Dining Cryptographers – The Protocol

Immanuel Scholz

http://www.eigenheimstrasse.de/svn/dc

1 Introduction

What are dining cryptographer networks? How and why do they work? And why is there still no real implementation available?

Back in 1988, David Chaum wrote an article for the first issue of "Journal of Cryptology" about a method to provide sender and receiver anonymity in a closed group network [2]. He called it "The Dining Cryptographers Problem", after his introductory example.

In his example, three cryptographers meet for dinner, which has paid paid beforehand. They are curious, whether one of them has paid the dinner or whether it was sponsored by the government. So they came up with the DC-protocol.

1.1 Original Protocol

Each cryptographer *exchanges a secret key* with both other cryptographers by flipping a coin under cover. Then, all three *announce the sum* of the coin flips[1] except if one really paid for the dinner. Then he *"adds" this message* by stating the inverse of the coin flip. After all cryptographers announced their sums, the *sums are summed up* again. If the final number is 0, nobody said that he paid, so the dinner must have been paid by the government.

To summarize, the following is necessary to set up a DC-network:

1. Exchange symmetric keys with other participants. As example the key between Alice and Bob is k_{ab}, between Alice and Charlie k_{ac} and between Bob and Charlie k_{bc}. One of the participants gets the inverse element of the key, e.g. when Alice gets k_{ab}, Bob remembers $-k_{ab}$.

2. Participants sum up their keys and add their message to the sum. So Alice calculates $k_{ab} + k_{ac} + m_{alice}$, Bob $-k_{ab} + k_{bc} + m_{bob}$ and Charlie $-k_{ac} - k_{bc} + m_{charlie}$. Only one of the participants should send a message at the same time.[2]

3. All sums are summed up again. The final result is the message of the one who had something to say or 0, if nobody said anything.

$$k_{ab} + k_{ac} + m_{alice} - k_{ab} + k_{bc} + m_{bob} - k_{ac} - k_{bc} + m_{charlie} = m_{alice} + m_{bob} + m_{charlie}$$

Although the network can run on a coin-flip-base, using another mathematical group is more appropriate. The formula works for all abelian groups.

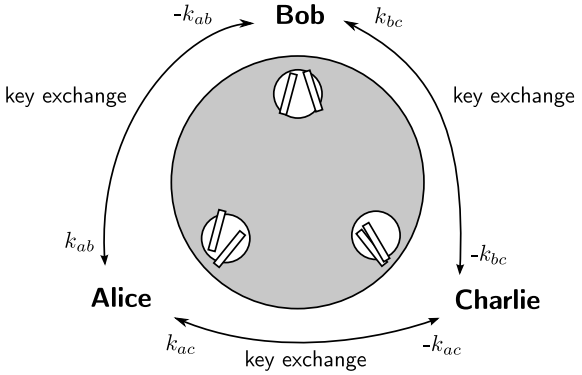

Figure 1: Alice, Bob and Charlie forming a DC – network

[1] David Chaum talked about stating "whether the two coins he can see...fell on the same side or on different sides". Mathematically, this can be expressed as binary addition, where H is head, T is tail, $H+H=T$, $H+T=H$, $T+T=T$, H ="different sites" and T ="same site".

[2] There exists a collision resolving algorithm with no data loss, but this is beyond the scope of this paper. [4, page 188–190]

Also, there may be more than three participants. It is not required that every participant exchanges keys with every other, but it does no harm to do so either.

1.2 Security

So why is it secure? Let's start with a definition of the attacker model.

For every message, every participant in an anonymity network can be in the role of the receiver, sender, none or both. The security in this network is broken, if an attacker learns something about a participant's role from the network. The attacker may observe communication lines[3] and may collude with a number of participants. The strongest possible attacker in this definition is observing *all* lines and collude with all but two participants. Naturally, if only one participant is not cooperating, his anonymity is broken, because he sends / receives if nobody else does. "Colluding" means, the attacker knows all the exchanged keys and the individual messages of the participant.

Providing receiver anonymity is easy. Every message is broadcast to every participant. For sender anonymity, everyone exchanges random keys with everyone else and add them to the value he sends over the wire. So in our example, the strongest attacker knows all the three values $k_{ab} + k_{ac} + m_{alice}$, $-k_{ab} + k_{bc} + m_{bob}$ and $-k_{ac} - k_{bc} + m_{charlie}$ as well as the keys of one participant, e. g. Bob: k_{ab}, k_{bc} and m_{bob}. Even then, he cannot learn m_{alice} or $m_{charlie}$. So he cannot know who of both sent the message.

For example, Bob adds the value observed from Charlie to his key exchanged with Charlie: $-k_{ac} - k_{bc} + m_{charlie} + k_{bc} = -k_{ac} + m_{charlie}$. As $-k_{ac}$ is a random value not known to Bob, he does not learn anything about $m_{charlie}$. $-k_{ac}$ acts as a key for an one-time-pad to "hide" the value of the message. This holds true for any formulas Bob calculates and is proved in [2].

1.3 Performance

Maybe the single most important reason, why there is no widespread DC–network implementation, is the amount of overhead it requires in most scenarios, especially for standard bilateral communications like web surfing. To achieve sender and receiver anonymity, every possible sender and receiver has to transfer a whole block each time, whether he wants to send something or not. This means, for w people in the network, a single byte requires a minimum of $2w$ bytes transferred. As the grade of anonymity is directly related to w, it should be as large as possible. And even if nobody in the network sends a message, all participants have to transfer their message block.

Finally, each byte requires each participant to send and receive one byte, which does not play well together with asymmetric connection speeds found in some countries.[4]

To reduce the amount of overhead, two things should be implemented. First, an anonymous reservation scheme helps to transfer data only, when really someone has something to say. Second, an application that needs message broadcasting is highly preferable, as broadcasting is done by the DC-network anyway. This reduces the overhead to $2w$ bytes transferred for every w bytes information.

2 Time

So far, we thought about one message. To transfer more messages, time is separated by *ticks* into *rounds*. Each participant has to send exactly once each round. A tick occurs, if the last participant sent his message for the current round or after some timeout expired, in which case all pending clients are considered broken and removed from the network. Each round, each client has to do the following things:

- Get the list of all participants and their keys
- Prepare and send the new message
- Get the message after all clients have sent
- Know that all other have received the message

The last item might be a surprise, but the following section explains the reason behind it.

[3] As example by controlling routers between the participant's computers.

[4] In Germany, ADSL connections have several MBit per second download speed but usually no more than one MBit per second upload speed.

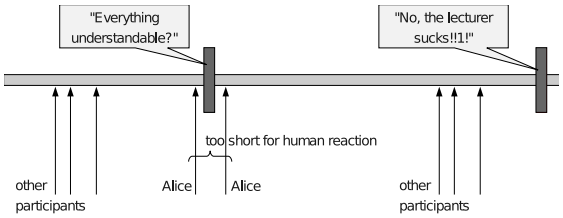

Figure 2: Timing Attack

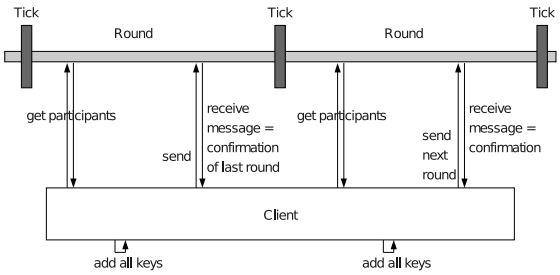

Figure 3: The protocol

2.1 Linkability

When considering a stream of messages, another privacy aspect is introduced: *linkability*. An attacker should not be able to link the role a participant has for different messages. At least, the used technical system should not give him any help in linking. In a chat, it is unlikely that a person is answering its own questions. However, the system should not provide any additional evidence about the origin of the answer, so it should be "as unlikely as without the technical system" to answer a question posted by himself.

As a key requirement, the network must ensure that every participant receives every message (receiver anonymity). Else, e. g. an attacker on a chat application could prevent a specific member from getting the question and look whether the answer is still posted.

Also, all participants have to be treated equally. In every round, everybody can try to send and even try to send a collision with himself. [4, page 187]

2.2 Timing Attack

Figure 2 shows a common attack, using the ability of the attacker to link two messages together. The attacker observes that Alice send her messages just before and just after the new round. Also, the attacker recognizes that the second messages is an answer to the first, so it is very likely that Alice did not send the second message, as she did not have time to read, type and send the answer.

One way to decrease the possibility of a successful timing attack: The server can delay the publication of the message, e. g. for one round. An easy way of doing this is returning the message of the last round within the connection, where the client sends the current message.

2.3 The Protocol

Figure 3 shows an overview of the protocol used in the implementation. The four necessary items in the list above are split to two adjacent rounds, where the return value of the send operation returns the message together with the confirmation. In this scheme, the server could lie about the distribution of the message. A modification of the DC-network protocol [4, page 184–185] has been implemented, so that the key of the new round depends on the message of the last round. This ensures that all participants got the correct message, even in scenarios with untrusted servers.

3 Excluding bad clients

To improve the availability, a fast way of detecting and excluding broken or malicious clients is required. Clients who do not send are excluded by the server at the next tick. After a broken round, all clients do a re-keying by requesting the participants' keys (see section 4) or sending their key to the server again.

Detection of malicious clients is a bit more complex and are beyond the scope of this paper. In short, an anonymous reservation scheme is deployed as described in [2]. Together with trap messages, malicious clients trying to disturb the network can be detected without restrictions on the anonymity. Other social precautions like "joining by invitation" or reputation systems could be deployed as well.

4 Key Exchange

Keys are exchanged using the Diffie–Hellman key agreement protocol [3]. The server publishes a prime p and generator g. Alice generates x_{alice} mod p at random. She calculates $g^{x_{alice}}$ mod p and sends this as her "key" to the server. Bob does the same. To generate a secret key between the two, both retrieve the key of the other and Alice calculates $(g^{x_{bob}})^{x_{alice}} = g^{x_{alice}x_{bob}}$, while Bob calculates $(g^{x_{alice}})^{x_{bob}} = g^{x_{alice}x_{bob}}$. The published keys have to be signed by an official signing key. The current implementation uses GnuPG for this purpose [1].

One advantage of using Diffie–Hellman is that no direct client to client communication is needed. Also, everyone exchanges keys with everyone, providing maximum security against attackers controlling even most of the other participants.

Diffie–Hellman is practical for rather short secrets only, typically 1024 bits. So the key is the seed to a pseudo random number generator. Unfortunately, the random sequence becomes provable, which means that the round key can be verified by publishing the seed. As example, if the round message is "Bob sucks", and Bob wants to know whether Alice or Charlie said this, he forces Charlie to publish his seed. Bob can verify that Charlie did not lie about the seed, as either the empty message or the insult shows up when Bob does the number generation with Charlie's seed.

Fortunately, Alice and Charlie could have exchanged an *additional* key or even a true random key for a true one-time-pad. So Charlie can make excuses when asked for the seed. See section 5.1 for more about the additional key.

5 On-demand disclosure

David Chaum proposed a scheme to disclose the origin of a message [2, chapter 2.6]. He suggested that every participant signs the symmetric key exchanged using a digital signature system and gives the signature to the other participant. E. g. the key k_{ab} exchanged between Alice and Bob is signed by both and Alice gets $sign_{bob}(k_{ab})$ while Bob gets $sign_{alice}(k_{ab})$. In case of "disclosure", all participants publish all exchanged keys and Alice can prove that Bob did not lie while Bob proves Alice's key testimony. We propose another scheme

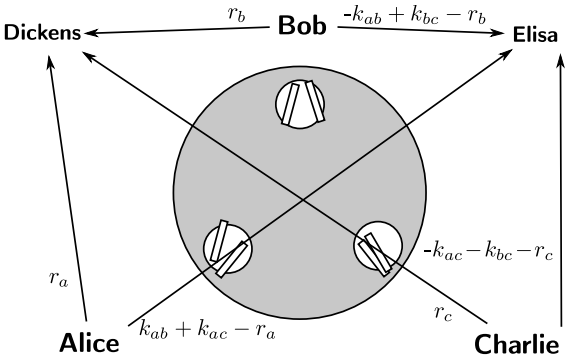

Figure 4: Watchmen on-demand disclosure scheme

which we call "watchmen disclosure scheme".

There exist n predefined watchmen, who may work together to disclose the sender of a specific message. Each client has to split his round key sum into n parts and send one part to each watchman. The split is done by generating $n-1$ random numbers r_1 to r_{n-1}, which are send to the first $n-1$ watchmen. The last watchman gets $k - \sum r_i$. Before a round message is published, all watchmen add their parts from all participants together and publish their sums. The sum of all sums must be 0. This is used to verify that no single participant lies about his round key. In case of "disclosure", the watchmen release all individual parts and so the round keys of all participants can be reconstructed. Figure 4 shows the key parts sent for two watchmen Dickens and Elisa.

5.1 Conspiracy

In a mood of plotting, Alice and Charlie exchange an additional key l_{ab}. They agree to use the key only on their round keys but not on the key parts send to the watchmen. In normal operation, the watchmen do not recognize the additional key – the sum of their sums is still 0. But if a disclosure is required, the keys from Alice *and* Charlie differ from the reconstructed round keys. If one of Alice or Charlie sent the message, nobody knows, who of the both actually sent it. Of course, both could be considered "bad guys".

5.2 Threshold secret sharing scheme

The watchmen scheme can be modified, to decrease the share of watchman required to disclose a message, as example in scenarios where only 5 out of 10 watchmen have to agree to track the messages origin. Adi Shamir proposed the threshold secret sharing scheme based on polynomials [5]. This can be applied if the following restrictions are made:

- All participants must use same set of watchmen, which may be necessary anyway
- The x-coordinates of the threshold scheme is fixed for each watchman

5.3 Comparison between DC-disclosure and Watchmen

So what are the practical differences between the original proposal based on asymmetric signed round keys and the watchmen scheme?

Unfortunately[5], both schemes are insecure against conspirations. However, in the watchmen scheme, a conspiration cannot be formed *after* the message was published, as the additional key has to be added before the parts are sent to the watchmen. In the original scheme, e. g. the participants could agree to cover each other depending on the message content.

For the process of disclosing a sender, the watchmen scheme does not require to contact the participants. Only the watchmen have to be contacted. This also means, the act of disclosing a sender can be kept secret.[6]

6 Serverless DC-network

What do we need a server for? The only things the server does is coordinating the clients, providing web-space for the messages and keys and finally adding some message parts. The coordination could be done within the network itself, web-space is available everywhere today and the adding can be done by each client on its own.

Having said that, in some countries server maintainer must gather information that can trace down communication partners (data retention). Without any server, there is no such information to store. So the protocol proposed in section 1.1 can be modified:

First every participant needs some place to publish any arbitrary information, e. g. a personal blog, a web-forum, a MySpace[7] account or something like that. Each participant publishes his round sums on this account.

Log-in can be done by invitation, where someone already in the network publishes the URL and Diffie–Hellman key of the new participant in the network. There could be websites doing this as substitute, if open membership is desired.

Log-out is done by sending a goodbye message (or automatically by failing to publish a round).

[5] Or fortunately – depending on the point of view.

[6] Again, whether this is an advantage or not depends on the point of view.

[7] Provider of free web-space. `http://www.myspace.com`

References

[1] Gnu privacy guard. Available from World Wide Web: `http://www.gnupg.org`.

[2] David Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1(1):65–75, 1988.

[3] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976. Available from World Wide Web: `http://citeseer.ist.psu.edu/diffie76new.html`.

[4] Andreas Pfitzmann. Sicherheit in rechnernetzen, 2000. Available from World Wide Web: `http://dud.inf.tu-dresden.de/~pfitza/DSuKrypt.pdf`.

[5] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, November 1979. Available from World Wide Web: `http://portal.acm.org/citation.cfm?id=359176`.