# SecretRoom: An Anonymous Chat Client

Cristian Gutu & Fengyao Ding

January 16, 2016

**Abstract**

While many people would like to be able to communicate anonymously, the few existing anonymous communication systems sacrifice anonymity for performance, or vice-versa. The most popular such app is Tor, which relies on a series of relays to protect anonymity [17]. Though proven to be efficient, Tor does not guarantee anonymity in the presence of strong adversaries like ISPs and government agencies who can conduct in-depth traffic analysis.

In contrast, our messaging application, SecretRoom, implements an improved version of a secure messaging protocol called Dining Cryptographers Networks (DC-Nets) to guarantee true anonymity in moderately sized groups [11]. However, unlike traditional DC-Nets, SecretRoom does not require direct communication between all participants and does not depend on the presence of honest clients for anonymity. By introducing an untrusted server that performs the DC-Net protocol on behalf of the clients, SecretRoom manages to reduce the $O(n^2)$ communication associated with traditional DC-Nets to $O(n)$ for $n$ clients. Moreover, by introducing artificially intelligent clients, SecretRoom makes the anonymity set size independent of the number of "real" clients. Ultimately SecretRoom reduces the communication to $O(n)$ and allows the DC-Net protocol to scale to hundreds of clients compared to a few tens of clients in traditional DC-Nets.

## 1 Introduction

Freedom of speech is a fundamental human right. In particular, anonymity is vital to the protection of true freedom of speech as it prevents fear of retaliation or punishment for one's words by adversaries such as former abusers, corporations, and governments [7, 5]. Anonymity allows people to escape oppression and judgment from those with more power. It lets victims of rape and abuse to speak freely about their experiences and employees to reveal unjust working conditions. It gives people called whistleblowers the ability to reveal scheming and corruption happening behind the scenes of the government. Whistleblowers need anonymity even if their cause is noble, for if the malicious government suspected a whistleblower of trying to denounce internal workings, the government could immediately prosecute the whistleblower with minimal regard as to the actual justice of the situation.

For example, a now-famous whistleblower by the name of Edward Snowden leaked several classified NSA documents in 2013 to journalists in Hong Kong, exposing an ongoing secret NSA program [4]. NSA had been spying on many people without consent or the proper authorization; it was only upon Snowden's exposure that the public became aware of NSA's illegal practices. However, in doing so, Snowden violated his contract and was persecuted, even though he was exposing the government's wrongdoings. To protect the rights of those who want to identify and reveal clandestine government corruption, we need means for anonymous communication.

Despite the importance of anonymity, the existing anonymous communication systems all have efficiency or security issues. Tor, for instance, provides anonymity with minimal performance overhead, but fails to provide anonymity in the face of a very powerful adversary [9]. On the other hand, cryptographic primitive like DC-Nets [11] allow for information theoretically secure anonymity, but sacrifice performance. In particular, DC-Nets suffer from a communication complexity that is quadratic in the number of persons communicating, and only allows for one person to talk at a time. In contrast, our product SecretRoom is a chat service that strikes a balance between security and efficiency. On the surface, SecretRoom has multiple chat rooms each corresponding to a different subject of interest. Behind the scenes, SecretRoom uses DC-Net's core concept of a broadcast channel to guarantee true anonymity, prevents DC-Net collisions by synchronizing clients using a secure schedule derivation system, and spawns client artificial intelligences (client AIs) to keep the anonymity set size independent from the number of real clients. By improving efficiency of DC-Nets without sacrificing anonymity, SecretRoom aims to make large-scale anonymous communication more reliable and practical.
Ultimately SecretRoom makes the following contributions:

1. Improves DC-Net communication complexity from $O(n^2)$ to $O(n)$ ($n$ = number of clients).
2. Implements a chat system that supports real time chatting for hundreds of clients.

3. Allows clients to join or leave at any time.
4. Introduces client AIs to maintain anonymity set independent of the number of real clients.

## 2 Background and Related Work

In this section we describe related work by focusing on trade-offs made by existing anonymity systems. The most widespread anonymous communication system is Tor. Tor allows its clients to hide their presence on websites, access otherwise restricted Internet content, anonymously publish websites, and use chat rooms to discuss sensitive topics [9]. Tor, however, is insufficient as it fails to guarantee complete anonymity for its clients. While Tor can hide the location from which each message is sent by distributing its clients' information over multiple locations on the Internet, people can still find out the identity of the sender of each message through other forms of metadata - that is, information about when, where, and to whom messages are sent. Namely, while Tor is fast enough for web browsing, it cannot hide when messages are sent nor when they arrive at a destination [9]. Thus, if an attacker is watching both sides of the communication, he can deduce which messages are sent by which client by observing the timing of the messages, making Tor an incomprehensive system against privacy invasions by powerful adversaries such as the US government.

In contrast, DC-Nets perform a secure multi-party computation of the boolean-OR function to guarantee anonymity even in the face of powerful adversaries as long as there exists at least one honest participant [11]. To achieve such strong anonymity, DC-Nets make all client activity indistinguishable by requiring communication between every client in order to broadcast a message from a single client. As a result, an attacker who is observing the network cannot target one specific client as all clients are seemingly doing the same thing. Ultimately, DC-Nets guarantee anonymity through the execution of the following 4 steps:

1. Establish pairwise secrets between clients.
2. Client with a message encodes his message into ASCII, XORs it with the pairwise secrets, and shares his result. (This is known as a "non-zero XOR" message.)
3. Client without a message XORs 0 with the pairwise secrets and shares his result.
4. Clients XOR all results to derive the intended message.

Having only one client send a non-zero XOR message at a time is what allows the final XOR computation to yield the message sender's intended message. The anonymity of DC-Nets is analyzed in [11]. Intuitively, if the pairwise secrets are random, then the adversary cannot learn the real message as the message is XORed together with random numbers.

Unfortunately, DC-Nets have their shortcomings. Because DC-Nets require communication between *every* client to broadcast a message from a single client, DC-Nets grow increasingly inefficient as more clients join. As a result, systems that build on the traditional DC-Net protocol

have an efficiency of $O(n^2)$ — which is unreasonable for large-scale communication [11]. In addition, DC-Net communication can be disrupted by malicious clients who, rather than sending the result of their XOR computation, just sends random bits, rendering the final XOR computation useless. Finally, only one client can send a non-zero XOR message at a time due to XOR collisions. If more than one client XORs a message in their XOR result, then the final XOR computation again renders a meaningless result.

### *2 .1 Establishing Pairwise Secrets*

Because many anonymous communication protocols, including SecretRoom, depend on secure pairwise secrets in order to XOR and authenticate various forms of data, they all need to implement a protocol which is cryptographically designed to do this. One such protocol is the Diffie-Hellman (DH) key exchange protocol which SecretRoom also implements [10]. The protocol works as follows between two parties, Alice and Bob:

1. Alice and Bob agree to use a modulus $p$ and base $g$.
2. Alice chooses a secret integer $a$ and sends Bob $A = g^a \bmod p$.
3. Bob chooses a secret integer $b$ and sends Alice $B = g^b \bmod p$.
4. Alice and Bob compute $s = A^b \bmod p = B^a \bmod p = g^{ab} \bmod p$, and $s$ is the shared secret.

In our protocol we consider *public data* to be DH's ($g^a \bmod p$) and ($g^b \bmod p$) because it is openly transmitted over the network without exposing the final pairwise secret. Also, we consider private data to be each client's *private numbers* ($a$ and $b$) because their values allow the public data to be computed into a pairwise secret.

## 3 Models

In this section we present the system and threat model of SecretRoom.

### *3.1 System Model and Goals*

SecretRoom is a centralized application that is based on real clients, client AIs ("bots" or programmed clients that imitate behaviour of real clients), and a server as illustrated in Figure 1. The server acts as an untrusted mediator who is responsible for collecting, computing, and distributing all information necessary for communication, removing the need for each client to directly communicate with every other client. The introduction of a server greatly reduces the communication overhead. At the same time, we do not leak any private information to the server. SecretRoom's server

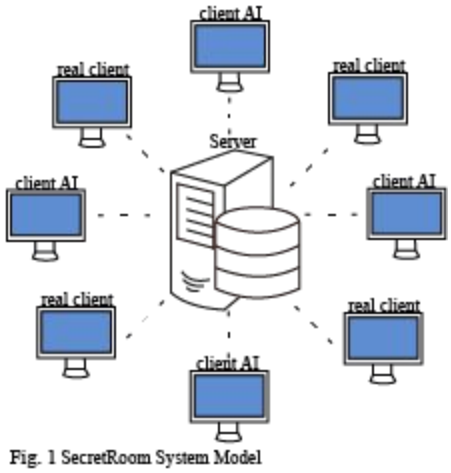

Fig. 1 SecretRoom System Model

contains several different chat rooms classified into different topics such as “Politics” and “School.” When clients load the SecretRoom website, they join a chat room per their topic of interest. In that specific chat room, the real clients and the client AIs form the anonymity set.

### *3.2 Threat Model*

SecretRoom follows a very similar threat model as DC-Nets [11]. Specifically, SecretRoom guarantees anonymity of all honest clients in a chat room. That is, if in a chat room of $n$ clients, $k$ clients as well as the server are malicious, the $n$-$k$ honest clients will remain completely anonymous as long as $k < n$-2.

## 4 Protocol

In this section we describe SecretRoom’s protocol and summarize the improvements over traditional DC-Nets.

### *4.1 Protocol Overview*

SecretRoom protocol consists of many rounds called *chat rounds*. The clients may join or leave the chat room at any point, even during a chat round. Each chat round corresponds to a client sending a message. Clients can start participating in chats the chat round after they join the chat room. Each chat round consists of the following steps:

1. If a new client joined or left during an *ongoing* chat round the clients share new pairwise secrets through the server (Section 4.2). They concurrently build a communication schedule to prevent DC-Net collisions (Section 4.3), and the server sends a *schedule turn counter* to all clients. The turn counter indicates whose turn it is to send a non-zero ASCII encoded message.
2. Each client XORs their pairwise secrets either with their ASCII encoded message or with 0 depending on whether it is their turn or not, sends the result of the XOR to the server, and notifies the server that they are ready for the next chat round.
3. Once all clients in a chat room are ready for the next chat round, the server derives the intended message by XORing the received XOR messages (step 4 of the DC-Net protocol) and broadcasts the final message back to all clients along with an incremented schedule turn counter.
4. To prevent re-executing the DH protocol to build new pairwise keys, all clients use their pairwise secrets as generation seeds for a pseudorandom generator [19] that allows them to compute new secrets without any network communication.

### *4.2 Reducing Efficiency from $O(n^2)$ to $O(n)$*

On a high level, our protocol reduces the DC-Net efficiency from the traditional $O(n^2)$ to $O(n)$ because, by including a server, clients no longer have to communicate with each other to obtain the pairwise secrets and the schedule. That is, in a chat of $n$ clients, each client only has to talk to one server to become part of the chat, whereas before, a client would need to talk to $n$-1 clients [11]. For example, a client that wants to join a chat of 100 clients would only need to contact the server to obtain the required data rather than 100 clients like in traditional DC-Nets. Thus our protocol reduces 99% of these network calls.

Our protocol is able to make such improvements by requiring the clients to upload the DH *public data* (Section 2.1) to the server. This data can then be downloaded by the clients to build the secure *private data.* So, by pre-establishing a local secret number, ($a$) DH's prime number ($p$) and the primitive root modulo $p$ ($g$) clients can compute pairwise secrets by uploading their public number ($g^a \bmod p$) to the server and downloading everybody else's public numbers. Having all other client's public numbers ($B_1 ... B_n$) a client can compute $B_n^a \bmod p$ to generate a different pairwise secret with each other client in the chat room. These pairwise secrets are later used in the XOR computations that are necessary for sending client messages, as described by steps 2 and 3 in the DC-Net protocol [11]. Ultimately, by polling the server for the public data, clients who use our protocol can perform the driving factor of DC-Nets (building the pairwise keys) without needing to communicate with *every* other client.

Additionally, our protocol requires clients to send their XOR computation of the pairwise secrets only to the server. This is in contrast to the traditional DC-Net protocol in which each client would need to send their XOR computation of the pairwise secrets to every other client in order for them to be able to derive the message. Because in our protocol only the server needs to receive everybody's XOR results, each client only has to talk to one server instead of every other client to send and receive messages, thus also reducing the message communication efficiency from $O(n^2)$ to $O(n)$.

### *4.3 Preventing DC-Net Collisions*

DC-Nets are powerful because they hide client's actions and thus make them anonymous [11]. While this is a desirable quality, the ability to hide client's actions alone does not make a system functional. Namely, although traditional DC-Nets keep clients anonymous, the protocol becomes useless when more than one client XORs a non-zero message into their final XOR result. When this happens, the final XOR yields a meaningless result and, for all practical purposes, renders the system useless as clients can't get anything out of it.

Our protocol prevents this downfall by synchronizing all clients using a *secure schedule* built through our centralized server: though all clients have a copy of the schedule, they only learn their own turns.

We currently have two ways to building this schedule in our protocol. In the implemented version, our protocol builds a schedule by making every client in the chat room build a pairwise secret *with the server* (using DH). Once every client does this, the server (and only the server) has a list of pairwise secrets (i.e., random numbers) with each client. The server sorts the list of secrets and broadcasts this list back to all clients as the schedule. Having the schedule, clients can identify when it is their turn based on the index of their pairwise secret in the list in relation to the turn counter, which is also provided by the server. We note that no client learns anything about any other clients' turn numbers, as each secret in the schedule is built only *with the server*.

However, our implemented version of schedule-building is not secure in the event of a malicious server, as each client's position on the schedule is represented exactly by the pairwise secret that it shares with the server. Using the pairwise secrets and the turn counter, the server can learn whose turn it currently is. There has been previous work that solves this problem [12]. [12] uses a secure shuffling only among the clients to ensure no single party learns the entire schedule, and we can add the shuffling method to the SecretRoom protocol to make the schedule-building secure. However, while the schedule-building implemented by [12] is secure, it is slow. On the other hand, the schedule-building currently used by SecretRoom is insecure but fast. Thus, we provide a mechanism for trading off security and efficiency, depending on the clients' willingness to trust the server.

### *4.4 Reusing Common Public Data*

Notice how building the schedule in Section 4.3 is similar to building the pairwise keys explained in Section 4.2. Namely, both processes use the Diffie Hellman protocol to generate pairwise secrets between parties. To remove the redundancy and inefficiency related with this, our protocol requires the centralized server to reuse the common public numbers ($g^a \ mod \ p$) associated with these two processes. As a result, we alleviate 50% of the network calls associated with building the pairwise keys and creating the secure schedule.

### *4.5 Client AIs*

Client AIs follow the same exact program flow as regular clients. The only real difference between client AIs and real clients is the way they are spawned. Namely, some number of client AIs are spawned automatically when we start the server. The client AIs increase the anonymity set of the real clients: since the activities of AIs and real clients are indistinguishable, attackers

cannot link a message to a particular small group of clients in the case that the number of real clients is small. In order to make the *contents* of the messages of client AI indistinguishable from real client activity, we can use one of the many available chatbots [18] who send more realistic messages related to the topic of the chat room. Thus, even if a chat room has no real clients at a given moment in time, the chat bots will send messages that resemble human thought, making a real client who joins the chat room later less suspicious.

## 5 Implementation Technologies

In this section we list the technologies used, how they were used in our project, and why they are significant.

Total List of Technologies Used:

- TypeScript
- Node.js
- Socket.IO
- Express.js
- gulp.js (including gulp-uglify)
- bower.js
- bootstrap
- HTML
- jquery
- Microsoft Azure

### *5.1 TypeScript, Node.js, and Express.js*

The SecretRoom centralized server which manipulates and distributes all the public data is developed using Microsoft's TypeScript and node.js. Node.js "is a platform built on Chrome's JavaScript runtime" that allows non-blocking JavaScript server side programming [13]. TypeScript allows the use of javascript ES6 features that mirror functionalities of popular Object Oriented languages such as C#, and Java [8]. Through the use of TypeScript and node.js, the SecretRoom server is organized into classes and interfaces that allow inheritance and thus the reuse of common code from parent classes. Because of this modular design, SecretRoom can easily be adapted to add new features or protocols. SecretRoom uses Express (a node.js web framework) to handle web requests and send html to clients [3].

### *5.2 Socket.IO*

To transmit client/server data SecretRoom employs a technology known as Socket.IO. Through the use of modern day web sockets Socket.IO "enables real-time bidirectional event-based communication" [16]. Due to this SecretRoom is a fully asynchronous application than can simultaneously handle client requests and compute data without breaking any of the driving protocols. It is important to note that if the server detects a client which does not support modern day web sockets it falls back to xhr-polling which, in essence, uses traditional GET and POST request to send and receive data.

### *5.3 gulp.js (including gulp-uglify)*

Since TypeScript with node.js is known to have some file referencing issues, SecretRoom also uses a library known as gulp.js to concatenate related classes in one large file which logically alleviates the file referencing issues when trying to import classes in TypeScript [1]. Moreover, to reduce the code file size even more SecretRoom also uses gulp-uglify to remove any unnecessary spaces. This process essentially makes the code file a long, unreadable, concatenated string that simply contains the code necessary to run the application.

### *5.4 bower.js*

For the web development aspect, SecretRoom uses another library known as bower.js which is a package manager that fetches, installs, and saves many other web dependencies that are required in our application [2]. This automation process allows for a more manageable workspace and ensures the most up to date versions of the technologies we use. In our application, bower takes care of installing jquery, bootstrap, and the client side version of Socket.IO.

### *5.5 Bootstrap*

Bootstrap is a HTML, CSS, and JS framework for developing responsive websites. Specifically, this means that no matter the client's screen resolution SecretRoom will always dynamically adapt to fit the screen [15].

### *5.6 jquery*

Because SecretRoom needs to manipulate the client's UI as they join and leave chatrooms SecretRoom uses jquery which is a javascript library that makes DOM manipulation less redundant [14].

### *5.7 Microsoft Azure*

SecretRoom is hosted on Microsoft's Azure. Azure is a "cloud computing platform with a growing collection of services, analytics, computing, database, mobile, networking, and storage" [6].

## 6 Results

In this section we explain our experiments, put them into context and analyze their significance. In particular, we focus on the latency of the communication, which is the time it takes for a client to send and receive a non-zero XOR message. This measures the usability and effectiveness of the SecretRoom chat protocol. Observe Figure 2, which is a representation of the final product.

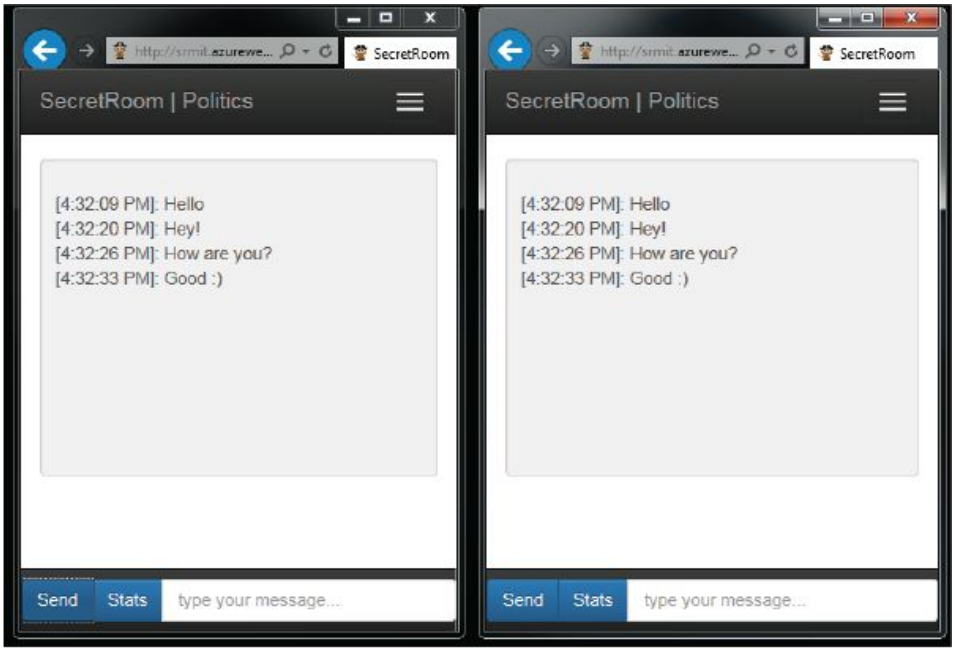


Fig 2. SecretRoom user interface. This is an example of two different clients.

### *6.1 Experiments*

We obtained our results by running all client AIs on one machine and all real clients on different machines around the world. Namely, for every trial we had 4 real clients from 2 different locations in Connecticut, US, 2 in London, UK, and 2 in Chisinau, Moldova. Ultimately, the majority of client in every trial were AIs. This was due to lack of resource — we did not have enough computers to test a chat room with real clients as the majority. Note, however, that even though client AIs were the majority, the validity of our results is not affected in any way as the client AIs performed the same exact computations as the real clients.

Additionally, throughout this experiment we had three separate chat rooms. We used these chat rooms to test the rebuilding of data when clients left and entered different chat rooms during *ongoing* chat rounds.

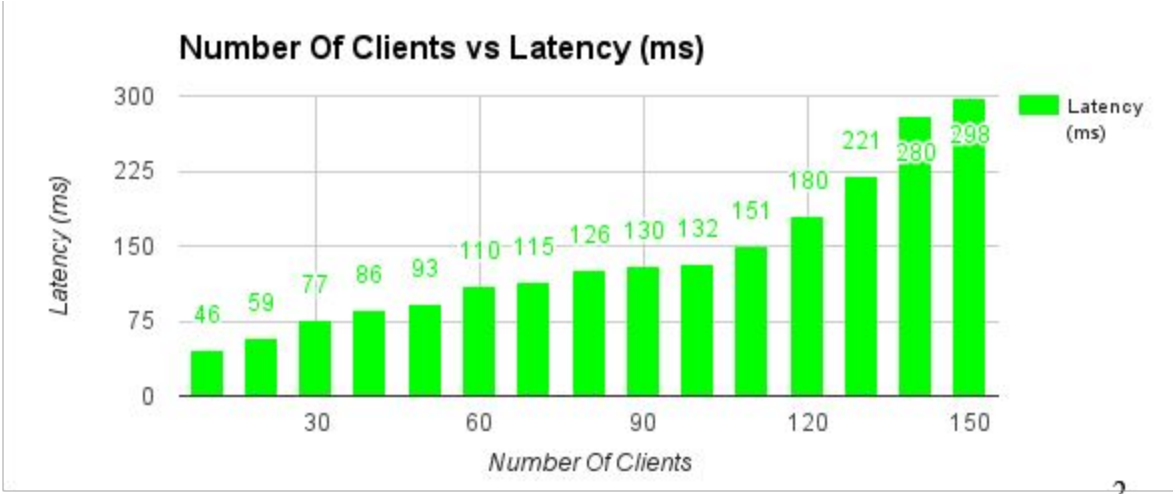


Fig 3. Even with 150 clients and constant messaging latency (ms) is not even close to being $O(n^2)$.

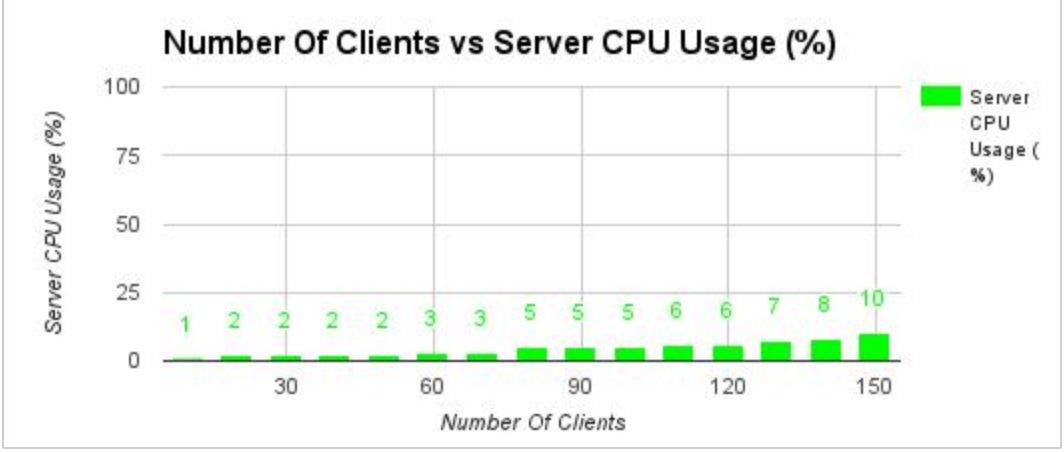


Fig 4. Knowing that server CPU usage with 150 clients is 10% we also know that at 1200 clients CPU usage will be 80% and at 1500 clients CPU usage will be 100%. In a real world scenario even 100 clients at the same time in the same chat is considered a lot, so 1500 clients is well beyond what we consider "large-scale communication."

### *6.2 Significance*

Though latency (ms) increases as the number of clients increases, SecretRoom already surpases scalability standards for traditional DC-Nets which can only scale to more than a few tens of clients before becoming unusable [11]. Even if more clients joined the chat, SecretRoom's latency (ms) is still low enough to guarantee a good user experience. Observing our results, it is interesting how even with 150 clients the latency is 298 (ms). This means that it *only* takes 298 (ms) — a time that seem instantaneous to a human user — for the server to receive all the XOR messages, derive the message, execute multiple chat rounds, and broadcast the message to all

clients while also managing these processes for two other chat rooms. We have thus shown that both strong anonymity and performance can be achieved for moderate numbers of people.

## 7 Discussion

In this section we explain why our protocol is secure and how it is more secure than traditional DC-Nets. We also present a general summary of our work and how it improves on traditional DC-Nets.

### *7.1 Security Analysis*

SecretRoom provides true anonymity because all client activities are indistinguishable, similar to DC-Nets [11]. For instance, when one client wants to send a message, *every* client sends a message to preserve anonymity of that one client. Using this approach an attacker cannot target any one client as they are all seemingly doing the same thing.

Additionally, our protocol also ensures that the server does not learn anything about the clients. Specifically, when building the pairwise secrets the server only knows the client's *public numbers* ($g^a \bmod p$) [10]. Because the server does not know each client's *private number* it cannot compute $B^a \bmod p$ and learn the clients' pairwise secrets. It thus can't learn if a client XORed a non-zero message, since the XOR sum of the pairwise secrets of each client is unknown. Finally, no adversary, including the server, learns about the sender of any message through observing patterns in the messages being sent by each client; the shared pairwise secrets are pseudorandom, and XORing the secrets makes the messages look indistinguishable from random numbers [17].

Furthermore, the security of the schedule building using shuffling was proven to be secure — that is, it does not reveal any identifying information about any particular client — in previous work [12] and is similarly applicable in SecretRoom. The server can thus be malicious in our protocol without compromising clients' anonymity. However, we note that the current implementation of SecretRoom uses the insecure schedule building that assumes the server is trusted for performance reasons.

Lastly, unlike DC-Nets, our protocol also ensures that anonymity is independent of the number of real clients in a chat. Namely, by introducing client AIs, *malicious* clients cannot "team up" against *honest* clients to find information about them. To understand this claim, consider a scenario where there are 5 real clients in a chat. Assume 3 of those clients are malicious and 2 of them are honest. The three malicious clients could team against the honest clients and prove that they are the ones actually sending messages by either not sending any messages or simply

pre-agreeing on a list of messages that they know is theirs but are designed to act as decoy messages. In either case the malicious clients can prove that the honest clients are the ones who actually spoke as they were the only ones out of the five people who actually sent *real* messages. Our protocol prevents this by spawning client AIs which also take part in the chat. In SecretRoom, the client AIs are always honest, and thus prevent malicious clients from teaming up against honest clients, as the number of honest clients (real clients and the AIs) is sufficiently large.

### *7.2 Comparison to Previous Work*

Like Tor and other anonymous communication systems, SecretRoom allows clients to communicate on public networks without compromising identifying information. However, SecretRoom improves on other systems in various ways. Unlike Tor [9], SecretRoom does not only hide where messages come from and go to, but also prevents against end-to-end timing attacks. That is, even if an observer is watching the traffic both from a specific client's computer and from the server's computer, he will be unable to tell if a message sent was an actual message or just a blank. Additionally, while Tor's security can break down when there are too few clients, SecretRoom always has client AIs, therefore making it impossible for attackers to suspect a specific group of people of sending a specific message.

SecretRoom also improves the DC-Net protocol by using a server-client model rather than the traditional peer-to-peer model [11]. Due to the presence of a server, SecretRoom fixes problems of DC-Nets such as communication overhead and collisions. Communication overhead is greatly improved from $O(n^2)$ to $O(n)$, making large-scale communication much more practical. This is different from the the traditional protocol which requires each client to communicate with *every* other client to send a message, making the overhead $O(n^2)$ [11]. SecretRoom also reuses DH *public data* to build the pairwise secrets and the schedule all in one step. Furthermore, when communicating via DC-Nets, SecretRoom's server does all of the synchronization and message derivation. Conversely, in naive DC-Nets, each individual client has to do these steps, slowing down all communications. SecretRoom also improves usability in many different ways compared to previous works. For instance, the complicated protocol is completely transparent to the clients, as the secrets and metadata are rebuilt dynamically at the start of each chat round if clients join or leave during an ongoing chat round. Furthermore, each client can type up to four different messages every second, which we believe is sufficient for any reasonable chat client. Ultimately by improving on these key aspects of DC-Nets, SecretRoom allows both low-latency communication and increased anonymity for hundreds of clients.

## 8 Conclusion and Future Work

In the future, we want to improve SecretRoom in various ways. First, we would want to be able to detect disruption, which would happen when one or more clients do not send the actual results of the XORed pairwise secrets. Because the validity of the final XOR depends on the validity of each client's individual XOR results, the final XOR would just reveal meaningless messages [11]. Another security issue that we could address in future work is the diversity of the client AIs. Currently, all AIs are located in one physical location, and a powerful adversary could infer that all the AIs are not real clients. Instead, we hope to provide an infrastructure in the future for generating a diverse set of bots. In addition, if the server crashes in our current model, the clients no longer have any way to talk with each other. We could mitigate this problem if we had multiple servers instead. Finally, we wish to implement the schedule-building from [12] where the server does not need to be trusted.

In conclusion, SecretRoom is a chat service that uses the DC-Net communication model in order to guarantee full anonymity, but introduces a server to handle all the computations and communications. The addition of a server reduces communication time from $O(n^2)$ to $O(n)$, making SecretRoom a much more practical way of communication in large groups of clients than naive DC-Nets. SecretRoom is thus fast enough to run over the Internet while being more secure than previous Internet-based works.

## Acknowledgements

We would like to thank MIT PRIMES and it's leaders for running this program. Thank you to Albert Kwon for suggesting reading material, meeting weekly, and helping debug code. We appreciate Prof. Srini Devadas for his suggestions and for thinking of this project.

## References

[1] Anon. Automate and enhance your workflow. Retrieved 2015 from http://gulpjs.com/.

[2] Anon. 2012. Bower. (2012). Retrieved 2015 from http://bower.io/

[3] Anon. Express - Node.js web application framework. Retrieved 2015 from http://expressjs.com/.

[4] Anon. 2013. Former U.S. officials give NSA whistleblower Snowden award in Russia - World. (October 2013). Retrieved 2015 from http://www.haaretz.com/news/world/1.551784

[5] Anon. 1995. McIntyre v. Ohio Elections Comm'n, 514 U.S. 334 (1995). (1995).

[6] Anon. 2015. The cloud for modern business. (2015). Retrieved 2015 from https://azure.microsoft.com

[7] Anon. 2011. The Universal Declaration of Human Rights. Visions Seen The Evolution of International Human Rights (2011).

[8] Anon. TypeScript lets you write JavaScript the way you really want to.TypeScript is a typed superset of JavaScript that compiles to plain JavaScript.Any browser. Any host. Any OS. Open Source. Retrieved 2015 from http://www.typescriptlang.org/.

[9] John Barker, Peter Hannay, and Patryk Szewczyk. Using Traffic Analysis to Identify the Second Generation Onion Router. 2011 IFIP 9th International Conference on Embedded and Ubiquitous Computing.

[10] Daniel J. Bernstein. Curve25519: New Diffie-Hellman Speed Records. Public Key Cryptography - PKC 2006 Lecture Notes in Computer Science, 207–228.

[11] David Chaum. 1988. The dining cryptographers problem: Unconditional sender and recipient untraceability. Journal of Cryptology J. Cryptology (1988), 65–75.

[12] Henry Corrigan-Gibbs and Bryan Ford. 2010. Dissent. Proceedings of the 17th ACM conference on Computer and communications security - CCS '10 (2010).

[13] Ryan Dahl. Node.js. Retrieved 2015 from https://nodejs.org/en/.

[14] Dave Methvin and Kris Borchers. 2015. jQuery. (2015). Retrieved 2015 from https://jquery.com/

[15] Mark Otto and Jacob Thornton. 2015. Bootstrap · The world's most popular mobile-first and responsive front-end framework. (2015). Retrieved 2015 from http://getbootstrap.com/

[16] Guillermo Rauch. 2015. Socket.IO. (2015). Retrieved 2015 from http://socket.io/

[17] C.E. Shannon. 1949. Communication Theory of Secrecy Systems*. Bell System Technical Journal 28, 4 (1949), 656–715. DOI:http://dx.doi.org/10.1002/j.1538-7305.1949.tb00928.x

[18] Arthur de Wolf, Dave Morton, Erwin Van Lun, Karolina Kuligowska, and Xander Verduijn. 2015. Chatbots.org - Virtual assistants, virtual agents, chat bots ... (2015). Retrieved 2015 from http://www.chatbots.org/

[19]Andrew Yao. 1982. Theory and application of trapdoor functions. Foundations of Computer Science (1982), 80–91. DOI:http://dx.doi.org/10.1109/SFCS.1982.45