRAB is relatively practical.

doi:10.6342/NTU202301028

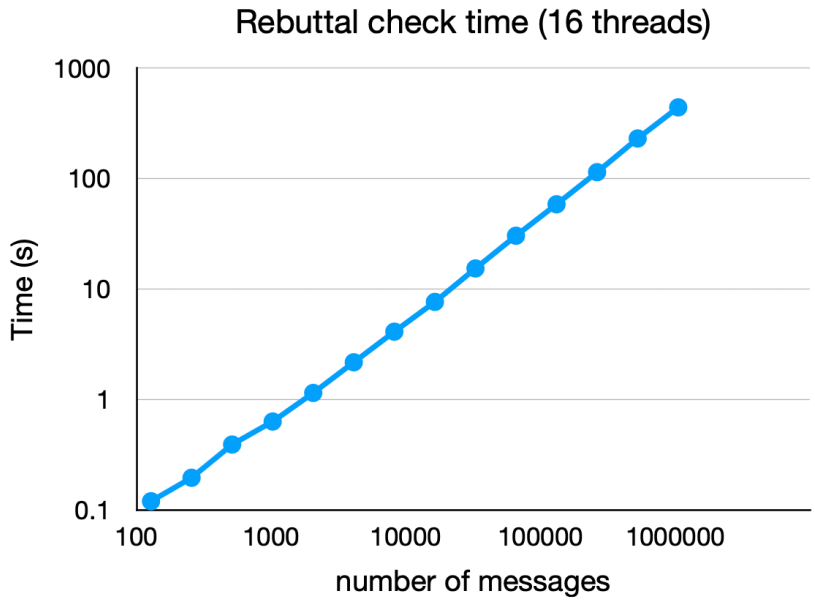


Figure 7.4: The time of rebuttal check for $10^2 \sim 10^6$ messages with 16 threads.

doi:10.6342/NTU202301028

doi:10.6342/NTU202301028

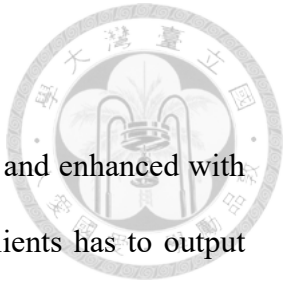
Chapter 8 Related Work

Many anonymous broadcast systems are based on mixnet and require a zero-knowledge proof of the shuffle. Many proofs have been proposed to for the correctness of a shuffle [5, 24, 28, 37]. Early works such as Furukawa and Sako [24] and Neff [37] proposed the proof with $O(n)$ proof time and space, where n is the number of the messages. However, the proof is too large to be useful. Bayer and Groth [5] improve the proof size to $O(\sqrt{n})$, so that the proof is much smaller than the data itself. Several works [4, 26, 31, 36, 45] use Bayer and Groth's proof in their systems. Bulletproofs [10] and Hoffman et al. [28] even improve the proof size to $O(\log n)$. However, the time required to generate the proof is still high, resulting in a high latency to output the result.

To reduce the latency, some mixnet-based anonymous broadcast systems [31, 32, 34, 45] perform parallel mix, where each server shuffles a portion of the messages in parallel. However, it requires several layers of parallel mix to achieve near-random permutation. Atom [31] suggests $O(\log^2 k)$ layers for their butterfly network [17]. Stadium [45] and Karaoke [34] use fewer layers of parallel mix, but they need to add noise messages and they only achieve a weaker version of anonymity. XRD [32] reduces the number of layers at the expense of client communication cost. Trellis [33] is a recent research that avoids the proof of shuffle by using a new technique called *boomerang encryption*. All of the above works do not need to trust any servers to ensure the correctness, but they require at

doi:10.6342/NTU202301028

least one honest server for anonymity.



Many other anonymous broadcast systems are based on DC-net and enhanced with DPF [1, 16, 38, 46]. In DC-net, for a client to broadcast, all the clients has to output a request. For every client to broadcast a message, each client has to output request of length $O(n)$, so the validation check for each request takes at least $O(n)$. Riposte [16] reduces the request length to $O(\sqrt{n})$ by using DPF [25], but it requires more trust in the servers to achieve $O(\sqrt{n})$ validation check time. Blinder [1] is a more robust system with $O(\sqrt{n})$ request size and check time. Spectrum [38] uses a MAC on the request to solve the collision problem and only requires that there is at least one honest server. Sabre [46] uses the more complicated DPF to reduce the request size to $O(\log n)$, and constructs a secret-shared audit protocol to make the validation time also $O(\log n)$. Overall, the request size and validation time grow with n, thus the scalability is bad. Also, the clients need to trust some honest servers to ensure the integrity of the messages.

Finally, Clarion [23] uses a special MPC protocol to perform the shuffle. As stated in Clarion's paper, Clarion sits between mixnet approaches and DC-net approaches. Mixnet approaches do not require any additional information or validation check for a request, but they impose high cost on the generation of the zero-knowledge proof. DC-net and related approaches, on the other hand, require additional information and validation check for each request, but the output can be generated at low cost. Clarion strikes a balance between the two. It requires less information and more efficient validation check compared to DC-net, and runs a lighter shuffling protocol compared to mixnet. However, Clarion requires at least one honest server to ensure the correctness. Our work, CRAB, reduces this requirement with little additional information and validation check. Table 8.1 presents a comparison of the recent anonymous broadcast systems.

doi:10.6342/NTU202301028

Table 8.1: Comparison of related work of anonymous broadcast

	Verifiability (no trust on servers)	Scalability		Latency	Shuffle mechanism
		Request size	Validation check		
Atom [31]	✓	$O(1)$	N/A	high	Mixnet
Stadium [45]	✓	$O(1)$	N/A	high	Mixnet
Karaoke [34]	✓	$O(1)$	N/A	high	Mixnet
XRD [32]	✓	$O(\sqrt{k})$	N/A	high	Mixnet
Trellis [33]	✓	$O(\log n)$	N/A	low	Mixnet
Riposte [16]	✗	$O(k\sqrt{n})$	$O(\sqrt{n})$	low	DPF
Blinder [1]	✗	$O(k\sqrt{n})$	$O(\sqrt{n})$	low	DPF
Spectrum [38]	✗	$O(k\sqrt{n})$	$O(\sqrt{n})$	low	DPF
Sabre [46]	✗	$O(k \log n)$	$O(\log n)$	low	DPF
Clarion [23]	✗	$O(k)$	$O(1)$	low	Multiparty shuffle
CRAB	✓	$O(k)$	$O(1)$	low	Multiparty shuffle

doi:10.6342/NTU202301028

doi:10.6342/NTU202301028

Chapter 9 Conclusion

This thesis presents CRAB, an anonymous broadcast system that achieves provable anonymity and client-rebuttability. We use Clarion as the underlying multiparty shuffle protocol, and we improve verifiability for clients and reduce trust on servers. We also show that CRAB has low latency and good scalability, and it is suitable for large-scale electronic voting. The limitation of CRAB is that it requires heavy preprocess for multi-party computation, so it may not be suitable for the applications that are frequently used.

doi:10.6342/NTU202301028

doi:10.6342/NTU202301028

References

[1] I. Abraham, B. Pinkas, and A. Yanai. Blinder–scalable, robust anonymous committed broadcast. In Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, pages 1233–1252, 2020.

[2] B. Adida. Helios: Web-based open-audit voting. In USENIX security symposium, volume 17, pages 335–348, 2008.

[3] A. Aly, A. Abidin, and S. Nikova. Practically efficient secure distributed exponentiation without bit-decomposition. In Financial Cryptography and Data Security: 22nd International Conference, FC 2018, Nieuwpoort, Curaçao, February 26–March 2, 2018, Revised Selected Papers 22, pages 291–309. Springer, 2018.

[4] asn d6. Curdleproofs, 2022.

[5] S. Bayer and J. Groth. Efficient zero-knowledge argument for correctness of a shuffle. In Annual International Conference on the Theory and Applications of Cryptographic Techniques, pages 263–280. Springer, 2012.

[6] D. Beaver. Efficient multiparty protocols using circuit randomization. In Advances in Cryptology—CRYPTO' 91: Proceedings 11, pages 420–432. Springer, 1992.

[7] J. Benaloh and D. Tuinstra. Receipt-free secret-ballot elections. In Proceedings of

doi:10.6342/NTU202301028

the twenty-sixth annual ACM symposium on Theory of computing, pages 544–553, 1994.

[8] D. Boneh, E. Boyle, H. Corrigan-Gibbs, N. Gilboa, and Y. Ishai. Zero-knowledge proofs on secret-shared data via fully linear pcps. In Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part III, pages 67–97. Springer, 2019.

[9] G. Botrel, T. Piellard, Y. E. Housni, A. Tabaie, G. Gutoski, and I. Kubjas. Consensys/ gnark-crypto: v0.9.0, Jan. 2023.

[10] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In 2018 IEEE symposium on security and privacy (SP), pages 315–334. IEEE, 2018.

[11] M. Chase, E. Ghosh, and O. Poburinnaya. Secret-shared shuffle. In Advances in Cryptology–ASIACRYPT 2020: 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7–11, 2020, Proceedings, Part III 26, pages 342–372. Springer, 2020.

[12] D. Chaum. The dining cryptographers problem. J. cryptology, 1:65–75, 1988.

[13] D. L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. Communications of the ACM, 24(2):84–90, 1981.

[14] M. R. Clarkson, S. Chong, and A. C. Myers. Civitas: Toward a secure voting system. In 2008 IEEE Symposium on Security and Privacy (sp 2008), pages 354–368. IEEE, 2008.

doi:10.6342/NTU202301028

[15] H. Corrigan-Gibbs and D. Boneh. Prio: Private, robust, and scalable computation of aggregate statistics. In NSDI, pages 259–282, 2017.

[16] H. Corrigan-Gibbs, D. Boneh, and D. Mazières. Riposte: An anonymous messaging system handling millions of users. In 2015 IEEE Symposium on Security and Privacy, pages 321–338. IEEE, 2015.

[17] A. Czumaj and B. Vöcking. Thorp shuffling, butterflies, and non-markovian couplings. In Automata, Languages, and Programming: 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I 41, pages 344–355. Springer, 2014.

[18] G. G. Dagher, P. B. Marella, M. Milojkovic, and J. Mohler. Broncovote: Secure voting system using ethereum' s blockchain. 2018.

[19] I. Damgård, M. Fitzi, E. Kiltz, J. B. Nielsen, and T. Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In Theory of Cryptography: Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006. Proceedings 3, pages 285–304. Springer, 2006.

[20] I. Damgård and M. Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In Public Key Cryptography: 4th International Workshop on Practice and Theory in Public Key Cryptosystems, PKC 2001 Cheju Island, Korea, February 13–15, 2001 Proceedings 4, pages 119–136. Springer, 2001.

[21] I. Damgård and J. B. Nielsen. Scalable and unconditionally secure multiparty computation. In Advances in Cryptology-CRYPTO 2007: 27th Annual International

doi:10.6342/NTU202301028

Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007. Proceedings 27, pages 572–590. Springer, 2007.

[22] I. Damgård, V. Pastro, N. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In Advances in Cryptology–CRYPTO 2012: 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings, pages 643–662. Springer, 2012.

[23] S. Eskandarian and D. Boneh. Clarion: Anonymous communication from multiparty shuffling protocols. Cryptology ePrint Archive, 2021.

[24] J. Furukawa and K. Sako. An efficient scheme for proving a shuffle. In Advances in Cryptology—CRYPTO 2001: 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19–23, 2001 Proceedings 21, pages 368–387. Springer, 2001.

[25] N. Gilboa and Y. Ishai. Distributed point functions and their applications. In Advances in Cryptology–EUROCRYPT 2014: 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings 33, pages 640–658. Springer, 2014.

[26] P. Grontas, A. Pagourtzis, A. Zacharakis, and B. Zhang. Towards everlasting privacy and efficient coercion resistance in remote electronic voting. In Financial Cryptography and Data Security: FC 2018 International Workshops, BITCOIN, VOTING, and WTSC, Nieuwpoort, Curaçao, March 2, 2018, Revised Selected Papers 22, pages 210–231. Springer, 2019.

[27] M. Hirt and K. Sako. Efficient receipt-free voting based on homomorphic encryption.

doi:10.6342/NTU202301028

In Advances in Cryptology—EUROCRYPT 2000: International Conference on the Theory and Application of Cryptographic Techniques Bruges, Belgium, May 14–18, 2000 Proceedings, pages 539–556. Springer, 2000.

[28] M. Hoffmann, M. Klooß, and A. Rupp. Efficient zero-knowledge arguments in the discrete log setting, revisited. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, pages 2093–2110, 2019.

[29] A. Juels, D. Catalano, and M. Jakobsson. Coercion-resistant electronic elections. In Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society, pages 61–70, 2005.

[30] R. Küsters, J. Liedtke, J. Müller, D. Rausch, and A. Vogt. Ordinos: a verifiable tally-hiding e-voting system. In 2020 IEEE European Symposium on Security and Privacy (EuroS&P), pages 216–235. IEEE, 2020.

[31] A. Kwon, H. Corrigan-Gibbs, S. Devadas, and B. Ford. Atom: Horizontally scaling strong anonymity. In Proceedings of the 26th Symposium on Operating Systems Principles, pages 406–422, 2017.

[32] A. Kwon, D. Lu, and S. Devadas. Xrd: Scalable messaging system with cryptographic privacy. arXiv preprint arXiv:1901.04368, 2019.

[33] S. Langowski, S. Servan-Schreiber, and S. Devadas. Trellis: Robust and scalable metadata-private anonymous broadcast. Cryptology ePrint Archive, 2022.

[34] D. Lazar, Y. Gilad, and N. Zeldovich. Karaoke: Distributed private messaging immune to passive traffic analysis. In 13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18), pages 711–725, 2018.

doi:10.6342/NTU202301028

[35] Y. Liu and Q. Zhao. E-voting scheme using secret sharing and k-anonymity. World Wide Web, 22:1657–1667, 2019.

[36] W. Lueks, I. Querejeta-Azurmendi, and C. Troncoso. Voteagain: A scalable coercion-resistant voting system. arXiv preprint arXiv:2005.11189, 2020.

[37] C. A. 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, 2001.

[38] Z. Newman, S. Servan-Schreiber, and S. Devadas. Spectrum: High-bandwidth anonymous broadcast. In 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22), pages 229–248, 2022.

[39] C. Ning and Q. Xu. Constant-rounds, linear multi-party computation for exponentiation and modulo reduction with perfect security. In Advances in Cryptology–ASIACRYPT 2011: 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings 17, pages 572–589. Springer, 2011.

[40] T. Okamoto. Provably secure and practical identification schemes and corresponding signature schemes. In Crypto, volume 92, pages 31–53. Springer, 1992.

[41] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Advances in Cryptology—CRYPTO' 91: Proceedings, pages 129–140. Springer, 2001.

[42] SabaEskandarian. Clarion, 2021.

[43] B. Schoenmakers. A simple publicly verifiable secret sharing scheme and its

doi:10.6342/NTU202301028

application to electronic voting. In Advances in Cryptology—CRYPTO' 99: 19th Annual International Cryptology Conference Santa Barbara, California, USA, August 15–19, 1999 Proceedings, pages 148–164. Springer, 1999.

[44] O. Spycher, R. Koenig, R. Haenni, and M. Schläpfer. A new approach towards coercion-resistant remote e-voting in linear time. In Financial Cryptography and Data Security: 15th International Conference, FC 2011, Gros Islet, St. Lucia, February 28-March 4, 2011, Revised Selected Papers 15, pages 182–189. Springer, 2012.

[45] N. Tyagi, Y. Gilad, D. Leung, M. Zaharia, and N. Zeldovich. Stadium: A distributed metadata-private messaging system. In Proceedings of the 26th Symposium on Operating Systems Principles, pages 423–440, 2017.

[46] A. Vadapalli, K. Storrier, and R. Henry. Sabre: Sender-anonymous messaging with fast audits. In 2022 IEEE Symposium on Security and Privacy (SP), pages 1953–1970. IEEE, 2022.

[47] M. N. Wegman and J. L. Carter. New hash functions and their use in authentication and set equality. Journal of computer and system sciences, 22(3):265–279, 1981.

[48] K. Yang and X. Wang. Non-interactive zero-knowledge proofs to multiple verifiers. In Advances in Cryptology–ASIACRYPT 2022: 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5–9, 2022, Proceedings, Part III, pages 517–546. Springer, 2023.

doi:10.6342/NTU202301028