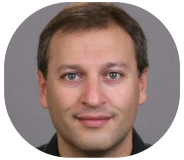
End-to-End Encrypted Zoom Meetings: Proving Security and Strengthening Liveness

Eurocrypt 2023, Lyon, France



Yevgeniy Dodis*
New York University

Daniel Jost*
New York University

Antonio Marcedone
Zoom Video Communications

Balachandar Kesavan
Zoom Video Communications

zoom

*Research performed while contracting for Zoom

Group Video Communications

- Fundamental tool, especially since the COVID 19 pandemic

- Zoom has been offering (optional) End-to-End Encryption (E2EE) for Meetings since October 2020

E2EE Messaging

- Extensive academic research:
    - Well established protocols (Signal)
    - Progress towards standardization (MLS)
    - Advanced security properties

E2EE Video Meetings

- Relatively little research

E2EE Messaging

- Extensive academic research:
  - Well established protocols (Signal)
  - Progress towards standardization (MLS)
  - Advanced security properties
- Long sessions (years)
  - Compromise recovery within a session is important (Forward Secrecy, Post Compromise Security)

E2EE Video Meetings

- Relatively little research
- Short sessions (hours)
  - Compromise recovery between sessions is often sufficient

E2EE Messaging

- Extensive academic research:
  - Well established protocols (Signal)
  - Progress towards standardization (MLS)
  - Advanced security properties
- Long sessions (years)
  - Compromise recovery within a session is important (Forward Secrecy, Post Compromise Security)
- Asynchronous (participants get past messages when they return online)

E2EE Video Meetings

- Relatively little research
- Short sessions (hours)
  - Compromise recovery between sessions is often sufficient
- Synchronous (participants must be online at the same time)
  - Liveness can be enforced!

Contributions

- Modularly define 2 new primitives that formalize the core of Zoom's E2EE Meetings protocol:
  - Continuous Multi-recipient Key Encapsulation Mechanism (cmKEM)
  - Leader-based Continuous Group Key Agreement with Liveness (LLCGKA)

Contributions

- Modularly define 2 new primitives that formalize the core of Zoom's E2EE Meetings protocol:
    - Continuous Multi-recipient Key Encapsulation Mechanism (cmKEM)
    - Leader-based Continuous Group Key Agreement with Liveness (LLCGKA)

- Prove that the core of Zoom's E2EE Meetings protocol satisfies those definitions

Contributions

- Modularly define 2 new primitives that formalize the core of Zoom's E2EE Meetings protocol:
  - Continuous Multi-recipient Key Encapsulation Mechanism (cmKEM)
  - Leader-based Continuous Group Key Agreement with Liveness (LLCGKA)

- Prove that the core of Zoom's E2EE Meetings protocol satisfies those definitions

- Propose two alternative strengthenings of the liveness properties for Zoom's LLCGKA,
  - One is deployed since Zoom Meetings v 5.13

Note: This presentation often simplifies and omits important details. Please check Zoom's Cryptography Whitepaper for more details. The statements and information provided are intended for **informational purposes only** and should not be relied upon in making a purchasing decision and may not be incorporated into any contract.

Continuous Multi-recipient Key Encapsulation Mechanism (cmKEM)

Continuous Multi-recipient Key Encapsulation Mechanism (cmKEM)

- Allows a leader to distribute a stream of keys to a variable group of participants (with help from an untrusted server).

- Parties create an ephemeral identity for each “meeting”. Each meeting can include multiple sessions, each with a different leader.

- Keys are indexed by the leader and two counters: the epoch, and a sub-epoch called period.
  - Switching periods is meant to be more efficient, but is only secure when adding participants.
  - Counters do not reset on leader changes.

cmKEM Security

Single game based definition with adaptive adversary who controls server and network.

- Adversary can corrupt long term identities, getting long term keys and the state of all ephemeral identities that are still active

cmKEM Security

Single game based definition with adaptive adversary who controls server and network.

- Adversary can corrupt long term identities, getting long term keys and the state of all ephemeral identities that are still active

Key confidentiality: An adaptive adversary cannot distinguish a cmKEM key from a random key, unless It corrupts one of the ephemeral identities with access to the key.

- Weak FS
- ephemeral identities honestly generated after a corruption maintain confidentiality

cmKEM Security

Single game based definition with adaptive adversary who controls server and network.

- Adversary can corrupt long term identities, getting long term keys and the state of all ephemeral identities that are still active

Key confidentiality: An adaptive adversary cannot distinguish a cmKEM key from a random key, unless It corrupts one of the ephemeral identities with access to the key.

- Weak FS
- ephemeral identities honestly generated after a corruption maintain confidentiality

Key Consistency: Each party who obtains a key for a given epoch and period will agree with their leader (of that epoch/period) on that key, unless either that party or the leader's state have been compromised/leaked.

- Note: stronger notions are possible

Zoom's cmKEM

- To create a new ephemeral identity, participants generate (and sign) an ephemeral DH Key pair
- To move to a new epoch, the leader picks a uniformly random seed, computes a shared DH key with each participant and encrypts the seed for them using AEAD.
- Participants ratchet the seed forward to generate keys for each period in the epoch.
- Security proof in the Random Oracle Model, based on Gap DH and the security of the signature scheme, AEAD, and PRG.
- More optimized solutions possible (e.g. TreeKEM based)

Leader-based Continuous Group Key Agreement with Liveness (LLCGKA)

LLCGKA

The functionality is similar to cmKEM, but it also keeps track of the **group roster** and accounts for **time**:

- algorithms take the current time as input
- time based events can be triggered

LLCGKA

The functionality is similar to cmKEM, but it also keeps track of the **group roster** and accounts for **time**:

- algorithms take the current time as input
- time based events can be triggered

Key confidentiality and consistency: analogous to cmKEM

LLCGKA

The functionality is similar to cmKEM, but it also keeps track of the **group roster** and accounts for **time**:

- algorithms take the current time as input
- time based events can be triggered

Key confidentiality and consistency: analogous to cmKEM

Group consistency: Any two parties who have the same (honest) leader at a given epoch/period also agree on the group roster at this and any previous state since the latest one joined

- Partitions cannot be reconciled

LLCGKA

The functionality is similar to cmKEM, but it also keeps track of the **group roster** and accounts for **time**:

- algorithms take the current time as input
- time based events can be triggered

Key confidentiality and consistency: analogous to cmKEM

Group consistency: Any two parties who have the same (honest) leader at a given epoch/period also agree on the group roster at this and any previous state since the latest one joined

- Partitions cannot be reconciled

Key Liveness: Participants either are up to date with the meeting state, or drop out.

- More formally, for any active participant in the meeting, their leader has been in the same state (epoch, period, group, key) recently (within *liveness-slack*, a protocol-dependent parameter)
- Assumes parties' clocks go at the same speed, with arbitrary offsets.

Zoom’s LLCGKA (up to v5.12)

- Directly leverages cmKEM to distribute keys
- For group consistency, leader periodically broadcasts the Leader Participant List (LPL), which includes the list of uids associated with a specific epoch and period.
- To authenticate the LPL and ensure liveness, leader broadcasts **signed** heartbeats, including:
  - A timestamp
  - Current epoch, period, and hash of the latest LPL
  - Hash of the previous heartbeat
- Participants only start using a key after the corresponding LPL and heartbeat is received.

Understanding Liveness

- Participants maintain an upper bound δ on the offset between their clock and their leader's
- They drop out if they detect that the last heartbeat they received was generated more than Δ_{live} earlier, i.e. if:

$$t_{hb} + \delta + \Delta_{live} > t_{now}$$

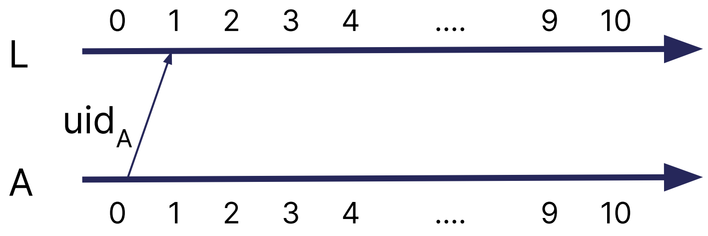
- The offset δ is the minimum difference between the time indicated in any heartbeat and the local time when it was received.

Understanding Liveness

- Participants maintain an upper bound δ on the offset between their clock and their leader's
- They drop out if they detect that the last heartbeat they received was generated more than Δ_{live} earlier, i.e. if:

$$t_{hb} + \delta + \Delta_{live} > t_{now}$$

- The offset δ is the minimum difference between the time indicated in any heartbeat and the local time when it was received.



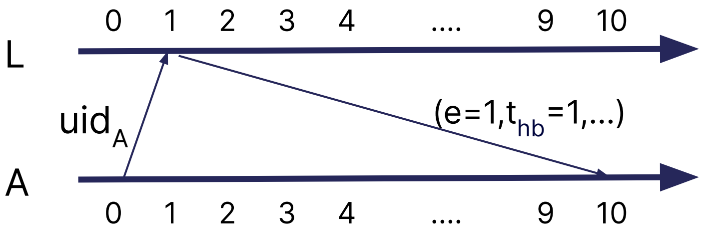
A will not drop out before Δ_{live} = 10 (assume Δ_{live} = 10)

Understanding Liveness

- Participants maintain an upper bound δ on the offset between their clock and their leader's
- They drop out if they detect that the last heartbeat they received was generated more than Δ_{live} earlier, i.e. if:

$$t_{hb} + \delta + \Delta_{live} > t_{now}$$

- The offset δ is the minimum difference between the time t_{hb} indicated in any heartbeat and the local time when it was received.



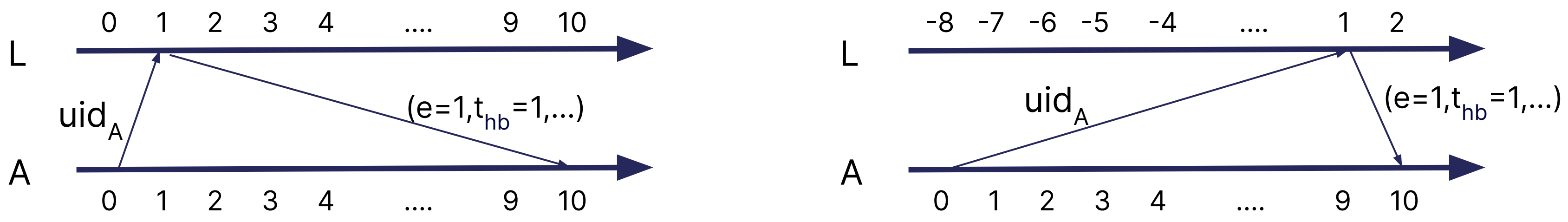
$\delta = 10 - 1 = 9$ → A will not drop out before $1 + 9 + \Delta_{live} = 20$ (assume $\Delta_{live} = 10$)

Understanding Liveness

- Participants maintain an upper bound δ on the offset between their clock and their leader's
- They drop out if they detect that the last heartbeat they received was generated more than Δ_{live} earlier, i.e. if:

$$t_{hb} + \delta + \Delta_{live} > t_{now}$$

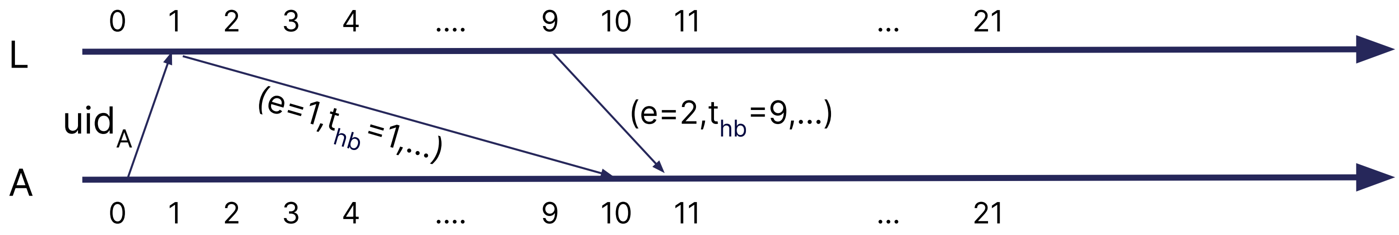
- The offset δ is the minimum difference between the time t_{hb} indicated in any heartbeat and the local time when it was received.



$\delta = 10 - 1 = 9$ → A will not drop out before $1 + 9 + \Delta_{live} = 20$ (assume $\Delta_{live} = 10$)

Understanding Liveness

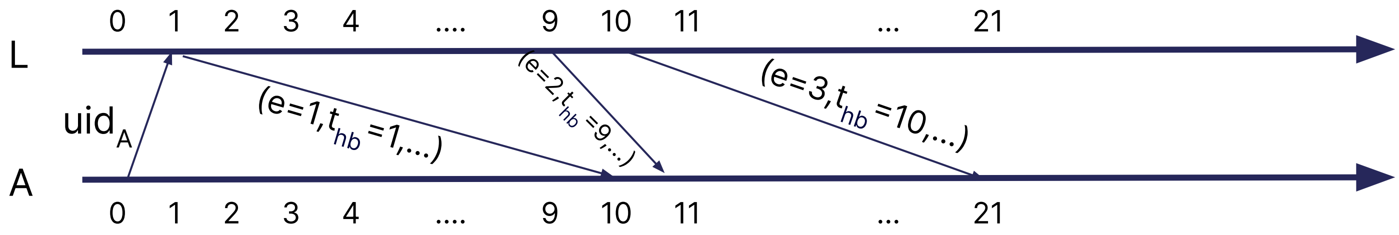
(Δ_{live} = 10)



δ = min(10-1, 11-9) = 2 → A will not drop out before 9 + 2 + Δ_{live} = 21

Understanding Liveness

(Δ_{live} = 10)



δ = min(10-1, 11-9, 21-10) = 2 → A will not drop out before 10 + 2 + Δ_{live} = 22

Note: For A's first leader, the difference between δ and the actual offset is at most Δ_{live}

Understanding Liveness

Theorem (informal): Zoom's LLCGKA (as described here) achieves liveness with slack

$$\textit{liveness-slack} = \min(n \cdot \Delta_{live}, t_{now} - t_{join}) + \Delta_{live}$$

where **n is the number of distinct leaders** encountered by the participant so far, **as long as they are all honest.**

Understanding Liveness

Theorem (informal): Zoom's LLCGKA (as described here) achieves liveness with slack

$$\textit{liveness-slack} = \min(n \cdot \Delta_{live}, t_{now} - t_{join}) + \Delta_{live}$$

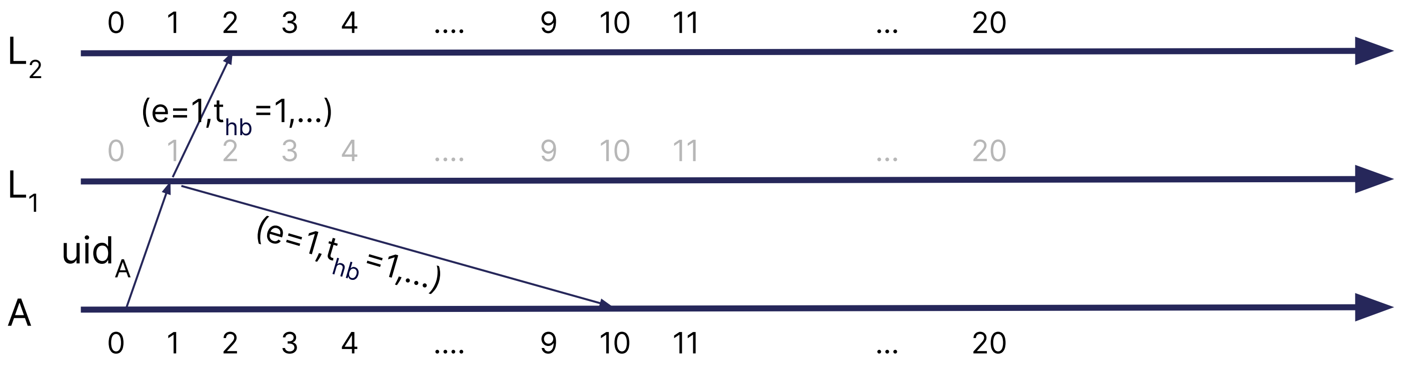
where **n is the number of distinct leaders** encountered by the participant so far, **as long as they are all honest.**

- Zoom's is the first video communications protocol with formal liveness guarantees!

Can we do better?

Example 1: δ degrades by Δ_{live} on leader change

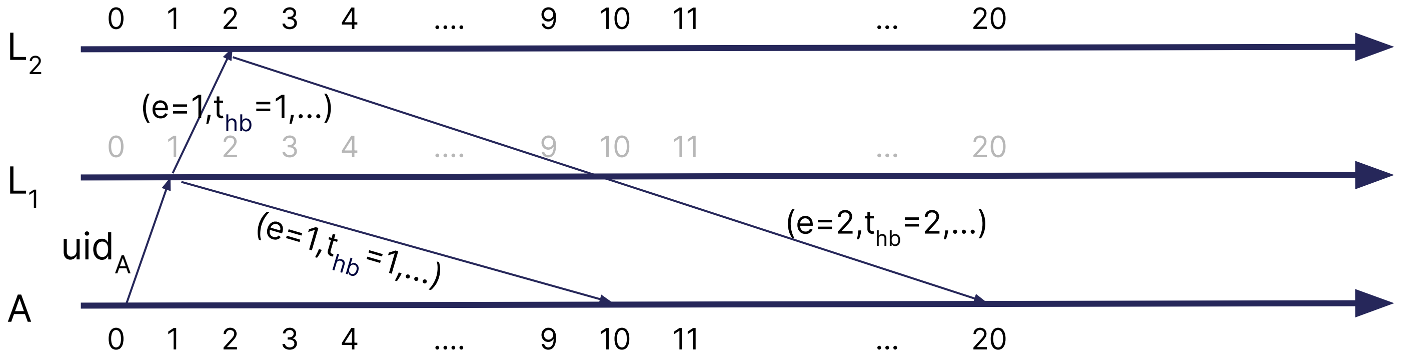
(Δ_{live} = 10)



δ_1 = 10-1 = 9 → A will not drop out before 1 + 9 + Δ_{live} = 20

Example 1: δ degrades by Δ_{live} on leader change

(Δ_{live} = 10)



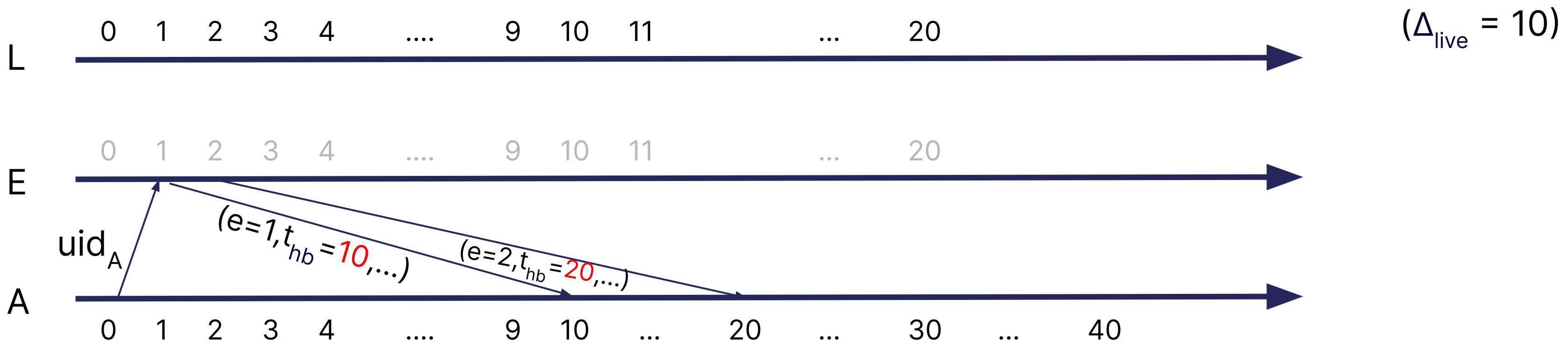
δ_1 = 10-1 = 9 → A will not drop out before 1 + 9 + Δ_{live} = 20

δ_2 = 20-2 = 18 → A will not drop out before 2 + 18 + Δ_{live} = 30

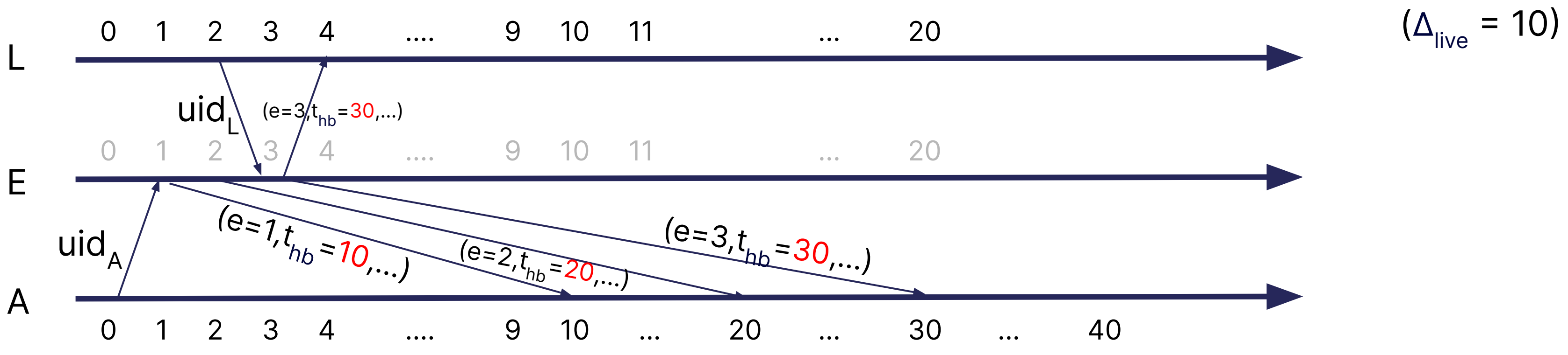
However, A and L_2 are synchronized (real offset is 0)!

Example 2: δ degrades arbitrarily if past leader is evil

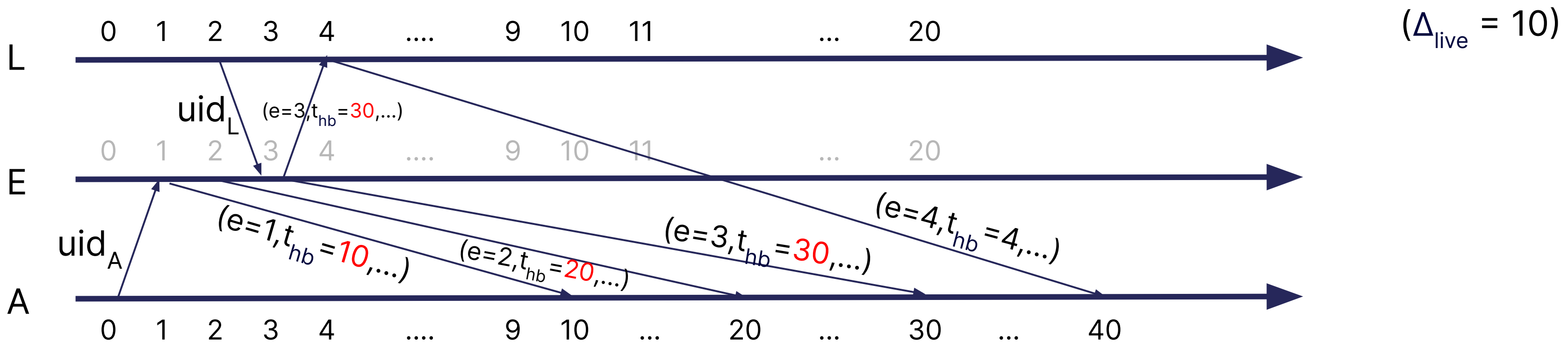
Example 2: δ degrades arbitrarily if past leader is evil



Example 2: δ degrades arbitrarily if past leader is evil



Example 2: δ degrades arbitrarily if past leader is evil



δ_L = 40-4 = 34 but A and L are actually synchronized

Proposal 1: optimistically leverage clock synchronicity

- In most cases, participants have well synchronized clocks. This proposal achieves slack:

$$\textit{liveness-slack} = \min(\text{offset}_{L \rightarrow P},\ n \cdot \Delta_{live},\ t_{now} - t_{join}) + \Delta_{live}$$

Proposal 1: optimistically leverage clock synchronicity

- In most cases, participants have well synchronized clocks. This proposal achieves slack:

$$\textit{liveness-slack} = \min(\text{offset}_{L \to P},\ n \cdot \Delta_{live},\ t_{now} - t_{join}) + \Delta_{live}$$

- Main ideas:
  - Participants maintain both a lower bound δ^{min} and an upper bound δ^{max} on the offset with their leader
  - When computing whether to drop out or not, participants correct the timestamp from the leader only if they are sure that it improves the estimate.

Proposal 1: optimistically leverage clock synchronicity

- In most cases, participants have well synchronized clocks. This proposal achieves slack:

$$\textit{liveness-slack} = \min(\text{offset}_{L\rightarrow P}, n \cdot \Delta_{live}, t_{now} - t_{join}) + \Delta_{live}$$

- Main ideas:
  - Participants maintain both a lower bound δ^{min} and an upper bound δ^{max} on the offset with their leader
  - When computing whether to drop out or not, participants correct the timestamp from the leader only if they are sure that it improves the estimate.

- Advantages:
  - Great liveness when clocks are synchronized (at the expense of correctness if they aren't)
  - No additional interaction
  - Still depends on previous leaders being honest

Proposal 2: Additional interaction

- Each participant sends to the server an unpredictable nonce, at regular intervals

- The first cmKEM message from a new leader must include the latest nonce (or the one before).
    - This ensures that the cmKEM message is recent

Proposal 2: Additional interaction

- Each participant sends to the server an unpredictable nonce, at regular intervals
- The first cmKEM message from a new leader must include the latest nonce (or the one before).
    - This ensures that the cmKEM message is recent

Theorem (informal): Zoom's LLCGKA with the improvement above achieves constant liveness slack

$$\textit{liveness-slack} = \min(\min(3,n) \cdot \Delta_{live}, t_{now} - t_{join}) + \Delta_{live}$$

regardless of whether past leaders were corrupted.

Proposal 2: Additional interaction

- Each participant sends to the server an unpredictable nonce, at regular intervals
- The first cmKEM message from a new leader must include the latest nonce (or the one before).
  - This ensures that the cmKEM message is recent

Theorem (informal): Zoom's LLCGKA with the improvement above achieves constant liveness slack

$$\textit{liveness-slack} = \min(\, \min(3,n) \cdot \Delta_{live}\, ,\, t_{now} - t_{join}\,) + \Delta_{live}$$

regardless of whether past leaders were corrupted.

- Deployed since Zoom Meetings v5.13!
  (**Note**: This improvement is now available in v.5.13 of the client, which modifies the design described above by varying the frequency of posting nonces depending on the number of parties. The full paper will describe this update.)

Conclusions

- Zoom's updated protocol achieves a very small liveness slack in our model!

- (E2E) Secure Group Video communications have unique requirements, benefit from specialized solutions.
  Future directions:
  - Insider security
  - Leaderless protocols
  - Expand model to Include the whole meeting (e.g. data streams)

- Analyzing real world protocols (academia/industry collaborations) is useful!

Thank you

Zoom Cryptography Whitepaper: https://github.com/zoom/zoom-e2e-whitepaper

Connect With Us

@amarcedone

@zoom_us | blog.zoom.us