Discord DAVE Protocol Design Review

Cryptographic Design Review

November 16, 2023

Prepared for:
Maxime Nay
Discord

Prepared by: **Fredrik Dahlgren and Tjaden Hess**

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on Twitter and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact, or email us at info@trailofbits.com.

Trail of Bits, Inc.
228 Park Ave S #80688
New York, NY 10003
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project.

Table of Contents

Project Summary

Contact Information

The following project manager was associated with this project:

Anne Marie Barry, Project Manager
annemarie.barry@trailofbits.com

The following engineering director was associated with this project:

Jim Miller, Engineering Director, Cryptography
james.miller@trailofbits.com

The following consultants were associated with this project:

Fredrik Dahlgren, Consultant
fredrik.dahlgren@trailofbits.com

Tjaden Hess, Consultant
tjaden.hess@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
September 21, 2023	Pre-project kickoff call
October 2, 2023	Status update meeting #1
October 10, 2023	Delivery of report draft
October 10, 2023	Report readout meeting
October 17, 2023	Delivery of comprehensive report

Executive Summary

Engagement Overview

Discord engaged Trail of Bits to review the design of its new end-to-end encrypted (E2EE) RTC protocol DAVE. The protocol aims to provide end-to-end encrypted voice and video calls for Discord users. The design is based on the MLS authenticated key exchange mechanism and uses encrypted WebRTC for the media transport.

A team of two consultants conducted the review from September 25 to October 6, 2023, for a total of four engineer-weeks of effort. Our testing efforts focused on ensuring that the integration with MLS and the media encryption specified by the DAVE protocol satisfied the protocol's security requirements. The work was performed with full access to protocol design documentation, the `mlspp` library source code, and the Discord monorepo.

Additionally, we performed a fix review to assess the implemented fixes for the issues described in this report. Out of the 11 issues identified during the engagement, 10 issues were completely addressed by the Discord team. The results from the fix review are described in appendix C.

Observations and Impact

At the time of the review, the Discord end-to-end encrypted RTC protocol was in development and not fully specified. Findings in this report represent plausible security issues that may arise in a fully implemented version of the protocol, based on the documentation available at the time of the review.

We identified two potential issues related to the validation of MLS credentials and key packages inside the `mlspp` library (TOB-DISCE2EE-2 and TOB-DISCE2EE-3). These validations must be explicitly enforced by the Discord E2EE RTC protocol.

Our review also uncovered two issues related to the proposed MLS group evolution mechanisms, which could potentially allow malicious users to prevent the addition or removal of other users (TOB-DISCE2EE-1 and TOB-DISCE2EE-5).

Additionally, we found two issues related to the proposed user interface and end-to-end verification flow that would make it harder for users to determine whether their sessions are secure against eavesdroppers (TOB-DISCE2EE-6 and TOB-DISCE2EE-7).

Finally, we discovered two issues related to the DAVE transport encryption scheme. We found that the interaction between codec-specific packetization and frame encryption is complex and exposes a large attack surface (TOB-DISCE2EE-10 and TOB-DISCE2EE-11). We recommend a thorough examination of this interaction in a future implementation review.

Overall, we found the protocol to be well designed, using modern protocols and primitives for continuous group key agreement and transport encryption. Care was taken to identify and mitigate a number of different threat scenarios, and these risks and their mitigations are also described in the protocol design document. The weakest component of the current design is the specification of the codec-aware media frame encryption, where we see a real risk of sensitive data being left unencrypted by a future implementation.

Recommendations

Based on the codebase maturity evaluation and findings identified during the security review, Trail of Bits recommends that Discord take the following steps prior to deploying the protocol in a production environment:

- **Remediate the findings disclosed in this report.** Many of the findings in this report constitute a severe risk to either the confidentiality, integrity, or availability of user data. For this reason, we recommend that these findings be addressed as part of a direct remediation before the protocol is deployed.

- **Drive MLS group evolution from the Discord signaling server.** The proposed implementation relies on group members to issue MLS proposals and commits. This allows multiple degrees of freedom that malicious users can misuse to cause denial-of-service conditions and other unexpected behavior. By requiring all MLS proposals to be submitted by the centralized signaling server and by validating commits against a group state tracked by the server, this attack surface can be minimized.

- **Review and monitor the codec-dependent encryption for correctness.** The proposed implementation of the codec-dependent selective encryption is brittle and prone to disclose sensitive data to privileged network nodes, such as the selective forwarding unit (SFU). Initial findings indicate that AV1- and H.264-encoded frames may disclose sensitive metadata about the encoded frame or even disclose the encoded frame itself in some cases.

  We recommend that Discord review the implementation of the WebRTC packetizer and depacketizer to ensure that the unencrypted portions of the frame are required by the WebRTC implementation to decompose and then reconstruct the frame and to ensure that everything else is either stripped or encrypted by the protocol.

  We also recommend that the Discord team set up integration tests to ensure that emitted media frames conform to the expected format and that sensitive information about the media stream is not left unencrypted on any of the supported platforms.

- **Move to packet-based end-to-end encryption if the requisite APIs become available.** The RTP generic header and dependency descriptor extensions are currently not exposed by WebRTC APIs available to web clients. If these become available in future versions of the API, it would be possible to move codec headers to an RTP header extension and make the transport encryptor codec-unaware by encrypting individual RTP packets rather than encoded frames. This would simplify the transport encryption protocol and mitigate the risk of plaintext disclosure. It would also avoid the risks involved with running the RTP depacketizer on unauthenticated input data. We recommend that the Discord team monitor the WebRTC APIs supported by popular browsers and update the protocol to packet-based encryption when the necessary APIs become available.

- **Perform an in-depth code review once the protocol has been implemented.** This engagement did not include a full review of the implemented protocol. A fully implemented version of the end-to-end encryption system will contain security-relevant edge cases and nuances not captured by the high-level protocol description. We recommend an especially thorough review of the DAVE protocol's symmetric encryption system and of MLS commit validations.

Finding Severities and Categories

The following tables provide the number of findings by severity and category.

EXPOSURE ANALYSIS

Severity	*Count*
High	4
Medium	5
Low	1
Informational	1
Undetermined	0

CATEGORY BREAKDOWN

Category	*Count*
Authentication	1
Cryptography	4
Data Validation	5
Denial of Service	1

Project Goals

The engagement was scoped to provide a design review of the Discord E2EE RTC protocol. Specifically, we sought to answer the following non-exhaustive list of questions:

- How are the MLS authentication and delivery services implemented, and what security properties do they provide?
- Which types of MLS proposals are allowed, and how are they validated?
- Are access controls in place to prevent users from adding or removing other members from an existing group?
- Which group members may issue an MLS commit?
- How are invalid MLS commits handled, and how are valid MLS commits agreed upon?
- Does the protocol protect against MLS commit spam?
- Which types of identities are supported by the protocol?
- How are identities authenticated by Discord and by peers?
- How are credential revocation and expiration handled by the protocol?
- How is post-compromise security guaranteed by the protocol?
- Are transport encryption keys updated with each new MLS epoch?
- Are transport encryption keys derived securely?
- Does the system properly protect against AES-GCM key wear-out?
- How are unencrypted RTP and media codec headers authenticated?
- How is misuse reporting implemented?
- How does the key agreement protocol resist machine-in-the-middle and impersonation attacks?

Project Targets

The engagement involved a review of the targets listed below.

RTC E2EE protocol

Version	September 21, 2023
Type	Design document

Discord monorepo

Repository	https://github.com/discord/discord
Version	`a02db9bc34e6bbc5b1eb1e0149d7b056828afacc`
Type	C++
Platform	Multiple

mlspp

Repository	https://github.com/cisco/mlspp
Version	`8a010ef9ce9f1bdd8b79a11ee1c089440a3f63fb`
Type	C++
Platform	Multiple

Project Coverage

This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches included the following:

- **MLS-based continuous key agreement.** We performed an in-depth review of the MLS-based key agreement protocol defined by the design document. This review was based both on the design document itself and on external references like the MLS RFC and the MLS architecture document. We focused on how MLS messages are validated by group members and by the delivery service (DS) and how identities and credentials are managed by the authentication service (AS). We also investigated ways for individual members to prevent the protocol from making progress.

- **DAVE transport encryption.** We reviewed the design of the DAVE protocol's media encryption, paying special attention to the selective encryption of encoded media frames. This part of the engagement entailed an extensive review of external documentation for the different codecs involved to understand whether the proposed design could either disclose sensitive data about the media stream or be vulnerable to tampering or impersonation attacks.

Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. The following list outlines the coverage limitations of the engagement and indicates system elements that may warrant further review:

- We were unable to obtain conclusive results on the exact H.264 network abstraction layers (NALs) and AV1 open bitstream units (OBUs) that must be encrypted to fully protect the confidentiality of the media stream. That said, initial results indicate that the current design may leave some media frames unencrypted (TOB-DISCE2EE-10).

- We used the provided codebase as a reference to better understand the specification but did not perform an in-depth review of the implemented code itself.

Summary of Findings

The table below summarizes the findings of the review, including type and severity details.

ID	Title	Type	Severity
1	Commit spam could prevent group updates	Denial of Service	**Medium**
2	Commit leaf nodes are not validated by the mlspp library	Data Validation	**Medium**
3	The mlspp library does not validate key package lifetimes	Data Validation	**Informational**
4	Web client may allow a malicious server to conduct a machine-in-the-middle attack on the session	Cryptography	**High**
5	Committing members may fail to send Welcome messages	Data Validation	**Medium**
6	Users could decrypt messages after being removed from the call UI	Data Validation	**High**
7	Key fingerprint verification is vulnerable to partial preimage attacks	Authentication	**Medium**
8	Abuse reporting mechanism is vulnerable to message forgery	Data Validation	**Medium**
9	DAVE's media encryption provides weakened forward secrecy guarantees	Cryptography	**Low**
10	The protocol may fail to encrypt AV1-encoded media frames	Cryptography	**High**
11	Video frame header length and header data are not authenticated	Cryptography	**High**

Detailed Findings

1. Commit spam could prevent group updates	
Severity: **Medium**	Difficulty: **Undetermined**
Type: Denial of Service	Finding ID: TOB-DISCE2EE-1
Target: E2EE RTC protocol	

Description

The section on commit ordering in the design document states that the signaling server will broadcast the first commit seen for each epoch to all members of the group. If a committing member starts spamming the group with new commits, this could effectively prevent other members from committing, which would prevent meaningful group updates and cause unnecessary load on the symmetric encryption mechanism.

The section on key rotation mentions rate limiting empty commits to avoid unnecessary key updates. This may also mitigate this issue, depending on how the system handles `Add` and `Remove` proposals. The documentation states the following restrictions on `Add` and `Remove` proposals:

> *Commits including add proposals are only valid if the added member is present in the member list received from the delivery service.*
>
> *Commits including removal proposals are only valid if the removed member is removed from the member list received from the delivery service.*

However, if a member can include `Add` and `Remove` proposals for members who were previously added or removed from the group, this could still be an issue.

Exploit Scenario

A malicious user with a privileged network position spams the network with commits containing `Add` and `Remove` proposals for previous members (who have already been added and removed once from the group). Since the server broadcasts only the first commit obtained in each epoch, this effectively prevents anyone else from committing to the group. In particular, new members cannot be added, and the malicious user cannot be removed.

Recommendations

Short term, disallow empty commits to the group when there are outstanding `Add` or `Remove` proposals to the group.

Long term, disallow group members from submitting any proposals and instead use external proposals to add and remove members. Require the DS to broadcast only commits that contain the full set of current proposals, and validate the commits to ensure that all credentials are authentic. Key rotation may be implemented by external `Psk` proposals. For the DS to validate handshake messages, they must be sent as unencrypted `PublicMessage` objects. These handshake messages should be point-to-point encrypted and authenticated using TLS.

2. Commit leaf nodes are not validated by the mlspp library	
Severity: **Medium**	Difficulty: **Medium**
Type: Data Validation	Finding ID: TOB-DISCE2EE-2
Target: `src/state.cpp` in `mlspp`	

Description

A committing member can update their own leaf node with an empty commit. The new leaf node is validated by the `mlspp` library, but the library does not check that the credential and signing key in the new leaf node match those in the replaced node. This means that members can update their identities in the ratchet tree.

```
bool
State::valid(const LeafNode& leaf_node,
             LeafNodeSource required_source,
             std::optional<LeafIndex> index) const
{
  // Verify that the credential in the LeafNode is valid as described in Section
  // 5.3.1.
  // XXX(RLB) N/A, no credential validation in the library right now

  // ...

  return (signature_valid && supports_group_extensions && correct_source &&
          mutual_credential_support && supports_own_extensions);
}
```

Figure 2.1: Credentials are not validated by the `mlspp` library.
(`mlspp/src/state.cpp#L1442-L1503`)

If the ratchet tree is included with the `Welcome` message to new members, the updated identity will be propagated to future members joining the group.

Exploit Scenario

A malicious user registers persistent keys for two different identities, S_1 and S_2, with Discord. During a session where the user is a committing member as S_1, they send a commit replacing their leaf node with a new node containing S_2, along with the corresponding signing key. This introduces a risk that joining members will misidentify the malicious user as S_2, whereas previous members in the group still see the user as S_1.

Recommendations

Short term, validate commit messages to ensure that existing group member credentials are not replaced by a commit.

Long term, consider performing an internal security review of the `mlspp` library to better understand the maturity of the codebase.

3. The mlspp library does not validate key package lifetimes

Severity: **Informational**	Difficulty: **Not Applicable**
Type: Data Validation	Finding ID: TOB-DISCE2EE-3
Target: E2EE RTC protocol	

Description

MLS key packages contain a `lifetime` field that determines the validity period of the key package. The `mlspp` library does not validate these values.

```
// TODO(RLB) Verify the lifetime field
```

Figure 3.1: Missing key package lifetime validation (`mlspp/src/state.cpp#1475–1477`)

The current protocol does not cache key packages beyond a single session, so the lifetime field may be set to the maximum value. However, if future versions of the protocol use cached key packages, it will be important to add lifetime validation on top of the validations that `mlspp` performs.

Recommendations

Short term, set the key package lifetime `not_before` and `not_after` parameters to the maximum allowed time span to avoid key packages expiring.

Long term, verify which validations specified by RFC 9420 are included in `mlspp` and add any missing checks to the Discord codebase.

4. Web client may allow a malicious server to conduct a machine-in-the-middle attack on the session

Severity: **High**	Difficulty: **High**
Type: Cryptography	Finding ID: TOB-DISCE2EE-4
Target: E2EE RTC protocol	

Description
The Discord web client stores key material in the browser, making it accessible to any JavaScript code served by Discord or an adversary with access to a TLS certificate for `discord.com`.

Unlike mobile or desktop clients, the web client code is served to the user every time they open the app. The delivered code may differ depending on the user requesting it, and it is not signed with a code signing key, unlike mobile apps delivered by app stores. Web app users are thus at increased risk of impersonation and machine-in-the-middle attacks due to malicious code.

Exploit Scenario
A nation-state attacker owns a TLS root certificate and issues a fraudulent certificate for `discord.com`. The attacker intercepts traffic from specific users and replaces the Discord application with a backdoored version. The attacker can then eavesdrop on and tamper with all further communication by the users.

Recommendations
Short term, warn users that the security guarantees offered by the web client are weaker than those offered by the mobile or desktop applications.

Long term, consider implementing a browser extension to enable code signature verification and code fingerprint verification. For an example of how this could be done, see this blog post on Meta's Code Verify browser plugin.

5. Committing members may fail to send Welcome messages	
Severity: **Medium**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-DISCE2EE-5
Target: E2EE RTC protocol	

Description

To invite a new member to an MLS group, a committing member must send a `commit` containing an `Add` proposal. Assuming the protocol uses public handshake messages, a signaling server can ensure that this takes place correctly. However, after all users process the commit, the committing member must send a `Welcome` message to the joiner so the joiner can initialize their ratchet tree state and begin decrypting messages. This `Welcome` message is encrypted, so the signaling server cannot determine whether it is correctly formed.

One possible solution to this issue is to require each joiner to confirm to the DS that they have received a valid `Welcome` message. After some timeout, the DS could then orchestrate a retry of the addition. However, the joiner may lie about receiving an invalid `Welcome` message, and when multiple joiners are added in a single epoch, they may disagree as to the validity of the `Welcome` message. This adds complexity to the new member addition process and makes a denial of service more likely.

Exploit Scenario

A malicious user wants to prevent new members from being added to a group. When the DS announces a new member, the user commits the `Add` proposal, but never sends a `Welcome` message. The joining user is then unable to decrypt messages and enter the call.

Recommendations

Short term, have the signaling server monitor commits adding new members to ensure the committer sends a corresponding `Welcome` message. If no `Welcome` is sent, the server should remove the committer from the group of committing members and request a new `Welcome` message from the group. If the new member rejects the `Welcome` message as invalid, the server should simply request a new `Welcome` from a random committing member of the group.

Long term, consider using the `ExternalInit` flow to add new members to a group. The DS serves a cached copy of the current `GroupInfo` struct to the new member, who then submits an external commit to the group, adding themselves. This removes the need for any existing group member to coordinate the addition of new members.

6. Users could decrypt messages after being removed from the call UI	
Severity: **High**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-DISCE2EE-6
Target: E2EE RTC protocol	

Description

The client UI listing the users that are connected to a given session is controlled by the voice state of the corresponding users. This state is not directly controlled by `Add` or `Remove` messages from the signaling server or the corresponding group epoch updates. In practice, this means that a user may be removed from the UI before the group epoch is updated, allowing the user to continue to decrypt messages to the session, even though they appear disconnected from the point of view of other members.

Such a user would need to intercept WebRTC messages between other users after being removed from the SFU, which would be feasible as a passive network adversary. Additionally, an active network adversary may be able to impersonate current group members using their knowledge of other members' sender keys.

Exploit Scenario

Mallory, a malicious network adversary convinces a member of a voice group to add her to the group by impersonating a desired group member. Upon realizing that Mallory is not the intended group member, the existing members kick her from the group. Mallory is removed from the Discord UI and the group members continue discussing sensitive information. By delaying handshake messages, Mallory keeps the MLS epoch from advancing and uses her network position to intercept and decrypt encrypted frames, thus spying undetected on the conversation.

Recommendations

Short term, ensure that users are not dropped from the list of participants until the symmetric keys used to encrypt application data are updated. Until the group epoch is updated and a new symmetric key is derived, the UI may display users as grayed out.

Long term, ensure that the MLS ratchet tree state and identities are auditable by the end user and that the UI clearly reflects what identities can decrypt packets from a given E2EE session.

7. Key fingerprint verification is vulnerable to partial preimage attacks

Severity: **Medium**	Difficulty: **High**
Type: Authentication	Finding ID: TOB-DISCE2EE-7
Target: E2EE RTC protocol	

Description

The Discord client will allow users to verify the signing keys associated with a particular device pair. This is done by computing two repeated SHA-512 hashes over the user's identity and the public key associated with the device, resulting in a 64-byte fingerprint that consists of two concatenated SHA-512 digests. These fingerprints may then be verified out of band (e.g., by meeting and comparing fingerprints).

However, it is well known that manual verification of long hexadecimal values is error-prone. This could allow an attacker to impersonate other users by creating a key pair with a fingerprint that is similar enough to the impersonated user. (The DEA was subject to an attack of this type earlier this year, which led to the loss of 50,000 USD.)

Exploit Scenario

A nation-state attacker who can conduct a machine-in-the-middle attack on Alice's connection generates a signing key whose fingerprint has the same first four bytes and last four bytes as Alice's fingerprint. The attacker registers their key for Alice's account and impersonates Alice in calls. When Alice and Bob meet to verify their signing key fingerprints out of band, Bob computes his fingerprint using the attacker's key. Alice and Bob fail to notice this because they only compare the first and last digits of the fingerprint.

Recommendations

Short term, encode the fingerprint as a QR code to allow users to verify it automatically if their device has a camera that can scan QR codes. Use the manual verification process as a fallback only, and direct users to use the QR code whenever possible.

Long term, review user studies on fingerprint verification and choose a format (e.g., sentence-based fingerprints) that is less susceptible to partial preimage attacks.

8. Abuse reporting mechanism is vulnerable to message forgery

Severity: **Medium**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-DISCE2EE-8
Target: E2EE RTC protocol	

Description

The current iteration of the protocol does not contain a mechanism for reporting abusive content. However, the Discord team is planning to implement this in a future version of the protocol by allowing users to capture voice and video streams and submit these to the Discord Trust and Safety team to review. Since there is no way to authenticate the captured streams, this mechanism is vulnerable to forgery attacks where the reporter creates a fake media stream and submits this as a report against another Discord user.

Exploit Scenario

Alice uses an online AI voice and video generator to create a fake recording of a video call with Bob. She submits this to Discord as proof of abusive behavior, and as a result, Bob is banned from the platform.

Recommendations

Short term, investigate solutions based on message franking (see the Abuse Reporting section in this white paper on Facebook Messenger), where the sender computes a binding commitment to the plaintext frames, which is either symmetrically or asymmetrically authenticated by the server. This offloads any storage requirements to the receiving client, who needs to store media and franking tags only if an abuse report is actively being recorded. To reduce the bandwidth requirements of the system, franking tags may be computed over a number of plaintext frames.

Long term, revisit the literature on message franking in the future to see if new and more lightweight schemes are proposed for the WebRTC setting.

9. DAVE’s media encryption provides weakened forward secrecy guarantees	
Severity: **Low**	Difficulty: **High**
Type: Cryptography	Finding ID: TOB-DISCE2EE-9
Target: E2EE RTC protocol	

Description

The DAVE protocol updates the symmetric encryption key whenever MLS transitions to a new epoch. This provides forward secrecy and post-compromise security between different MLS epochs. The MLS protocol also includes a symmetric ratchet mechanism called a secret tree, which can be used to regularly update symmetric keys within a given epoch. This symmetric ratchet also provides forward secrecy guarantees within a single epoch. The current protocol design does not use the MLS symmetric ratchet, which means that the forward secrecy guarantees provided by the design are weaker than they should be. This is particularly true for long-lived epochs where no users join or leave the session.

Exploit Scenario

Eve, an attacker, can store encrypted frames from Alice, who is participating in a group call. At some point, Eve compromises Alice's symmetric key. Since the protocol does not ratchet symmetric keys, Eve can go back and decrypt all of Alice’s messages sent during the current epoch.

Recommendations

Short term, use the MLS secret tree to derive symmetric encryption keys and nonces. The `mlspp` library does not expose the secret tree directly, but it is possible to instantiate a new secret tree (as an `mlspp` `GroupKeySource` struct) based on the `exporter_secret` key. The secure frame nonce can be repurposed as two 16-bit epoch and generation counters to ensure that the sending and receiving parties both agree on the current epoch and ratchet generation.

10. The protocol may fail to encrypt AV1-encoded media frames	
Severity: **High**	Difficulty: **High**
Type: Cryptography	Finding ID: TOB-DISCE2EE-10
Target: E2EE RTC protocol	

Description

In the current design of the DAVE protocol, AV1-encoded frames are only partially encrypted. More precisely, the encryptor assumes that each frame contains at most one frame OBU and that this is the last OBU in the frame. The frame OBU payload is then encrypted, leaving the OBU header and the remaining OBUs unencrypted.

However, according to section 6.9 of the AV1 specification, a frame OBU is simply a more compact representation of a frame header OBU, followed by a tile group OBU, and the same frame could, in principle, be represented either way:

> *A frame OBU consists of a frame header OBU and a tile group OBU packed into a single OBU.*
>
> *The intention is to provide a more compact way of coding the common use case where the frame header is immediately followed by tile group data.*

If the AV1 encoder uses separate frame header and tile group OBUs to represent the frame, the OBUs would not be encrypted by the sender under the current design. This would immediately disclose the current frame of the sender's media stream to the SFU.

The section on ordering OBUs (section 7.5 in the AV1 specification) seems to require that frame OBUs and tile group OBUs be placed last in the encoded frame, but the section also mentions that *"Some applications may choose to use bitstreams that are not fully conformant to the requirements described in this section."* In practice, this means that nonconformant encoders may disclose frame and tile group OBUs.

Exploit Scenario

Alice's Discord client uses AV1 to encode her media stream. The stream is encoded using separate frame header and tile group OBUs, causing Alice's client to send the entire media stream in plaintext to the server, thus breaking the protocol's end-to-end confidentiality guarantees.

Recommendations

Short term, test which types of OBUs are emitted by the different Discord client implementations to ensure that the media stream is encrypted correctly by each client. At least the following OBUs should be encrypted to preserve the media stream's confidentiality:

- **Frame header OBUs:** Contain metadata describing how the current frame has changed compared to previous frames
- **Tile group UBUs:** Contain tile data associated with the frame
- **Tile list OBUs:** Contain tile data similar to a tile group OBU, together with additional headers allowing the decoder to process parts of the current frame

(Tile list OBUs are currently dropped by the WebRTC packetizer according to the documentation provided by Discord.)

Long term, encrypt frame header, tile group, and frame OBUs independent of their position in the encoded frame. Review the AV1 specification in detail to ensure that user data is not disclosed by the remaining OBUs.

References

- A Technical Overview of AV1
- AV1 Bitstream & Decoding Process Specification

11. Video frame header length and header data are not authenticated	
Severity: **High**	Difficulty: **High**
Type: Cryptography	Finding ID: TOB-DISCE2EE-11
Target: E2EE RTC protocol	

Description

Because packetization and depacketization rely on values located in the frame payload, a contiguous initial portion of each frame is left unencrypted. Currently, the design does not include these unencrypted headers in the associated data of AES-GCM. A malicious SFU could thus modify these headers undetected.

Of particular concern are the H.264/H.265 and AV1 frames, which contain variable-length unencrypted header data. Because the unencrypted header frame size value is not authenticated, a malicious SFU can replace it with a larger value. The SFU can then include additional unencrypted video-coding layers, which will be copied directly into the output frame. Additionally, if nonce replay protections are not present on the decryptor side, the SFU could use the nonce and authentication tag from a fully unencrypted header-only frame to forge arbitrary plaintext frames.

```
if (codecUnencryptedHeaderBytes) {
   // copy the unencrypted header to the output frame
   memcpy(frame.data(), encryptedFrame.data(), codecUnencryptedHeaderBytes);
}
```

Figure 11.1: Unauthenticated data is directly included in the visible frame.
(discord/discord_common/native/secure_frames/decryptor.cpp#66–69)

Allowing the SFU to include or replace content in user media frames would allow impersonation of users by the Discord server, even when MLS end-to-end persistent identity verification is in use. Additionally, unauthenticated codec headers present a large surface area for exploitation of implementation flaws in the underlying codecs.

Furthermore, lack of header authentication may open clients to adaptive attacks against confidentiality. For example, forged video-coding layers may reference elements defined in encrypted video-coding layers, disclosing information about the encrypted layers by observing differential failures in the decoder.

Exploit Scenario

A malicious Discord employee modifies the SFU to add video codec NALs that replace a particular user's broadcast stream with a pre-recorded video faked to look like a legitimate stream. Advanced users check the persistent identity fingerprint and are convinced that the stream is legitimate.

Recommendations
Short term, include all unencrypted components of media frames in the authenticated data of the underlying AEAD.

Long term, keep abreast of new extensions to the WebRTC APIs that would allow Discord to switch to a packet-based transport encryption mechanism. This would allow a future version of the protocol to encrypt media content and headers at the RTP packet level, which would allow the protocol to encrypt and authenticate the entire encoded frame in a way that would be invisible to the RTP packetizer and depacketizer.

A. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

Vulnerability Categories	
Category	Description
Access Controls	Insufficient authorization or assessment of rights
Auditing and Logging	Insufficient auditing of actions or logging of problems
Authentication	Improper identification of users
Configuration	Misconfigured servers, devices, or software components
Cryptography	A breach of system confidentiality or integrity
Data Exposure	Exposure of sensitive information
Data Validation	Improper reliance on the structure or values of data
Denial of Service	A system failure with an availability impact
Error Reporting	Insecure or insufficient reporting of error conditions
Patching	Use of an outdated software package or library
Session Management	Improper identification of authenticated users
Testing	Insufficient test methodology or test coverage
Timing	Race conditions or other order-of-operations flaws
Undefined Behavior	Undefined behavior triggered within the system

Severity Levels	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is small or is not one the client has indicated is important.
Medium	User information is at risk; exploitation could pose reputational, legal, or moderate financial risks.
High	The flaw could affect numerous users and have serious reputational, legal, or financial implications.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is well known; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of the system.
High	An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue.

B. Guidance on Using Short AES-GCM Tags

This appendix lists considerations that must be observed when using truncated AES-GCM tags. While NIST SP 800-38D permits the use of 32- and 64-bit authentication tags in some real-time applications, certain properties of AES-GCM make short tags dangerous in many situations.

We recommend GCM tag lengths of 64 bits at a minimum and at least 96 bits if possible. Other AEADs based on HMACs rather than polynomial MACs are more robust to use with truncated tags.

Two properties of AES-GCM, detailed in Authentication weaknesses in GCM, impact the security of schemes using truncated GCM tags:

1. **Long messages decrease the effort required to construct a forgery.** If an adversary observes a properly authenticated message of bit length 2^k, with tag length `t`, then the adversary can forge a different message with a probability of roughly $2^{(t - k)}$. As a concrete example, if an honest user sends a 128-KB packet using a 32-bit tag, then the adversary will need to send roughly 2^{22} messages before finding a successful forgery.

2. **Forgeries against truncated tags disclose information about the authentication key.** If an adversary can determine which forgery attempts are successful, they can use that information to increase the probability of future forgeries, eventually revealing the full authentication key.

NIST SP 800-38D: Appendix C specifies requirements and guidelines for using short tags; we summarize those relevant to the Discord DAVE protocol:

1. Do not reveal information about decryption success for individual packets. In particular, do not send retry requests for frames that fail authentication. Consider mitigating timing side channels so that externally visible timing behavior is similar for decryption success or failure.

2. Include as little encrypted/authenticated data as possible in each frame. Be aware of opportunities for adversaries, such as the SFU, to request the encryption or authentication of large frames.

3. Design the system to be robust against individual frame forgeries. Forgery of a single frame should result in only a minor change in the end user's media stream and should not cause system-wide effects.

4. Rotate sender keys frequently, preferably via symmetric derivation. Asymmetric key rotation via MLS commits may be selectively delayed by a malicious DS.

C. Fix Review Results

When undertaking a fix review, Trail of Bits reviews the fixes implemented for issues identified in the original report. This work involves a review of specific areas of the updated protocol design documentation, not comprehensive analysis of the entire system.

On November 9, 2023, Trail of Bits reviewed the fixes and mitigations implemented by the Discord team for the issues identified in this report. We reviewed each fix to determine its effectiveness in resolving the associated issue.

In summary, of the 11 issues described in this report, Discord has resolved 10 issues and has partially resolved the one remaining issue. For additional information, please see the Detailed Fix Review Results below.

ID	Title	Status
1	Commit spam could prevent group updates	Resolved
2	Commit leaf nodes are not validated by the mlspp library	Resolved
3	The mlspp library does not validate key package lifetimes	Resolved
4	Web client may allow a malicious server to conduct a machine-in-the-middle attack on the session	Resolved
5	Committing members may fail to send Welcome messages	Resolved
6	Users could decrypt messages after being removed from the call UI	Resolved
7	Key fingerprint verification is vulnerable to partial preimage attacks	Resolved
8	Abuse reporting mechanism is vulnerable to message forgery	Partially Resolved
9	Secure frames encryption provides weakened forward secrecy guarantees	Resolved

10	Secure frames may fail to encrypt AV1-encoded media frames	**Resolved**
11	Video frame header length and header data are not authenticated	**Resolved**

Detailed Fix Review Results

TOB-DISCE2EE-1: Commit spam could prevent group updates
Resolved. The DS has been added as an external sender and is the only one allowed to make proposals to the group. Commit messages are filtered by the DS, which accepts only commits containing all the currently open Add and Remove proposals made by the DS.

TOB-DISCE2EE-2: Commit leaf nodes are not validated by the mlspp library
Resolved. Both the DS and the individual members of the group now validate all commits to ensure that member credentials and signing keys are not updated by the commit.

The Discord team has also decided to perform an internal review of the `mlspp` library to identify any potential security issues in it. Any issues identified will be addressed and the fix will be merged upstream to `mlspp`.

TOB-DISCE2EE-3: The mlspp library does not validate key package lifetimes
Resolved. The DS now ensures that the key package fields `not_before` and `not_after` are set to the maximum allowed lifespan.

TOB-DISCE2EE-4: Web client may allow a malicious server to conduct a machine-in-the-middle attack on the session
Resolved. The Discord team will inform users that the Discord web client cannot offer the same security guarantees as native clients against machine-in-the-middle attacks.

TOB-DISCE2EE-5: Committing members may fail to send Welcome messages
Resolved. The DS now requires the committing member to include the `Welcome` message with the corresponding `Commit`. If a member fails to transmit `Welcome` messages, they may be removed from the group of committing members. If a `Welcome` message is reported as invalid by the joining member, the DS may request a new `Welcome` message from a different member of the group.

TOB-DISCE2EE-6: Users could decrypt messages after being removed from the call UI
Resolved. Official Discord clients will be able to review the current MLS group state to see if leaving members can still decrypt messages to the group. According to the section on UI considerations, this should also be reflected by the media session modal UI.

Additionally, the Discord team is exploring more long-term solutions where the MLS group membership state would be directly reflected by the client UI.

TOB-DISCE2EE-7: Key fingerprint verification is vulnerable to partial preimage attacks
Resolved. The fingerprint verification UI will use a QR code for automatic verification and word-based fingerprints as a manual fallback.

TOB-DISCE2EE-8: Abuse reporting mechanism is vulnerable to message forgery
Partially resolved. The section on abuse reporting in the design documentation now clearly states that a design based on message franking should be preferred as long as the impacts on bandwidth and storage are negligible. However, the abuse reporting mechanism will not be part of the initial release of the protocol, according to the Discord team. Since the details around the design of this feature were not determined at the time of the fix review, this issue is considered partially resolved.

TOB-DISCE2EE-9: Secure frames encryption provides weakened forward secrecy guarantees
Resolved. The secure frames protocol now uses an MLS secret tree initialized using the `exporter_secret` key to derive symmetric encryption keys.

TOB-DISCE2EE-10: Secure frames may fail to encrypt AV1-encoded media frames
Resolved. All OBU payloads are now encrypted independently of their position within the frame.

TOB-DISCE2EE-11: Video frame header length and header data are not authenticated
Resolved. All unencrypted data from the frame will now be included as additional data for the AEAD.

D. Fix Review Status Categories

The following table describes the statuses used to indicate whether an issue has been sufficiently addressed.

Fix Status	
Status	Description
Undetermined	The status of the issue was not determined during this engagement.
Unresolved	The issue persists and has not been resolved.
Partially Resolved	The issue persists but has been partially resolved.
Resolved	The issue has been sufficiently resolved.