Martin Kleppmann, Stephan A. Kollmann,
Diana A. Vasile, and Alastair R. Beresford

Department of Computer Science and Technology, University of Cambridge, UK
{mk428,sak70,dac53,arb33}@cl.cam.ac.uk

Abstract. We examine the security of collaboration systems, where several users access and contribute to some shared resource, document, or database. To protect such systems against malicious servers, we can build upon existing secure messaging protocols that provide end-to-end security. However, if we want to ensure the consistency of the shared data in the presence of malicious users, we require features that are not available in existing messaging protocols. We investigate the protocol failures that may arise when a new collaborator is added to a group, and discuss approaches for enforcing the integrity of the shared data.

Keywords: group communication · collaborative editing · secure messaging

1 Introduction

Secure messaging apps with end-to-end encryption, such as Signal, WhatsApp, iMessage and Telegram, have broken into the mainstream: for example, WhatsApp alone has 1.3 billion monthly active users [16]. The success of these apps demonstrates that it is feasible for protocols with strong security properties to be deployed at internet-wide scale, and that their benefits can be enjoyed by users who are not technical experts.

However, secure messaging alone is not sufficient for protecting all forms of sensitive data exchange. Some communication takes the form of collaborating on some shared resource, such as a document, or database. For example, journalists collaborate on sensitive investigations by interviewing sources, analysing documents, and sharing their notes and drafts with colleagues [11,12]; lawyers collaborate on contracts and other sensitive documents while communicating with their clients under legal privilege [10]; and medical records are maintained by several specialists involved in treating a patient. Most currently deployed systems for these forms of collaboration rely on a server that is trusted to maintain the confidentiality and integrity of the data.

In this paper we discuss how existing protocols for secure messaging can be leveraged to bring end-to-end security to scenarios in which several users collaborate on a database or a set of shared documents. We give a brief overview of existing algorithms and technologies, report on lessons learnt from our initial implementation of secure collaboration software, and highlight some open problems that we hope will stimulate further work in the information security community.

2 Threat Model and Security Objectives

We assume that the collaboration software has any number of *users*, each of whom may have one or more *devices* (which may be desktop or laptop computers, or mobile devices such as tablets and smartphones). Users and their devices may form groups of *collaborators*, and the collaborators in each group have shared access to a particular document or dataset. Each collaborating device maintains a copy (replica) of the shared data on its local storage.

Devices may frequently be *offline* and unable to communicate, for example because they might be mobile devices with poor cellular data coverage. We require that devices should be able to modify their local copy of the data even while offline, and send their changes to other devices when they are next online.

The system may also include some number of *servers*, which store messages for any recipient devices that are offline, and forward them to those devices when they are next online. Devices may communicate with each other directly (e.g. via a LAN, Bluetooth, or peer-to-peer over the Internet), or indirectly via servers. Furthermore we assume the existence of a public key infrastructure (PKI) through which users and devices can authenticate each other.

We consider the following types of adversary:

Network Attacker. This adversary has full control over any network via which devices communicate, including the ability to observe and modify all traffic.

Malicious Server. This adversary controls any messages sent via or stored on a server, including the ability to observe and modify any messages.

Malicious User. This adversary is able to create any number of devices that may participate in group collaboration, and which may deviate arbitrarily from the protocol specification.

In the face of these adversaries we seek the following security properties:

Confidentiality. The data shared between a group of collaborators cannot be obtained by an adversary who is not a member of that group.

Integrity. The data shared between a group of collaborators cannot be modified by an adversary who is not a member of that group.

Closeness. A user or device can become a group member only by explicitly being added by a group administrator.

Convergence. When any honest group members communicate, their local copies of the shared data converge towards a consistent state (even if some other group members are malicious).

We propose encoding the shared data and any modifications as messages, and using a secure group messaging protocol to exchange them among collaborators. Existing secure group messaging protocols maintain the confidentiality and integrity properties in the presence of all types of adversary [4,15]. Closeness is sometimes weaker in existing protocols: for example, WhatsApp does not guarantee closeness in the presence of a malicious server [13]. However, group key agreement protocols that ensure closeness have been studied previously [7], so we do not consider this property further in this paper.

Thus, when building a secure collaboration protocol on top of a secure messaging protocol, the primary security goal is to ensure *convergence* in the presence of the aforementioned adversaries.

3 Convergence of Shared State

Since we allow the data on a device's local storage to be modified while the device is offline, independent modifications on different devices can cause their copies of the shared data to become inconsistent with each other. Fortunately, this problem has been studied extensively in the distributed systems literature. We propose using *Conflict-free Replicated Data Types* or *CRDTs* [8,14], a family of algorithms that provide abstractions and protocols for automatically resolving conflicts due to concurrent modifications.

CRDTs provide a consistency property called *strong eventual consistency*, which guarantees that whenever any two devices have seen the same set of updates (even if the updates were delivered in a different order), the data on those devices is in the same state [6,14]. This property implies that the state of a device is determined entirely by the set of updates it has seen.

Thus, we can achieve the convergence property for collaborative data by encoding every update as a message and sending it to other devices via a secure group messaging protocol. On each device, we use a CRDT to interpret the set of messages delivered to that device, and derive its local copy of the shared data from those messages. Now, the problem of achieving convergence is reduced to ensuring that all honest group members receive the same set of messages.

In the context of secure messaging protocols, ensuring that group members receive the same set of messages is known as *transcript consistency* [4,15]. (Sometimes transcript consistency is taken to mean that all group members must receive the same sequence of messages in the same order; for our purposes, it is sufficient to require the weaker property that collaborators must receive the same set of messages, regardless of order.) Not all messaging protocols provide this property; for example, Signal does not ensure transcript consistency in the presence of a malicious user [13]. However, the property can be implemented as a separate layer on top of an existing messaging protocol.

A simple approach based on a hash function is illustrated in Figure 1: whenever a device sends a message to the group (e.g. message m_4), it includes a hash of the last message it sent (m_2), and the hashes of any other messages it received in the intervening period (m_3). A recipient accepts an incoming message only after it has received all prior messages that it depends on, which are referenced by their hashes. Assuming preimage resistance of the hash function, whenever two devices observe the same message, then they must have also received the same set of prior messages (namely those that are transitively reachable through their hash references).

The construction in Figure 1 is similar to the internal structure of a Git repository [3], in which each commit references the hash of one or more parent commits.

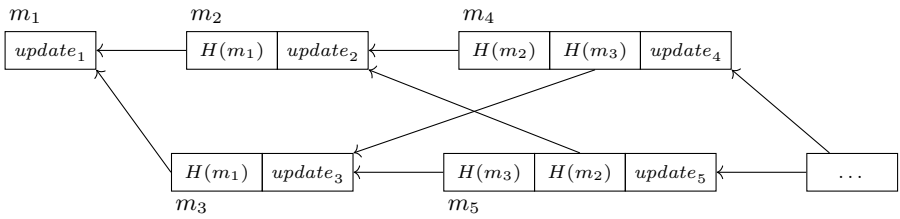

Fig. 1. Chaining messages by referencing hashes of previous messages.

4 Adding New Collaborators

The approach in Section 3 ensures convergence in a static group of collaborators, where all members are added when a group is created. In this setting, every message in the history of the group is delivered to every member device. However, if the membership is dynamic – that is, if group members can be added or removed – additional challenges arise.

With most group messaging protocols, when a new member is added, that member is able to receive any messages that were sent after they were added, but no prior messages. However, in the context of collaboration on some shared data, receiving later messages is not sufficient: the new member also requires a copy of the shared data to which any subsequent updates can be applied.

The simplest solution is to give the new member the full update history: that is, the administrator who invites the new member also sends the new member a copy of all past messages sent in the group. If a hash-chaining construction like in Figure 1 is used, the new member can check the integrity of this message history by computing the hashes of the messages and comparing them to the referenced hashes.

However, sending the full update history has two downsides. Firstly, it may be much larger than a copy of the current state, and thus inefficient to store and transmit. Secondly, the full update history includes every past version of the data, including any content that has been deleted and is no longer part of the current state. For privacy reasons, the existing collaborators may not want to expose the full details of former versions of the data to the new collaborator.

5 Checking the Integrity of Snapshots

If it is not acceptable to send the full update history to a new collaborator, a snapshot of the current state of the shared data must be sent instead. However, a naive snapshot protocol would lose the convergence property in the presence of a malicious user: namely, the user who sends the snapshot may send data that does not match the true current state, and thus cause the new collaborator's state to diverge from the rest of the group. For example, the malicious user could claim that another user wrote something that, in fact, they never wrote.

We are exploring protocols that allow the new collaborator to verify that a snapshot is consistent with the prior editing history of the shared data, without revealing the full update history to the new collaborator. Approaches include:

1. The snapshot can be sent to all group members, not just the new collaborator. Existing group members can then check the integrity of the snapshot, and vote on whether they believe the snapshot to be correct or not. A Byzantine consensus protocol [2,5] can make this voting process resilient to malicious users, provided that a quorum (typically, more than 2/3) of voting members is honest. However, this approach requires members to be online in order to vote. If members use mobile devices that are frequently offline, as proposed in Section 2, a voting protocol may introduce prohibitive delays before the snapshot integrity is confirmed.
2. As an alternative, the new collaborator could initially trust the first snapshot it receives, and then run a consistency-checking protocol in the background. This protocol would not prevent the new collaborator from seeing an inconsistent snapshot, but it could ensure that any inconsistency is eventually detected, provided that the new collaborator eventually communicates with an honest group member. This approach is analogous to Certificate Transparency [9], which does not prevent certificate authorities from misissuing certificates, but which deters such behaviour by making it detectable and irrefutable.
3. There may be cryptographic constructions that allow the creator of the snapshot to prove to the new collaborator that the snapshot is consistent with the full message history, without revealing the message history. For example, we are currently exploring the use of redactable signatures and one-way accumulators [1] for this purpose.

In general, we may differentiate *fail-safe* and *fail-secure* approaches to snapshot integrity checking. A fail-secure (or fail-closed) approach in this context means that the new collaborator must wait until the integrity of the shared data has been fully verified, e.g. using some voting protocol or cryptographic proof, before they are allowed to see it. On the other hand, a fail-safe (or fail-open) approach would allow the new collaborator to immediately see the snapshot — even if it might be incorrect — and resolve any inconsistencies after the fact.

6 Conclusions

End-to-end encryption is now in regular use by over 1 billion people for secure messaging. Yet, end-to-end encryption is not currently used by collaborative applications where multiple people modify some shared resource, such as a document or database. In this paper we have outlined a method of building collaborative apps on top of secure messaging protocols, providing not only confidentiality and integrity in the face of network attackers and malicious servers, but also the properties of closeness and convergence. Handling the insider threat of a malicious collaborating user is more challenging, and we highlighted snapshot integrity as a particular issue which requires further work to fully address.

References

1. Benaloh, J., De Mare, M.: One-way accumulators: A decentralized alternative to digital signatures. In: Workshop on the Theory and Application of Cryptographic Techniques. pp. 274–285. Springer (May 1993). https://doi.org/10.1007/3-540-48285-7_24
2. Castro, M., Liskov, B.H.: Practical Byzantine fault tolerance. In: 3rd USENIX Symposium on Operating Systems Design and Implementation (OSDI) (Feb 1999)
3. Chacon, S., Straub, B.: Pro Git. Apress, second edn. (2014), https://git-scm.com/book/en/v2
4. Cohn-Gordon, K., Cremers, C., Garratt, L., Millican, J., Milner, K.: On ends-to-ends encryption: Asynchronous group messaging with strong security guarantees. Tech. Rep. 2017/666, IACR Cryptology ePrint (Jul 2017), https://eprint.iacr.org/2017/666
5. Correia, M., Veronese, G.S., Neves, N.F., Verissimo, P.: Byzantine consensus in asynchronous message-passing systems: a survey. International Journal of Critical Computer-Based Systems **2**(2), 141–161 (2011). https://doi.org/10.1504/IJCCBS.2011.041257
6. Gomes, V.B.F., Kleppmann, M., Mulligan, D.P., Beresford, A.R.: Verifying strong eventual consistency in distributed systems. Proceedings of the ACM on Programming Languages (PACMPL) **1**(OOPSLA) (Oct 2017). https://doi.org/10.1145/3133933
7. Kim, Y., Perrig, A., Tsudik, G.: Communication-efficient group key agreement. In: 16th IFIP Information Security Conference (SEC). pp. 229–244 (Jun 2001). https://doi.org/10.1007/0-306-46998-7_16
8. Kleppmann, M., Beresford, A.R.: A conflict-free replicated JSON datatype. IEEE Transactions on Parallel and Distributed Systems **28**(10), 2733–2746 (Apr 2017). https://doi.org/10.1109/TPDS.2017.2697382
9. Laurie, B., Langley, A., Kasper, E.: RFC 6962: Certificate transparency. IETF, https://tools.ietf.org/html/rfc6962 (Jun 2013)
10. Lerner, A., Zeng, E., Roesner, F.: Confidante: Usable encrypted email: A case study with lawyers and journalists. In: 2nd IEEE European Symposium on Security and Privacy. pp. 385–400 (Apr 2017). https://doi.org/10.1109/EuroSP.2017.41
11. McGregor, S.E., Charters, P., Holliday, T., Roesner, F.: Investigating the computer security practices and needs of journalists. In: 24th USENIX Security Symposium (Aug 2015)
12. McGregor, S.E., Watkins, E.A., Al-Ameen, M.N., Caine, K., Roesner, F.: When the weakest link is strong: Secure collaboration in the case of the Panama Papers. In: 26th USENIX Security Symposium (Aug 2017)
13. Rösler, P., Mainka, C., Schwenk, J.: More is less: On the end-to-end security of group chats in Signal, WhatsApp, and Threema. Tech. Rep. 2017/713, IACR Cryptology ePrint (Jan 2018), https://eprint.iacr.org/2017/713
14. Shapiro, M., Preguiça, N., Baquero, C., Zawirski, M.: Conflict-free replicated data types. In: 13th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS). pp. 386–400 (Oct 2011). https://doi.org/10.1007/978-3-642-24550-3_29
15. Unger, N., Dechand, S., Bonneau, J., Fahl, S., Perl, H., Goldberg, I., Smith, M.: SoK: Secure messaging. In: IEEE Symposium on Security and Privacy. pp. 232–249 (May 2015). https://doi.org/10.1109/SP.2015.22
16. WhatsApp Inc.: Connecting one billion users every day (Jul 2017), https://blog.whatsapp.com/10000631/Connecting-One-Billion-Users-Every-Day

From Secure Messaging to Secure Collaboration (Transcript of Discussion)

Martin Kleppmann, Stephan A. Kollmann,
Diana A. Vasile, and Alastair R. Beresford

Department of Computer Science and Technology, University of Cambridge, UK

Frank Stajano: Among the people who come up with these conflict resolution algorithms, is there agreement about what to do in a situation where the edits look fundamentally incompatible to a normal person? For example, while we are offline, I delete a paragraph, whereas you add a word in the middle of that paragraph.

Reply: That's a good question. There are a few different consistency models that people use, but on the whole, they are fairly primitive in terms of what conflict resolution they do. In your example, where one user deletes an entire paragraph, another user changes something in the middle of that paragraph — I actually tried this with Google Docs. The merged result is a document in which that paragraph is missing, but that one changed word from the middle of the paragraph is still there. So you've essentially got a 'stranded' word in the middle of the document.

Frank Stajano: Are real human beings happy with this way of resolving?

Reply: Well, millions of people seem to be using Google Docs successfully, so I'm just going to assert that it seems to be good enough in practice. It would be nice to have a user interface that warns you, saying: "Hey, there were some edits made here to the same place in the document. You might want to check this is a valid, merged result." All the algorithms do is to ensure that everyone ends up in the same state, and that state doesn't lose any data, if possible.

Ilia Shumailov: I think the general flow of action is, because they keep the history of changes, you can always just go back to another change and re-merge the text yourself. At least that's what I do every time I have a conflict.

Reply: Yes indeed. They keep the editing history, and you can look backward in the editing history as well. However, from the point of view of our discussion today, take conflict resolution as a solved problem. I'm just going to assert that these algorithms work, under the assumption that the same set of messages are delivered to everybody in the group.

We have proved that for several of these protocols, if several nodes have seen the same set of operations, but maybe in a different order, then they converge towards the same state. What we need to ensure now in the protocol is that all of the participants see the same set of operations, and that's where security protocols come in.

(Presentation continues, describing the threat model and security objectives from Section 2 of the paper.)

Ian Goldberg: When you define confidentiality rigorously, it's not enough to say that non-collaborators cannot access the data, because you could have a

non-collaborator colluding with a collaborator. So you have to rule out colluding collaborators, and then it gets really complicated really fast. It's not only a closed group, you have to deal with what happens when people leave the group, or what happens if you have two colluding people in the group and one of them leaves the group. How do you stop that person?

Reply: If there's existing literature on defining these things, I'd love to hear more about that.

Ian Goldberg: It's very complicated. And just in the context of group messaging, this is active research right now and my group works on that exact thing. And you have problems involving adding and removing group members.

In the secure messaging realm, what you're talking about is sometimes called *transcript consistency*, and then there are a couple of different kinds of transcript consistency. There is *global transcript consistency*, which is a guarantee that everyone in the group sees exactly the same messages in the same order. That's very hard to do, and the UI for that is ridiculous because you may actually receive messages out of order, but then some BFT-like protocol runs and it says, "oh, that message really should have gone there". And so what is the UI going to do? You see messages and then this one jumps up four messages? It's kind of crazy.

Reply: Yeah, what we want here is a weaker kind of transcript consistency, which doesn't enforce ordering, just enforces that the set of messages is the same.

Ian Goldberg: Right. You probably do still want *causal transcript consistency*, which means a reply to any message will necessarily appear after that message.

Reply: Yes, but that's much cheaper to achieve, I think.

Ian Goldberg: Exactly, it's much easier.

Frank Stajano: I have somewhat similar comments with respect to integrity, where the fact that non-collaborators cannot modify the content is necessary but insufficient. For example, if David and I have a shared document containing the expense reports for our company, and both of us are collaborators and authorised, but neither of us should be able to go back and change some expenses that we've already approved. There may be many other integrity properties of the document besides the fact that people who are not collaborators should not be able to modify.

Reply: Yes, absolutely. Underneath the shared document is a message log containing all of the changes that ever happened. And you can have stronger integrity properties on that log, so you would not be able to tamper with the message log, for example.

(Presentation continues with Sections 3 and 4 of the paper, and posing the problem of checking the integrity of snapshots.)

Daniel Weitzner: It's an interesting problem. I wonder if you have a bit of a concurrency-versus-privacy problem? If the first two users are still actively editing and at some point – as I think Ilia suggested – they want to go back in history, then what about the user who was invited later? Does the user who joined later get to see the history after whatever snapshot you did? Do you end

up with two different versions of the document based on when you joined? If you've flattened out all the state then you've lost a lot of information for the first two users.

Reply: Yes, potentially. In particular, if there's some editing happening concurrently to the state snapshot being taken—

Daniel Weitzner: Or just afterwards. It doesn't even need to—

Reply: Yeah, "concurrently" in the distributed sense, that means, they're not aware of each other. Then yes, it could absolutely happen that this other edit can't be applied to that state, because it's assuming some kind of metadata that has been flattened out. That's absolutely a potential problem, yeah.

Frank Stajano: You say it's a problem that two thirds of the participants have to be honest, and online. I'm not sure about online, but is it not going to be the case that, in order to accept the newly invited guy, the previous guys have to agree to let him in? And therefore, at the time they say "okay, I think the new guy is alright", they could also say "this is the snapshot" (or hash of snapshot or something like that) that could contribute to the new guy having a view that they concur on.

Reply: I guess it depends what kind of permission you want to require for new people to be added. At least with Google Docs, and I think with most of the group messaging protocols, any one of the existing participants can just unilaterally invite a new person. So they don't require any kind of agreement from the group—

Frank Stajano: If this is the protocol, how can you even start worrying about privacy if any guy around can invite any of their pals? Then of course privacy goes out the window, right?

Reply: Well no, it's already the case that any one user, if they want to, can just send a copy of the document to their mates, completely outside of the protocol. There's no way of constraining what these individual people can do with their decrypted copy of the data.

Frank Stajano: Then why are you worried that they might see a past edit, if any of these guys can send it anyway, by your own assertion?

Reply: Well, the intention is that if the existing users don't want to share the past state, they have a way of sharing only the current state. Of course, if they wanted to leak the past state they could do that, but we are trying to avoid inadvertently leaking that past state when inviting a new user.

Ilia Shumailov: I think the goal is to save yourself from the server, not from the collaborators.

Frank Stajano: He's eliminated the server, hasn't he?

Ilia Shumailov: Yeah, but how do you keep it consistent?

Frank Stajano: Right, yes.

Reply: Keep what consistent?

Ilia Shumailov: The version of the document, all operations.

Reply: These algorithms will work perfectly fine without a server because they're tolerant of messages arriving out of order. You can run this on top of peer-to-peer protocol without any problems really.

Ilia Shumailov: Yeah this is just an optimisation not to share all of the operations for all of the documents with every new peer. Another interesting question would then be: what happens when the person who was offline joins in with a completely separate state of operations? Does that imply that, in order to impose new rules as a malicious user, you start DoSing all of the legitimate users? And then all of the non-legitimate users can form a current consensus with the online users, and distribute their own version of the document.

Reply: I guess that could happen, yes. I hadn't really considered users DoSing each other, but it's conceivable.

Alexander Hicks: This question is related to what you were discussing with Frank a few minutes ago. If you only want to share the latest state, I'm going to take the example of Google Docs here — it's a bit contrived, but can't you just create a new document, copy and paste what you want, and you've shared the latest state successfully? And you can even have access to your past edits by going back to your other doc if you want, without any risks. Obviously, it's maybe not practical to always open a new document, but it seems like it's doable at least.

Reply: Yes, the only problem is that there is no way for the new participant to tell whether the new copy of the document, that copy-and-pasted document, is actually consistent with what happened previously. So the person who does the copying and pasting of the document may well manipulate the document at the same time, and there is nothing stopping them.

Alexander Hicks: True, but then I guess from there on they would be satisfied they have the latest copy. Unless they're assuming that you're also editing the other copy, but then you can't really avoid that anyway.

Reply: But if you want the existing collaborators to still be able to continue editing, all three have now become equal collaborators, so you want to ensure that all three of them are in the same state. If you copy and paste the entire document into a new one, that means that all of the other participants also need to switch over to the new one, and presumably the participants would, in the process, do some comparison and checking whether the document still agrees with what they thought it was before the copy-and-paste.

Alexander Hicks: Sure, but you could probably generate some proof of that without necessarily revealing the past changes.

Reply: Well yes, that's exactly what we're trying to work out. Are there cryptographic techniques that we can use to prove the integrity?

Alexander Hicks: Something like a light client, I guess.

Reply: Like what?

Alexander Hicks: I guess maybe Paddy [Patrick McCorry] can say... for Bitcoin you have light clients, which only verify up to the past few blocks — is that correct? Something like that could probably work.

Reply: Yes, sounds conceivable. I don't know very much about Bitcoin.

Ian Goldberg: At the end of the day, you can always just run a SNARK, right? So you have a circuit that applies a change to a state, yielding another state, and the SNARK just checks that that was done correctly. It will not be

cheap to generate, but it will be cheap to check. And then you can just, along with the latest state, carry a proof that this latest state was generated correctly, without revealing any of the inputs.

Reply: Yes, that would be interesting to try.

Patrick McCorry: This is sort of similar to state channels, where everyone agrees on the new state, so it's n out of n – everyone has to sign it. And when the new person joins, everyone has to agree to that as well. Is it the same process here when someone joins the collaboration? Do you require everyone's authorisation?

Reply: No. At the moment, the way we're thinking about this is that any one member of the group can add new members without having to coordinate with the others. That works nicely if these are mobile devices that are offline most of the time, where we really don't want to have to wait for someone else to come online. Even more so if, for example, one of the devices has been dropped in the toilet by somebody, and so it is never going to come back online again. In that case, we can at most wait for some kind of quorum, but we wouldn't want to have to wait for everybody.

Ilia Shumailov: Well, then that definitely completely destroys your technique here. You said that you wanted two out of three legitimate users. Then you just add a lot of users who have copied state from you, and just say "okay, this is the new state". Right?

Reply: Yes certainly, if you allow arbitrary Sybil identities to be created, then any sort of majority voting seems a bit meaningless. Though the nice thing with these redactable signatures is that they don't depend on any majorities: we can verify the signatures with respect to all of the previous users without waiting for any communication with them.

Ilia Shumailov: Oh, so that implies that with each one of them, you actually make a confirmation that this is the document.

Reply: Yes.

Ilia Shumailov: Well, does that not imply that if I add a bunch of copies of myself and claim that this is the new document, it's still going to work? What happens in the case of conflict?

Reply: Editing conflicts within the document are on a separate layer above this. Here, at this layer, we just need to ensure that everyone sees the same set of messages.

Ilia Shumailov: So, if I clone myself a number of times, and then I say, "okay, this is the new document", what happens with the guy who comes in?

Reply: In this snapshot, part of the information are the user IDs of who wrote what. Those user IDs might be a hash of their public key, for example. So if you don't have the private key for the other participants, you wouldn't be able to impersonate edits from the other people. You could still make a brand new document, in which only you have made the edits, and there have been no edits from others. For that document you would still need some kind of checking with the other participants, and they would at that point say: "hey no, this isn't the document that we were working on – this is something completely different".