White Paper

Cisco Spark Security and Privacy

Version 1.0 (June 2016)

Cisco® Spark is a cloud collaboration platform that provides messaging, calling, and meeting features. The Cisco Spark® application is a client app that connects to this platform and provides a comprehensive tool for teamwork. Users can send messages, share files, and meet with different teams, all in one place.

This white paper provides a security and privacy overview for the Cisco Spark Cloud and Cisco Spark Messaging.

The Cisco products, services, and features described in this document are in varying stages of development. While some of such Cisco products, services, and features currently exist, others are under development or planned for the future. Find additional information at www.cisco.com.

Cisco will have no liability for delay in the delivery or failure to deliver the products, services, or features set forth in this document.

Security and Privacy Challenges for Cloud Collaboration

One of the key benefits to enterprises consuming cloud services is the ability to leverage value-added features and functionality as quickly as the cloud service provider can deploy them. But for many cloud providers, “adding value” often means having full access to user data and content. For collaboration applications, most cloud providers directly access message, call, and meeting content in order to offer features like message search, content transcoding, or integration with third-party applications. Conversely, modern consumer collaboration services tend to be geared toward protecting consumer privacy by offering end-to-end encryption at the expense of value-add features.

Cisco Spark is the best of both worlds: an end-to-end encrypted cloud collaboration platform that offers enterprises the ability to choose what, if any, value-added integrations Cisco and third parties provide. Cisco Spark uses an open architecture for the secure distribution of encryption keys, allowing enterprises to gain exclusive control over the management of their encryption keys and the confidentiality of their data. This means that content is encrypted on the user client and remains encrypted until it reaches the recipient, with no intermediaries having access to decryption keys for content unless the enterprise explicitly chooses to grant such access.

While you can obtain additional features by granting explicit access to keys, we built end-to-end encryption into the fabric of Spark from the very beginning, which means many of the value-added features and functionality operate on encrypted data. Using innovative message indexing, permissions models, authentication flows, encryption, and deployment models, Spark supports features such as global search of content that was never decrypted in the Cisco Spark cloud.

Most cloud service providers claim to be secure because they encrypt data in transit between users’ devices and their servers, or between their own data centers. But encryption in transit is not enough to protect data from exposure to the cloud service provider itself. All connections to and from the Spark cloud are encrypted in transit—but we’ve taken Spark beyond, ensuring we only see user content where explicit access is granted.

Our commitment to providing a trusted service offering is not limited to protecting user content. Spark protects all data about users and usage using a combination of privacy tools and features that includes *obfuscated identity*[1], choice, and transparency. As with end-to-end encryption, we built these protections into the service from the start.

End-to-End Content Encryption

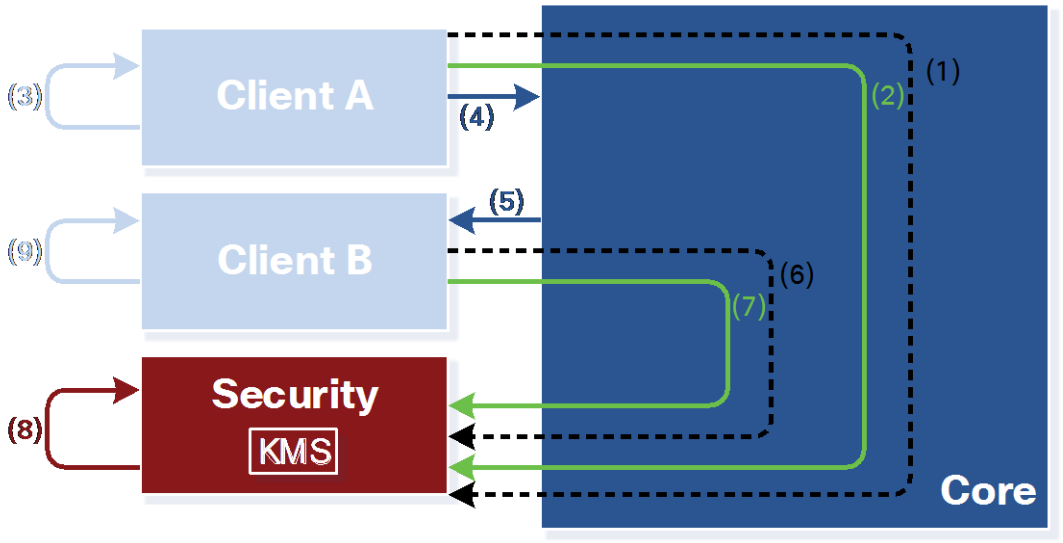
The cornerstone of end-to-end content encryption in the Cisco Spark Cloud is a component known as the Key Management Server (KMS). The KMS is responsible for creating, storing, authorizing, and providing access to the encryption keys that Spark clients use to encrypt and decrypt messages and files. End-to-end encryption in Spark is possible because of the architectural and operational separation between the KMS and the rest of the Spark Cloud. Think of them as being in separate realms, or trust domains, in the cloud: The KMS is in the *“Security Realm”* and all other component services that make up Spark are in the *Core.*

Communication with the KMS transits the Cisco Spark Cloud, but is also encrypted end-to-end, so that it is not readable by the Core, and is authenticated using access tokens not used elsewhere in the Cisco Spark Cloud. This model ensures appropriate access to encryption keys, while also guaranteeing that no Core services components can access those communications or the keys that the KMS stores. When operated by Cisco, the services in the Security Realm operate on separate infrastructure in a separate tenant. Security-conscious enterprise customers may choose to deploy the Security Realm services, including the KMS, on their own premises, the details of which are described in the next section.

[1] See Obfuscated Identity section

When a Spark user wants to send content to a Spark room, the user's client must first establish a secure channel between itself and the KMS, shown as **(1)** in Figure 1. To establish a shared secret key for a secure channel between the Client and the KMS, the Client and KMS perform an authenticated ephemeral Elliptic Curve Diffie–Hellman (ECDH) exchange. This exchange produces a symmetric shared key that can be used to securely exchange messages.

Figure 3. Client - KMS Communication in Cisco Spark



Authentication on this channel is required to ensure it is not possible for Cisco, or any other third party, to view or modify the information and keys that transit this channel or to act as a man-in-the-middle. The authentication mechanism leverages a public key infrastructure (PKI) certificate on the KMS, where the certificate contains a Common Name (CN) or Subject Alternative Names (SAN) entry matching the enterprise's domain name. Clients encrypt their half of the ECDH exchange to the KMS using the public portion of the KMS server certificate. ECDH responses from the KMS are signed using the private portion of the KMS's server certificate[2]. This is a slight modification to the ECDHE-RSA mechanism in that the client encrypts using the server's public certificate rather than signing using its own key, which means clients do not require certificates. However this, of course, means that clients must be able to authenticate the KMS's certificate. To do so, KMS certificates must use either a public certificate authority (CA) that is widely trusted on desktop and mobile devices or enterprises must be able to push their private CA root certificate to the end devices their users will use to access Cisco Spark. Pushing private CA certificates to clients is not handled by Cisco Spark, but Cisco Spark clients use any certificate that exists in the client's trusted certificate store.

Once the Client and KMS have agreed on a symmetric key using the authenticated ECDH mechanism, the client then uses that channel to request a new encryption key specifically for the purpose of encrypting content destined for a single Spark room and the participants in that room, as shown in **(2)**. This key is known as a conversation key.

After the user writes a message, their client encrypts the message with the conversation key **(3)**, labels it with the room IDs of the destination room, and sends it to the Core **(4)**. The Core receives the message in encrypted form. The Core does not have the conversation key, so it cannot decrypt the message. The Core checks the list of users

[2] Clients are authenticated using their KMS authentication token, as described in the Room Authorization section, below.

associated with the room ID specified in the message's metadata, then sends the encrypted message to the other users in the room **(5)**. Additionally, the message, in its encrypted form, is stored in the Core's message database, associated with the appropriate room.

The message remains encrypted when the other users' clients receive it. The other user clients contact their KMS to get the conversation key in order to decrypt it **(6, 7)**. The exchange between the content recipients **(6)** and their associated KMS is identical to the first exchange **(1)**. The KMS authenticates each user to verify permission to access the conversation key by virtue of being in the associated room **(8)**. The KMS distributes the conversation key to the recipients, allowing them to decrypt and read the message **(9)**.

It is important to remember that two different layers of encryption are in use in the flows described above: hop-by-hop and end-to-end. As described above, user content and client-KMS interaction are encrypted end-to-end using symmetric encryption with room-specific conversation keys for user content and ephemeral keys for client-KMS content. Currently, the symmetric cipher used in Spark is AES256-GCM. As end-to-end encrypted content is communicated from client to server, server to server, and from server back to other clients, it is additionally protected using hop-by-hop encryption. Hop-by-hop encryption uses Transport Layer Security (TLS) protocol, which is the same protocol that a web browser uses when communicating with a bank or an online retailer. While the encryption methods used for both the end-to-end and hop-by-hop are the best available today, Spark is designed around a concept known as algorithm agility. Algorithm agility allows for the rapid adoption of new encryption mechanisms as today's strongest methods become outdated and new industry recommendations replace them.

User-generated content, as used above, includes every message, room title, and file shared on Spark.

Conversation Key URIs

Conversation Keys are identified and referenced by unique uniform resource identifiers (URIs). When end-to-end encrypted content is sent from a Client to the Spark Cloud, the header contains the key URI. The URI provides details of which KMS generated the key and provides a location that can be used to fetch the key—assuming the requestor can pass authentication and authorization checks.

Many Conversation Keys and Key Rotation

At least one Conversation Key per client that is contributing content to a Spark Room exists at any given time. For example, if Alice is contributing to a Spark Room from her mobile device and her laptop, and Bob is contributing to the same room from his laptop while viewing content from his mobile device, three (3) symmetric conversation keys would be active in the room: one for Alice's mobile device, one for her laptop, and one for Bob's laptop.

Conversation Keys are rotated to address when users depart rooms, either voluntarily (leave) or forcefully (kicked). When clients become aware of a user leaving (see Visible Authorization, below), the client is required to rotate its Conversation Key before contributing additional content to the room. The Spark Cloud enforces this behavior. Client-KMS symmetric keys also rotate periodically.

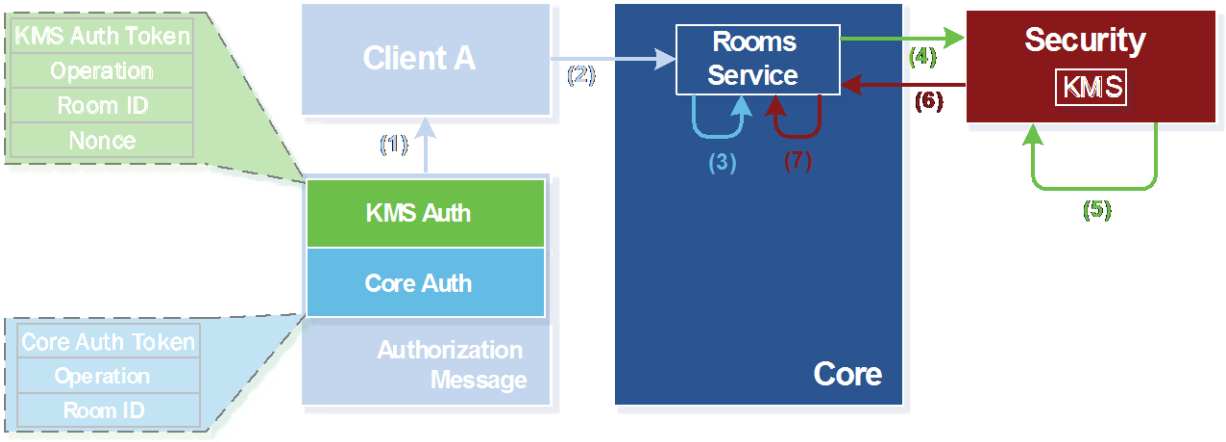
Room Authorization

When users add or remove other users from rooms, the user's client needs to make sure that both Spark and the user's KMS are aware of the changes to room authorization. The client produces an Authorization Message to notify Spark and the KMS about these changes. The Authorization Message contains of two distinct sub-messages:

- The first is an authorization change operation block that contains the client's authentication token[3] for the Core, the add/remove operation, and the unique room identifier to which that operation applies.
- The second is an encrypted authorization change operation block that contains the KMS authentication token, the add/remove operation, the unique room identifier to which the operation applies, and a *nonce* (a one-time use random block of data).

The sub-message is encrypted using the ephemeral symmetric key previously negotiated between the client and the KMS during Conversation Key retrieval. It is important to reiterate that the encrypted sub-message uses the KMS authentication token, whereas the sub-message destined for the Core uses the Core's authentication token. This use of distinct authentication tokens, along with the additional encryption layer on the KMS's change operation, prevents the Core from impersonating the client, which would otherwise allow the Core to falsify authorization requests to the KMS.

Figure 4. Production of two-part authorization message



- Once the client has produced the two-part Authorization Message **(1)** in Figure 2, the client sends the message to the Rooms Service in the Core over TLS **(2)**.
- The Rooms Service validates **(3)** the Core sub-message, including the Core authentication token and verifies the requesting user has authorization to modify the room specified in the Room ID.
- If everything looks good, the Rooms Service forwards the encrypted sub-message to the user's KMS **(4)**, along with the Room ID contained in the Core sub-message and the user's ID decoded from the Core authentication token.
- The inclusion of the plaintext message's Room ID and the authenticated user's ID allows the KMS to verify that the Rooms Service request matched the KMS request on authentication and authorization. The KMS decrypts the received message, validates the KMS authentication token, and ensures the requesting user is authorized to make changes to the specified Room **(5)**.
- If everything looks good, the KMS applies the requested operation and returns its success to the Rooms Service **(6)**. Upon receipt of the KMS's success in applying the operation, the Rooms Service applies the plaintext operation **(7)**.

A benefit of this approach is that it brings fate sharing. Client requests to add or remove a user from the room will cause the user to be added or removed on either both the Core and the KMS or on neither, a property typically

[3] All tokens are OAuth bearer tokens.

referred to as *atomicity* meaning that the two operations form an indivisible and irreducible set of operations. This ensures the Core and the KMS stay in sync while also providing the security benefits of Core controlled authorization to encrypted content and KMS controlled authorization to the keys needed to decrypt the content.

Visible Authorization

The list of authorized users for any given Spark Room is visible to all other authorized users of the room in Spark clients. In addition to the list of authorized users, the room displays join, leave, and kick activity in line with content in the room. These in-line messages make it clear to members of a room who was added to or removed from a room, and by whom. In the case of an employee leaving an organization, such as when an employee quits or is terminated, Spark will force a leave in all rooms to which the employee belonged. This both makes it clear to remaining room participants that the employee no longer has access to existing or future room content and forces all other participants to rotate their Conversation Keys.

Feature Access to Keys

In their current form, some features are only available if Cisco is granted access to keys. One such feature is Document Transcoding, which is the backend process that converts many file types, such as Microsoft Office documents, to image formats to ensure web and mobile devices can quickly view formats that are not natively supported on the recipient's platform. Features that require Key access request the keys from the enterprise KMS in the same manner all other user clients do, but they do so with a machine account that has been specifically authorized to access specific keys in the enterprise KMS by the enterprise administrator. Machine accounts are like user accounts, but where user accounts are associated with a username and password, machine accounts are associated with a machine ID and machine key and used for machine-to-machine (M2M) authentication.

We realize that some customers may not be comfortable with Cisco accessing key material for documents shared in Spark Rooms. For this reason, Spark will always give enterprise customers the ability to turn off any features that require key access for their entire enterprise. Turning off these features removes authorization for the feature's associated machine account in the enterprise's KMS and thus ensures no more Key access occurs. It is important to note, however, that an enterprise user uploading documents or content in to a room that contains participants from another enterprise might still result in cloud access to content, as the other enterprise might use a cloud KMS or features that require access to keys.

For a complete list of features that currently require Key access, along with instructions to disable them, please consult the Cisco Cloud Collaboration Management Portal. The Integrations and Extensibility section describes how third parties can add new features to the Cisco Spark platform.

Security Realm Deployment Options

You have seen how Cisco built end-to-end encryption into the fabric of Spark, relying on the separation of the Security Realm from the rest of the Cisco Spark Cloud to make it happen. For customers that want even stronger guarantees that Cisco, as the cloud service provider, has no access to their content, Cisco offers flexibility in the deployment of the services contained in the Security Realm—including the KMS.

Customers have the option to use Security Realm services hosted by Cisco or to deploy the services to the customer premises. Cisco will provide commercially available versions of each service, but will also provide source code for Security Realm services, such as the KMS, to any enterprise customer that requests it for purposes of verification of our claims. Additionally, all the services that live in the Security Realm use and provide industry standard protocols, such as JSON over RESTful HTTPS. Cisco is also actively working to standardize the key management techniques used by Cisco Spark to allow future third-party open-source and commercial solutions to

function as drop-in replacements for Cisco-provided services. Well-documented, standards-compliant, interfaces mean that customers that are highly sensitive, truly paranoid, or those requiring significant customization are free to eventually build their own implementations of these services as well.

Today, when the Security Realm is hosted outside of the Cisco Spark Cloud, Cisco will jointly operate the software in conjunction with the customer or partner. In this joint model, Cisco has access to logs, analytics, and provides upgrades. Logs never contain keys or personally identifiable information. Similarly, metrics are aggregated and help Cisco understand server performance. The customer or partner manages the hardware deployment and overall service provisioning and configuration.

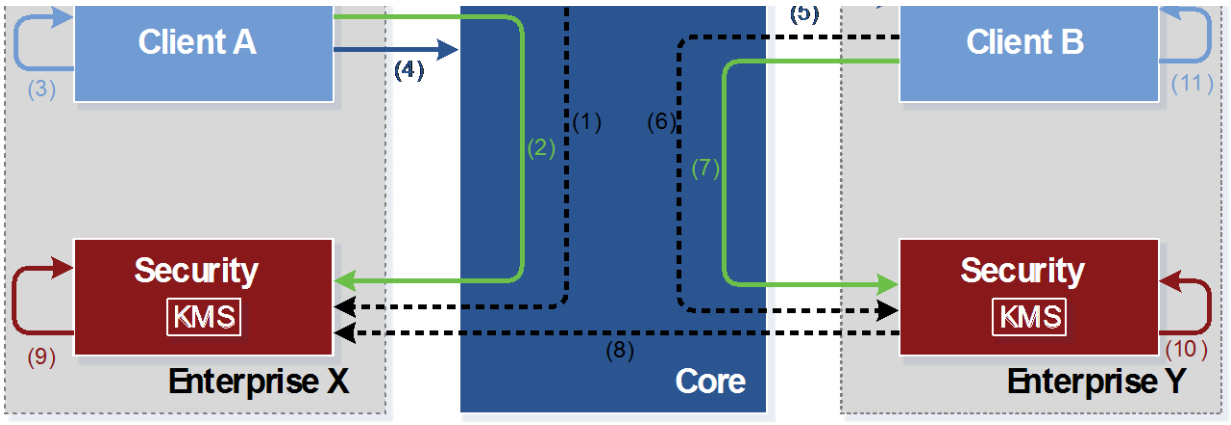
When administration flows and interfaces are finalized and stable, the Security Realm will be available in a fully partner- or enterprise-hosted and managed model. In this model, Cisco continues to operate the services in other Realms, but the Security Realm is fully operated by a Cisco partner on the partner's premises infrastructure or by the enterprise on the enterprise's infrastructure or preferred cloud provider. When this is available, the partner or enterprise will have complete operational control over the KMS, Search Indexer, associated databases, and all other services in the Security Realm.

For more information on Cisco's efforts to standardize the KMS architecture, refer to http://cs.co/keymanagement.

Key Management Server (KMS) Federation

Users in Cisco Spark are only associated with a single enterprise, and each enterprise potentially has its own KMS. When Cisco Spark users associated with different enterprises communicate, conversation keys still need to make it from one user to the other. In order to accomplish this, Cisco Spark uses a process known as Federation. Many of the steps in the Federation process are identical to the steps in the End-to-End Content Encryption section (above), with the notable difference that each Client communicates only with its associated KMS and the additional step of the two KMS communicating to share keys. Unlike traditional federation models, KMS federation does not require any configuration or setup by the customer or partner. The overall service remains a cloud model – where any user can talk to any other user in the world.

Figure 3. Spark communications across enterprises.



- As previously discussed, when a Spark user wants to send content to a Spark room, the user's client (Client A) must first establish a secure channel between itself and its associated KMS (Enterprise X), as shown in **(1)** in Figure 3, and then request a Conversation Key **(2)**. As before, this is done over a secure channel established **(1)** using an authenticated ephemeral ECDH exchange.

- Client A then encrypts **(3)** the content using symmetric encryption and the Conversation Key and sends it to the Spark Cloud **(4)**.
- The Spark Cloud checks the list of users associated with the room ID specified in the message's metadata (in this case, a user in Enterprise Y), and Spark sends the encrypted message to the recipient's clients **(5)**. The message is still encrypted when Client B receives it.
- Client B contacts the Enterprise Y KMS to obtain a Conversation Key for the received message **(6, 7)**. When the Enterprise Y KMS receives this request, it looks at the Conversation Key URI and determines that the Key exists in a remote KMS. As such, the Enterprise Y KMS establishes a mutual TLS channel, through the Core, to the originating KMS in Enterprise X **(8),** in order to request the Key.
- This mutual TLS channel is authenticated using the PKI certificate associated with each enterprise's KMS, wherein the PKI certificates must be issued by a CA that will not issue subordinate CA or intermediate certificates.

For a complete list of such certificate authorities, please consult the Cisco Cloud Collaboration Management Portal.

This connection, as all connections in Spark, routes through the Core to minimize firewall requirements and remote peers. However, it is encrypted end-to-end and invisible to the Core – just like all other communications in Spark. When the Enterprise X KMS receives this request, it checks the authorization list for users associated with the Spark Room and finds that, indeed, a user at Enterprise Y has authority to access the room.

Since Enterprise X's KMS has visibility only to the users associated with Enterprise X, it is unable to authenticate Enterprise Y users. Therefore, it must rely on the PKI certificate provided by Enterprise Y's KMS to establish the querying KMS's identity as belonging to Enterprise Y.

- Enterprise X's KMS verifies that at least one user at Enterprise Y is a valid participant in the room in order to authorize the key request **(9)**.
- Once the querying enterprise's KMS has been authenticated and the authorization check performed, Enterprise X's KMS returns the requested Conversation Key to Enterprise Y's KMS over the established mutual TLS channel.
- Enterprise Y's KMS caches the key in its local database and then performs an authorization check to ensure that the requesting client belongs to a user that is authorized to access this key **(10)**.
- Once authorized, the Enterprise Y KMS distributes the conversation key to Client B, allowing Client B to decrypt and read the message **(11)**.

In the case of an enterprise-associated user communicating with a non-enterprise user, the same exchange happens, with the only difference being that one of the two KMS involved is operated and managed by Cisco.

Verifiability

The protocols used by Cisco Spark, including those used for key management, are either existing standards or pending standards in the IETF or W3C as referenced throughout this document. These protocols, when implemented properly, ensure the end-to-end security of enterprise content. While the protocol level is easy for a customer to verify through packet inspection, the inner workings of trusted services within the Security Realm are not as easy to observe. In order to overcome this black box issue, Security Realm services can provide audit logs that can be compared against both client usage and explicitly granted cloud service access to verify the system is operating as expected. Additionally, Cisco will provide any enterprise customer, access to the source code for the components contained within the Security Realm in order to allow for inspection, compilation, and binary comparison to the same components offered in binary form for deployment.

Real-time Media Encryption

All media in Cisco Spark, such as voice, video, and desktop share, are transmitted using Secure Real-Time Transport Protocol (SRTP; is defined in RFC 3711). Currently, the *Cisco Spark Cloud decrypts real-time media for mixing, distribution, and public switched telephone network (PSTN) trunking and demarcation purposes.*

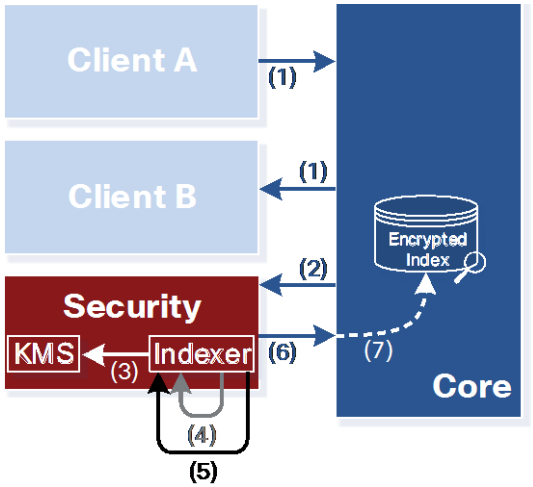
To further increase the security of SRTP in the future, Cisco is also one of the driving forces between the new Privacy Enhanced RTP Conferencing (PERC) working group at IETF. The goal of PERC is to enable end-to-end encrypted media that can still be authenticated hop-by-hop. As this new standard matures, Cisco Spark will make use of this enhancement to real-time media encryption such that real-time media encryption keys are backed by the KMS and the Cisco Spark Cloud will no longer decrypt PERC-compatible communications. PERC will not affect PSTN call decryption, which will require cloud decryption by the PSTN provider for the foreseeable future. For more information on PERC, refer to https://datatracker.ietf.org/wg/perc.

Encrypted Search: Secure and Snappy

Because the Spark Core never sees content in the clear, you would think enabling message search in the cloud would be impossible. But we've found a truly innovative way to enable global message search without having decryption capability in Cisco Spark Core.

We've done this by adding another component, the Indexer, to the Security Realm. Like the KMS, the Indexer is architecturally and operationally separate from the Core, but tightly bound with the KMS. It plays a crucial role in the two fundamental tasks needed to support global message search: building and querying the search index.

Figure 4. Messaging Indexing Sequence



Building the Search Index

The first step is building the search index.

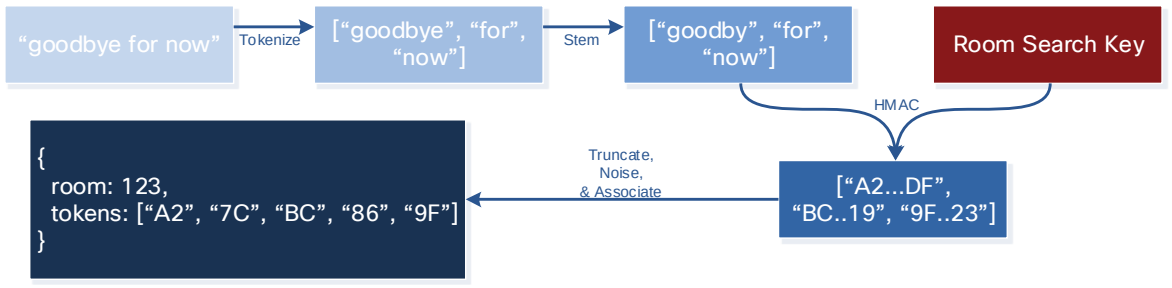
- Every time a user sends a message in Spark **(1)**, the message is sent to the Indexer in its end-to-end encrypted form, as shown **(2)** in Figure 4. The Indexer then queries the KMS for the conversation key necessary to decrypt the message. The Indexer is a Spark Bot[4] that is invited to every room by enterprise policy to help with search.
- The KMS provides the right conversation key to the Indexer because the Indexer is an active participant in the room and authorized to decrypt room resources **(3)**.
- The Indexer decrypts the message and separates it into the individual words that comprise it **(4)**.
- The Indexer then applies a cryptographic one-way hash to each word, or the roots of each word, using a special room search key stored in the KMS that is specific to the room[5] **(5)**. As an example, "goodbye for now" would be broken down into "goodbye," "for," and "now" with each word hashed. The result is a list of hashed words that all belong to the room where the message was posted. The hashes are basically one-way encryption—given a particular hash, there is no way to reverse it back to the word in the original message. The Indexer adds to this list some random "noise" hashes to ensure that room's messages

[4] See *Integrations and Extensibility*, below.
[5] Uses SHA-256 HMACs truncated to 80 bits.

cannot be reversed through frequency analysis. (Since words like “and” and “the” frequently appear in English usage, adding noise ensures that hashes in the room don’t follow that same frequency distribution).

- As a final step, the Indexer sends this list of hashes to the Spark Cloud **(6)**, where they are stored in the search index in encrypted form **(7)**. As a result, the Spark Cloud has an index of every word in every message associated with rooms in an encrypted form that cannot be decrypted by the Spark Cloud.

Figure 5. Messaging Index Data Flow

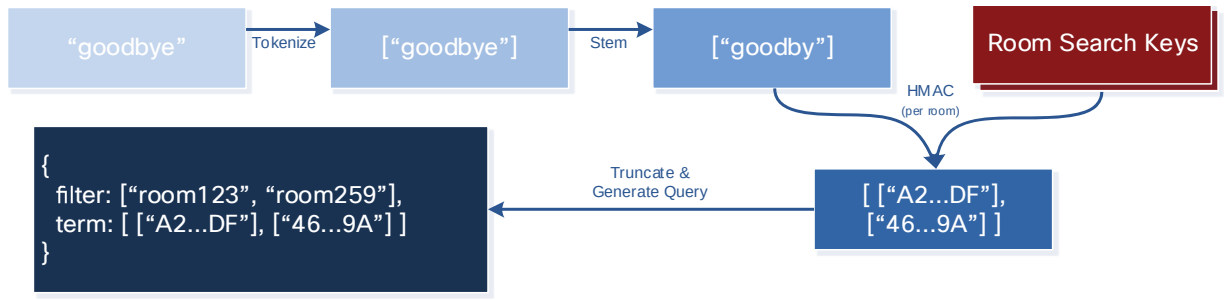


Querying the Search Index

When a user performs a search, the client interacts with the Indexer as though the user is having a one-on-one conversation with the Indexer. This means that all the end-to-end encryption previously described for user-to-user conversations is used for user-to-Indexer queries.

To expand on this, the search query is first encrypted on the user client using an end-to-end encryption key specifically for encrypting communications between the user’s client and the Indexer. Each user is associated with a specific Indexer. The user’s client sends the encrypted query to the Spark Cloud, which forwards the encrypted query to the Indexer together with the list of rooms to which the user has authorized access. The Spark Cloud cannot decrypt the search query since it’s encrypted with a key it cannot access; the end-to-end conversation key for this particular client’s conversation with the Indexer.

Figure 6. Search Data Flow



The Indexer then performs all the same steps as when it was building the search index: It breaks the query down into words and roots of words, then hashes each using the room search key for each room for which the user has access. So if a user who belongs to 10 rooms types in a two-word search query, the Indexer will generate a minimum of 20 hashed search terms and potentially many more depending on how many roots each word has. The Indexer then sends this list of hashed search terms back to the Spark Cloud.

The Spark Core can then search through the search index to see if any matches exist. As it finds rooms in the search index that are associated with any of the hashes received from the Indexer, it builds a list to send back to

the user's client, which combines the results with the associated conversation keys, either from client cache or by fetching them from the KMS, and then displays the search results to the user.

The Best of Both Worlds

We built the search capability in Spark without sacrificing either security or user experience. Although a single search can involve generating and comparing thousands of hashed search terms, the Spark search experience is just as snappy as users have come to expect from today's Internet search providers.

Integrations and Extensibility

Although we've provided a fantastic platform with the Cisco Spark Cloud, and a great team collaboration tool with Cisco Spark, we know partners and customers will want to customize and extend our offerings. Our goal is to make our APIs easy to learn and use. Developers want APIs to be comprehensive, yet simple, so they can focus on their own applications rather than a platform's complexity. We've taken extreme lengths to make our APIs elegant, abstracting away the complexities of the underlying platform.

While the core of Cisco Spark Cloud never has access to content, we realize that developers, partners, and customers will want to extend the platform in ways that will require access to content. As such, extensions to the Cisco Spark Cloud reside outside of the Core, and must be explicitly enabled by a customer or user. Because they exist outside of Cisco Spark Cloud, customers can choose where to deploy their platform extensions, by whom they're managed, and to what resources they have access. Integrations and extensions to Cisco Spark require access to encryption keys in the same manner described in the Feature Access to Keys section.

Cisco Spark currently defines three categories of integrations—Bots, Apps, and Webhooks. However we believe that customers and third parties will extend the platform in amazing new directions that we could not have foreseen. More information on the Cisco Spark Cloud APIs is available at https://developer.ciscospark.com.

Bots

Bots provide extended functionality for an entire enterprise, such as a call-recording service. Bots must either create or be invited to rooms to which they need access. Room invites can come from the enterprise policy or from a user who is already in the room.

Apps

Apps extend functionality for a single user, such as a personal assistant or document translation. In many ways, an App can be considered a cloud- or server-hosted Client with no user interface. Apps have access to all rooms to which the associated user has access, although the user can kick the App from specific rooms.

Webhooks

Webhooks provide single-URL access to simplified Core functionality, such as posting content to a room or notifications of new content. Webhooks can behave similarly to Bots and Apps in that they can have their own identity, like a Bot, or piggyback on a user's identity, like an App. Webhooks achieve single-URL simplicity by embedding scope-limited long-lived OAuth2 access tokens in the URL query string and performing all content encryption and decryption within the Webhook worker. In either case, Webhooks are scope-limited to a single resource (e.g. a room). These access scope limitations are configured and fixed when a Webhook is added.

Enterprise and User Choice

While we believe that the security of the Cisco Spark Cloud is the best in the industry, every Cisco customer has different security requirements. Customer choice is the key to making this hybrid model work. Customers can choose to use only the Core or to extend the Core with Bots, Apps, and Webhooks. Enterprises can define an

enterprise policy to allow for specific Bots and Apps, and users can make App choices within that policy. The Cisco Spark platform is designed around well-documented, standards-based APIs, which means that components outside of the Core (including Bots, Apps, and Webhooks) can be obtained from third parties or homegrown.

Certificate Pinning

Spark Client-to-Core communications are sent in a TLS-encrypted connection. Clients use a technique known as *certificate pinning* to ensure communications are not intercepted, read, or modified while in transit. Cisco pins server certificates to a few root CAs that have committed to not issue intermediate certificates through both the issuer's Certification Practice Statement and the root certificate containing a "pathLenConstraint" field in the BasicConstraints extension set to zero (0) to indicate that no CA certificates may follow the issuing Certificate in a certification path.

Data Privacy

Our commitment to providing a trusted service offering is not limited to protection of user content. In Spark, data about users and usage is protected using a combination of privacy tools and features that includes obfuscated identity, granular administrator roles, enterprise and user choice, and transparency. As with end-to-end encryption, these protections were built into the service from the ground up.

Obfuscated Identity

Having a rich concept of user identity is crucial for collaboration services. Users want to be able to connect quickly with people on their teams; search for other users by name, email, or phone number; and confirm identity visually with profile photos. At the same time, user identity information can be sensitive from both a user and an enterprise perspective. Limiting its exposure to just the contexts where it is needed is an essential design principle of Spark.

To limit exposure of user identity information, we distinguish between "real" and "obfuscated" identity in the Spark Cloud. The data that we collect as part of Spark user registration—user name, email address, phone number, and so on—is considered "real identity" and stored in the user's profile in a Spark Cloud component known as Common Identity. For each user, we also generate a random 128-bit universally unique identifier (UUID), which is the user's obfuscated identity. Similarly, for enterprises, we use a random 128-bit "organization ID" as the obfuscated identity of each enterprise. Within the Spark service, we use obfuscated identity everywhere we can, including:

- **Message routing:** All messages in Spark are routed from sender to recipient solely on basis of obfuscated identity. All cloud-internal queries related to individual users also rely on obfuscated identity, for example all of the KMS and Indexer interactions described above.
- **Server-side logging:** All logs generated by Spark Cloud application components for troubleshooting purposes use obfuscated identity.
- **Analytics:** Spark operates on a DevOps model and our development team makes decisions about how to improve the service by analyzing performance and usage data. The team bases those decisions on analysis of Spark usage records that rely on obfuscated identity.

Of course, when it comes time to render user or enterprise identity in a Spark client, the Cisco Cloud Collaboration Management Portal, or a third-party integration, we have a way for authorized clients and cloud service components to access real identity. Whenever any Spark client, cloud component, App, or Bot needs this access, it authenticates against Common Identity, which provides real identity information only to authorized requesters.

Granular Administrator Roles

Every Spark customer and partner has access to the Cisco Cloud Collaboration Management Portal, which provides a full-service management experience supporting trials, purchasing, account configuration, adoption, customer support, and developer API usage. Because we recognize that these features may involve access to sensitive information about users and accounts, product usage, and configuration information, we built the portal to support multiple different administrator roles with access to different subsets of information. For example, support administrators can access user information and support logs, while partner sales administrator access is more limited focused on aggregate usage reports and management of service trials. Full administrators have access to all of the portal's features and can assign appropriate roles to other administrators in their organizations.

We offer these roles to partners and customers, but we also use them ourselves to limit access to only to those Cisco administrators who need it. While our support administrators and engineers can access support logs and user information to help customers and partners troubleshoot, our sales and customer success personnel have limited access associated with the sales administrator role.

Enterprise and User Choice

Spark is designed to give both users and enterprises privacy choices without presenting complicated configuration interfaces. For enterprise administrators, choices include:

- **Single Sign-On (SSO):** Administrators can configure Spark to work with their existing SSO solutions. We support identity providers using Security Assertion Markup Language (SAML) 2.0 and OAuth 2.0.
- **Directory synchronization:** Administrators can have employee lifecycle changes reflected in Spark in real-time when using Microsoft Active Directory.
- **Data sharing with Cisco partner:** Enterprises can choose whether to share quality of service and engagement data with their Cisco partners to enable higher level partner support.

For users, choices include:

- **Device permissions:** Depending on which mobile or browser platform on which the user is running Spark, Spark will request a variety of device permissions, including phone, microphone, camera, audio recording, screen sharing, calendar, contacts, files and photos, and push notifications. On most platforms, these require explicit permission that the user can revoke at any time.
- **Proximity features:** On mobile devices, Spark clients can automatically pair with Cisco voice and video endpoints by listening for ultrasonic signals when the Spark client is active. Because this requires use of the device’s microphone, users can turn off this feature if they so choose.
- **Profile photos:** Profile photos are not required to use Spark.
- **External participant indicators:** Cisco Spark clients make it clear to users, through visual indicators, when a room contains participants who are not part of their enterprise organization.
- **Room moderator control:** Users can moderate rooms, allowing chosen participants to be made in to moderators with exclusive control of the room’s title and participant list.

Transparency

We want our users and customers to understand their choices and how we are managing and protecting the data they entrust with Cisco. We use a layered model of transparency to make this happen. We provide short disclosures that help users make real-time decisions within the Spark client itself. Further information is available in our support pages, which we update regularly. And for all the details of what information we collect, how we use it, and how we protect it, we rely on the Cisco Online Privacy Statement with a dedicated Spark privacy supplement.

Platform and Service Security

In addition to Cisco's Secure Development Lifecycle, Cisco Spark conducts frequent internal and external, whitebox and blackbox, penetration tests on the Spark platform and services. More information on the Cisco's Secure Development Lifecycle is available at: https://www.cisco.com/c/en/us/about/security-center/security-programs/secure-development-lifecycle.html

Incident Management & Corporate Security Policies

Cisco Product Security Incident Response

The Cisco Product Security Incident Response Team (PSIRT) is responsible for responding to Cisco product security incidents. The Cisco PSIRT is a dedicated, global team that manages the receipt, investigation, and public reporting of security vulnerability information related to Cisco products and networks. The on-call Cisco PSIRT works 24 hours with Cisco customers, independent security researchers, consultants, industry organizations, and other vendors to identify possible security issues with Cisco products and networks.

Reporting or Obtaining Support for a Suspected Security Vulnerability

Individuals or organizations experiencing a product security issue are strongly encouraged to contact Cisco PSIRT. Cisco welcomes reports from independent researchers, industry organizations, vendors, customers, and other sources concerned with product or network security. Please contact Cisco PSIRT using the following methods.

	Emergency Support
Phone	+1 877 228 7302 (toll-free within North America); +1 408 525 6532 (International direct-dial)
Hours	24 hours a day, 7 days a week
	Non-Emergency Support
Email	psirt@cisco.com
Hours	Support requests received via e-mail are typically acknowledged within 48 hours.

Find more information at http://www.cisco.com/web/about/security/psirt/security_vulnerability_policy.html

Transparency and Law Enforcement Requests for Customer Data

Cisco is committed to publishing data regarding requests or demands for customer data that we receive from law enforcement and national security agencies around the world. We will publish this data twice yearly (covering a reporting period of either January-to-June or July-to-December). Like other technology companies, we publish this data six months after the end of a given reporting period in compliance with restrictions on timing of such reports. Find more information at http://www.cisco.com/web/about/doing_business/trust-center/transparency-report.html

Americas Headquarters
Cisco Systems, Inc.
San Jose, CA

Asia Pacific Headquarters
Cisco Systems (USA) Pte. Ltd.
Singapore

Europe Headquarters
Cisco Systems International BV Amsterdam,
The Netherlands

Cisco has more than 200 offices worldwide. Addresses, phone numbers, and fax numbers are listed on the Cisco Website at **www.cisco.com/go/offices.**

COLLAB-SPARK-0616