TECHNICAL WHITE PAPER

Authentication with a Secure and Distributed API Design

TABLE OF CONTENTS

INTRODUCTION

As the usage of RESTful API endpoints continues to increase in the data center and for multi-cloud operations, the need for a secure and highly available method for accessing API endpoints increases. One of the less involved steps you can take to authenticate against an API is to use a token.

In this paper, we go over the architecture of token based requests, our secure API token architecture, how to create and use a token, and provide some thoughts on token rotation and distributed design. We are using Rubrik Cloud Data Management (CDM) 5.0 platform's set of RESTful APIs and token management to demonstrate the concepts and workflows.

OVERVIEW OF AUTHENTICATION TYPES

Most RESTful API endpoints will accept two different types of authorization: **Basic** authentication and **Token** (Bearer) authentication. This information is stored in the header of an API request to prove the identity of the user. More specifically, a key name of Authorization is paired with a value that contains the appropriate credentials.

BASIC AUTHENTICATION

The idea behind basic authentication is to send a key-value pair in the request header that contains the credentials necessary to use a RESTful method. This is similar in nature to sending a username and password to authenticate against any other session-based request; the server checks to ensure that you, the user, have the appropriate permissions and role required to access the desired resources.

Basic authentication is a simple method for authorizing a session and is often used for short-lived and ad-hoc requests. For example, a monitoring solution that is infrequently gathering data from an API resource will often use basic authentication to gather the needed data. Additionally, you may wish to retrieve some administrative cluster information - such as the cluster status or network topology - to answer an ad-hoc question from a colleague.

Generating the header required for basic authentication can be explained in three steps:

Take the string "{username}:{password}" and encode it using Base64, where {username} is the actual username and {password} is the actual password. The colon between username and password is important, even if there is no password. For this example, the username is "SpongeBob" and the password is "SquarePants".

```
"SpongeBob:SquarePants" is the plain text string
"U3BvbmdlQm9iOlNxdWFyZVBhbnRzCg"  is the base64 encoded string
```

Prefix this string with the word Basic, resulting in "Basic {base64_value}".

```
"Basic U3BvbmdlQm9iOlNxdWFyZVBhbnRzCg"
```

A header key of Authorization is created, with the above results stored in the value.

```
"Authorization: Basic U3BvbmdlQm9iOlNxdWFyZVBhbnRzCg"
```

For more information on basic authentication for scripting, you can read my blog post entitled "Tackling Basic RESTful Authentication with PowerShell" to better understand how the base64 string is encoded in the value. There is a section dedicated to crafting a basic authentication key-value pair.

TOKEN AUTHENTICATION

Token authentication is a much more popular method for handling authorization, especially across public cloud services. Instead of using the credentials of a user, a token is instead used to represent the session of that user. The user's permissions, role, and scope remain tied to the user account itself with the token being used for authentication purposes. These are helpful for services that need to programmatically call upon the API and can be invalidated if leaked, breached, or no longer needed without adversely affecting the user account.

Rubrik uses the concept of an **API Token** to map to the token authentication model for tasks such as automated workflows, script authentication, and Multi-factor Authentication (MFA).

The header value for an API Token looks similar to the basic authentication, except that the word "Basic" is replaced with "Bearer" and the value used is the API Token, not a base64 encoded username and password. For example:

```
Authorization: Bearer joiN2NiNGIyN
```

This will be covered in more depth in the Common API Token Workflows section of this document.

SECURE API TOKEN ARCHITECTURE

A Rubrik cluster is comprised of a collection of nodes that form a distributed cluster under a single namespace. With the 5.0 CDM release, sessions and API Tokens are globally available from any node within the cluster and can survive node failures, restarts, and upgrades.

To accomplish this, the cluster maintains session details in a distributed session table. This table holds the *userId* as the partition key and the token as the clustering key. Tokens are generated with JSON Web Tokens (JWT), an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. The tokens are digitally signed using HMAC-SHA256 algorithm.

- **SHA-256** is a cryptographic hash function that generates a 256-bit hash for text.
- **Hash-based Message Authentication Code** (HMAC) is a piece of information used to authenticate a message, confirming the integrity and authenticity of the message.

When a token reaches its expiration time, it is deleted. Deletion was chosen over archiving as it is more logical for the purposes of tokens which are meant to be transient. Sessions may be created and deleted in high frequency for this same reason.

Note: You can visit `/node/{id}/sessions` to retrieve a list of all active sessions, which includes sessions based on API Tokens.

Rubrik's API Tokens have additional layers of security applied to them, such as following a "View On Create" philosophy that ensures the token value is only visible during creation. Once created, the only possible future action is to delete the token.

The following requests are not available to API Tokens for enhanced security:

- Updating or deleting any MFA (Multi-factor Authentication) servers.
- Creating new sessions or generating additional API tokens.
- Creating new user accounts or updating user account information.
- Updating user preferences.
- Creating, updating, or deleting LDAP services.

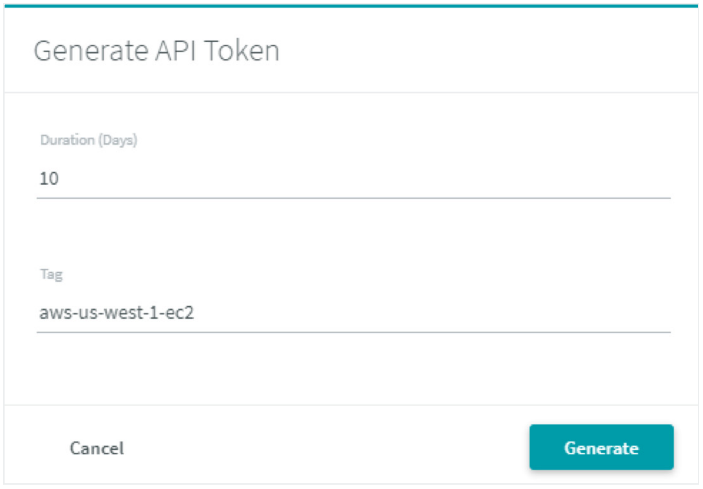
COMMON API TOKEN WORKFLOWS

The following workflows are commonly used to create, consume, remove, and rotate an API Token when interacting with a Rubrik cluster.

GENERATING A NEW API TOKEN

The process for manually creating a new API Token is to log into the UI under whatever account you wish to represent with a token.

- Click on the account name in the upper right corner of the UI and select **API Token Manager**.
- From there, a list of all API Tokens that have been generated will appear, but the token value itself is not shown and is only available at the time of generation for security purposes.
- Click the green "+" button in the upper right corner to start a new token wizard, then fill in the duration (in days) and a tag value (to describe the token) to complete the workflow.


Generate API Token

Duration (Days): 10

Tag: aws-us-west-1-ec2

The UI simplifies choices down to duration and tag.

You can also request a token using an automated workflow and the PowerShell SDK for Rubrik. The command is `New-RubrikAPIToken` and will generate an API Token that adheres to your expiration and tagging needs. Keep in mind, however, that there is a limit to how many active tokens you can have concurrently valid, and that scripting token generation should be a part of your token lifecycle management security process to rotate in fresh tokens.

In the example below, I have requested a new API Token to use with a serverless function running on AWS Lambda in US West 1 (N. California). The PowerShell function is sending a POST request to `/session` using the current session information. I have truncated the resulting values for brevity and highlighted the API Token in red:

```
New-RubrikAPIToken -Expiration 600 -Tag 'aws-us-west-1-lambda'

Id                       : 7cb4b25c
organizationId           : Organization:::b6c0d1d1
userId                   : 30047f2a
Token                    : joiN2NiNGIyN
expiration               : 2019-06-26 20:05:28
Tag                      : aws-us-west-1-lambda
```

Under the covers, an API request is being sent using my session information to generate a new API Token. The payload of the body is as follows:

```
Body = {
 "initParams": {
 "apiToken": {
 "tag": "aws-us-west-1-lambda",
 "expiration": 600
 }
 }
}
```

Note: At the time of this paper, the length of a tag cannot exceed 20 characters.

If you'd also like to get a list of different API Tokens that exist, send a GET request to `/session` with a query parameter containing your user ID value. Alternatively, the PowerShell function `Get-RubrikAPIToken` will retrieve all known tokens based on your current session's user ID value.

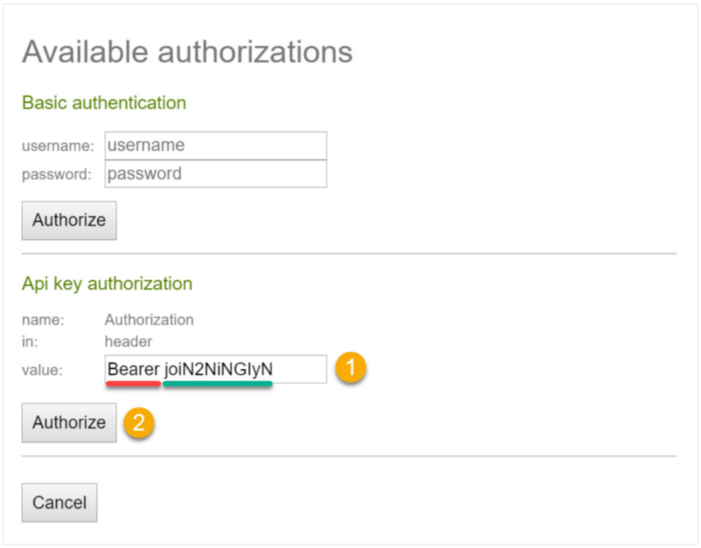
MAKING A REQUEST WITH AN API TOKEN

The use of a token for authorization changes the header construction slightly. The key remains authorization but the value changes to using the word Bearer followed by the token itself. Using "Bearer" seems to be the single most troublesome spot for those new to authenticating with a token. For example:

```
Authorization: Bearer joiN2NiNGIyN
```

If using the Rubrik REST API Explorer, click on the green Authorize button on the top right of the page.

1. Scroll down to the API Key Authorization section and enter the word `Bearer` (red line), insert a space, and then paste the API Token (green line). Note that the token below is truncated for brevity.
2. Click Authorize to use the API Token for subsequent requests during your session.


Available authorizations

Basic authentication

username: username
password: password

Api key authorization

name: Authorization
in: header
value: Bearer joiN2NiNGIyN

The "Basic" value changes to "Bearer" when a token is used.

The API Token will remain in place until it reaches the expiration date and time. The token will delete itself and become invalid beyond that point.

It's also possible to use the token with the PowerShell SDK for Rubrik. Instead of passing along a credential object or manually entering your username and password, use the `token` parameter as shown below:

```
Connect-Rubrik -Server 192.168.1.124 -Token joiN2NiNGIyN

Name                     Value
----                     -----
Time                     2019-06-26 14:40:06
Api                      1
Server                   192.168.1.124
Header                   {Authorization, User-Agent}
Id                       null
userId                   null
Version                  5.0.0
```

You are now able to execute API requests by way of the SDK using the API Token. For example, if I wanted to get information on an SLA Domain:

```
Get-RubrikSLA -Name 'Gold AWS DND' -PrimaryClusterID 'local'

id                       : e641d876-0955-4e7e-a7f1-a162281fdc74
primaryClusterId         : 8b4fe6f6-cc87-4354-a125-b65e23cf8c90
name                     : Gold AWS DND
--snip--
```

REMOVING AN API TOKEN

In the event that you need to remove an API Token prior to its expiration, such as seeing that a colleague accidentally put it in their GitHub repository, it's important to understand the process.

From a UI perspective, visit the API Token Manager page and select the token that should be removed, then choose "Delete" to remove it. Or, send a POST request to the `/session/bulk_delete` endpoint with an array of token ID values.

Alternatively, use the PowerShell SDK for Rubrik to remove the token. The cmdlet `Remove-RubrikAPIToken` accepts the token id value as the only parameter, as shown below:

```
Remove-RubrikAPIToken -TokenId '7cb4b25c'

Confirm
Are you sure you want to perform this action?
Performing the operation "Deletes session tokens" on target "7cb4b25c".
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "Y"): y
```

Because this is a destructive operation, the cmdlet requires confirmation and be silenced with `-Force` if you do not want to entertain manual acceptance. You can also pass an array argument that contains multiple token id values if you want to remove more than one at a time, which I have found to be handy in testing environments. Here is an example:

```
Remove-RubrikAPIToken -TokenId
("11111111-2222-3333-4444-555555555555","aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee")
```

ROTATING YOUR API TOKEN

While it's possible to generate a token that lasts for 365 days, this isn't the ideal use case for a production environment. I prefer to use shorter lived tokens that live in a secure vault and are rotated on a regular basis, such as weekly or monthly. Or issue them to others who need access for a limited period of time, such as power users, contractors, or a project team.

In these situations, my standard operation procedure is to:

- Use short lived tokens and store them in AWS KMS, HashiCorp Vault, secured variables in AppVeyor and Azure Automation, or even something more consumer like KeePass for certain situations.
- Make sure your code is referencing a token from these locations. Don't use a hard coded value that is not easily rotated.
- Generate a new token prior to the expiration of the old token using the PowerShell SDK for Rubrik, a direct call to the RESTful API (such as Invoke-RestMethod or curl), or any other trigger-based workflow that you prefer that can hit an API endpoint as the user or service account that is represented by the token.

Getting in the habit of using tokens for automation will increase your security awareness, reduce your reliance on user credentials, and aid your transition into a more service-oriented architecture approach.

HIGHLY AVAILABLE API DESIGN

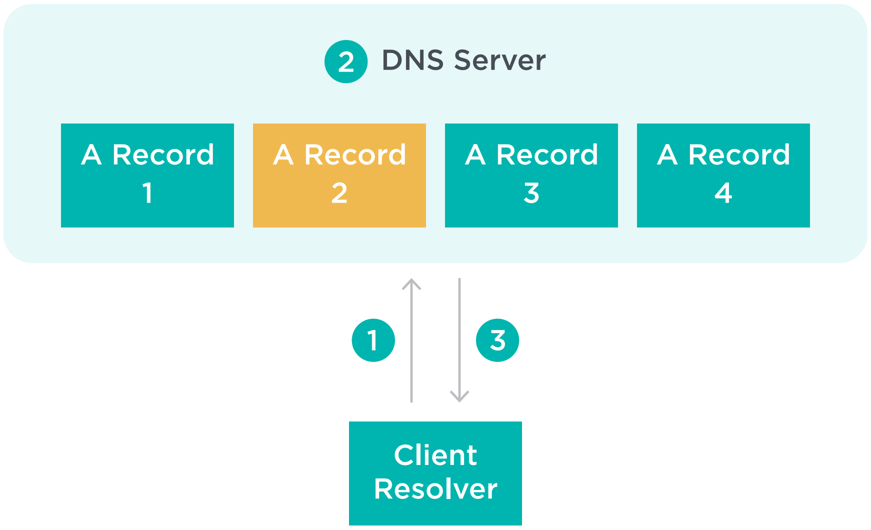
Since session state is persistent across a Rubrik cluster, any node can be used for API requests. This works regardless of the authentication method chosen: basic and token authentication are both valid across the entire cluster namespace. Because of this, there is no need to push state handling to a virtual IP or virtual namespace at the network layer.

However, customers are often keen to use a namespace that exceeds a single node when using various automation and scripting models. In this section, we will explore different design considerations for highly available API requests, such as round robin DNS and third party load balancing.

ROUND ROBIN DNS

A simple method for offering load distribution is to leverage round-robin DNS. This requires configuring multiple DNS A records to represent the various IP addresses that are presented by two or more Rubrik cluster nodes. Each A record is generated using identical host names mapping to a unique IP address with a short Time To Live (TTL) of 1 minute, which is adjustable for your specific preferences. It is advised to keep the TTL short, such as 1 - 5 minutes, to ensure that various requests are being sent to different nodes frequently.

1. A client seeks to resolve the Rubrik cluster's DNS address and sends a request.
2. The DNS server responds with one of the IP addresses in the list of A records. In this case, the second A record is selected.
3. The client resolves to the second A record and stores it in memory for the duration of the TTL. This provides load distribution among multiple nodes for responding to API requests.



For example, a hostname of "cluster-b-rr.rubrik.us" is generated on a DNS server that is authoritative for the *rubrik.us* testing domain. Using a DNS query tool such as nslookup will reveal all of the A records that exist for the hostname, as shown below:

```
nslookup cluster-b-rr.rubrik.us

Name:         cluster-b-rr.rubrik.us
Addresses:    192.168.150.121
              192.168.150.122
              192.168.150.123
              192.168.150.124
```

There are a few design considerations to be aware of with this approach. Namely, the fact that the health and utilization of the nodes are not considered. However, the trade-off is simplicity in the implementation and ability to scale rapidly by adding (or removing) A records from DNS without the need for any third party software or hardware. Using metadata related to node health is covered in a later section entitled Node Health Checks.

TEST CONFIGURATION

The following configuration was used for this test:

- Rubrik CDM 5.0 on a 4 node cluster
    - Node A - 192.168.150.121
    - Node B - 192.168.150.122
    - Node C - 192.168.150.123
    - Node D - 192.168.150.124
- Rubrik API Token
- Live Mount of a VMware Virtual Machine

The following workflow was used to test the usage of API Tokens across the cluster. Each request was made to the DNS name with a 1 minute TTL to ensure that requests rotated among different nodes within the Rubrik cluster.

1. Send a request to Node A to create a Live Mount for a VMware VM and receive the async request id that represents the long-lived task of generating the clone.
2. Send a request to Node B to check the status of the Live Mount.
3. Send a request to Node C to get details on the Live Mount.
4. Send a request to Node D to remove the Live Mount.

TEST RESULTS

For each test run, the results were observed, captured, and validated using a series of PowerShell scripts across a single session established with an API Token that was stored securely in memory on the test client. No other sessions were established or initiated. Each script was separated by a client-side DNS flush to ensure that a new IP addresses was retrieved by the client from the DNS server.

Test 1: Send a request to Node A to create a Live Mount for a VMware VM and receive the async request id that represents the long-lived task of generating the clone.

```
ipconfig.exe /flushdns
ping cluster-b-rr.rubrik.us -n 1
Get-RubrikVM -id $vmid | Get-RubrikSnapshot | Select-Object
-First 1 | New-RubrikMount
```

```
Successfully flushed the DNS Resolver Cache.

Pinging cluster-b-rr.rubrik.us [192.168.150.121] with 32 bytes of data:
Reply from 192.168.150.121: bytes=32 time=1ms TTL=61

id            :
MOUNT_SNAPSHOT_301079a4-3430-4c78-864c-c863147be99c_723b7ee4-a70b-4251-
a0b6-ff932422b361:::0
status        : QUEUED
progress      : 0
startTime     : 2019-07-18 00:40:57
links         :
{@{href=https://cluster-b-rr.rubrik.us/api/v1/vmware/vm/request/MOUNT_SNAPSHOT_301079a4-
3430-4c78-864c-c863147be99c_723b7ee4-a70b-4
            251-a0b6-ff932422b361:::0; rel=self}}
```

Test 2: Send a request to Node B to check the status of the Live Mount.

```
ipconfig.exe /flushdns
ping cluster-b-rr.rubrik.us -n 1
Get-RubrikRequest -id $snapid.id -Type:vmware/vm
```

```
Successfully flushed the DNS Resolver Cache.

Pinging cluster-b-rr.rubrik.us [192.168.150.122] with 32 bytes of data:
Reply from 192.168.150.122: bytes=32 time=1ms TTL=61

id          :
MOUNT_SNAPSHOT_301079a4-3430-4c78-864c-c863147be99c_723b7ee4-a70b-4251-
a0b6-ff932422b361:::0
status      : SUCCEEDED
startTime   : 2019-07-18 00:40:57
endTime     : 2019-07-18 00:41:08
nodeId      : cluster:::RVM182S004243
links       :
{@{href=https://cluster-b-rr.rubrik.us/api/v1/vmware/vm/snapshot/mount/86669656-b6cf-
446f-95cb-ff9af1b50394; rel=result}, @{href=https://cluster-b-rr.rubrik.us/api/v1/
vmware/vm/request/MOUNT_SNAPSHOT_301079a4-3430-4c78-864c-c863147be99c_723b7ee4-a70b-
4251-a0b6-ff932422b361:::0; rel=self}}
```

Test 3: Send a request to Node C to get details on the Live Mount.

```
ipconfig.exe /flushdns
ping cluster-b-rr.rubrik.us -n 1
Get-RubrikMount -id 86669656-b6cf-446f-95cb-ff9af1b50394
```

```
Successfully flushed the DNS Resolver Cache.

Pinging cluster-b-rr.rubrik.us [192.168.150.123] with 32 bytes of data:
Reply from 192.168.150.123: bytes=32 time=1ms TTL=61

snapshotDate                : 2019-07-17 23:19:58
createDatastoreOnlyMount    : False
vmId                        : VirtualMachine:::1226ff04-6100-454f-905b-5df817b6981a-vm-78
isReady                     : True
datastoreReady              : True
hostId                      : VmwareHost:::1226ff04-6100-454f-905b-5df817b6981a-host-71
datastoreName               : rubrik_86669656b6cf446f95cbff9af1b50394
mountTimestamp              : 2019-07-18 00:40:57
powerStatus                 : poweredOn
id                          : 86669656-b6cf-446f-95cb-ff9af1b50394
mountedVmId                 :
VirtualMachine:::1226ff04-6100-454f-905b-5df817b6981a-vm-43478
hasAttachingDisk            : False
```

Test 4: Send a request to Node D to remove the Live Mount.

```
ipconfig.exe /flushdns
ping cluster-b-rr.rubrik.us -n 1
Remove-RubrikMount -id 86669656-b6cf-446f-95cb-ff9af1b50394
```

```
Successfully flushed the DNS Resolver Cache.

Pinging cluster-b-rr.rubrik.us [192.168.150.124] with 32 bytes of data:
Reply from 192.168.150.124: bytes=32 time=1ms TTL=61

id           :
UNMOUNT_SNAPSHOT_e9156564-9714-45e9-87df-b07038ebc35a_a47fb010-f5d3-4021-
b68f-456aa02e296e:::0
status       : QUEUED
progress     : 0
startTime    : 2019-07-18 01:44:05
links        :
{@{href=https://cluster-b-rr.rubrik.us/api/v1/vmware/vm/request/UNMOUNT_SNAPSHOT_
e9156564-9714-45e9-87df-b07038ebc35a_a47fb010-f5d3
 -4021-b68f-456aa02e296e:::0; rel=self}}
```

NODE HEALTH CHECKS

If using a third party solution - such as load balancers from Citrix NetScaler, F5 BIG-IP, or VMware NSX - more opportunities to gauge a node's health are made available. These solutions offer the ability to report on the health of a node based on user-defined checks with data validation. The desired nodes can be added to a virtual pool and checked for health to determine if they should be active or inactive within the pool's available node list.

For simple node availability, it is advised to use a request to one of the following open (unsecured) endpoints using the GET method:

- `/cluster/me/api_version`
- `/cluster/me/version`

The resource being queried does not require authentication, making them ideal for simple up / down health checks. A healthy node should return a status code of 200 along with a string formatted in JSON that provides the API version and Cloud Data Management version, respectively. If a node does not respond with the correct status code and response string, the node can be removed from the virtual pool on the load balancer.

If your load balancer supports authentication, you can also query the following endpoint using the GET method to determine if a node is healthy:

- `/cluster/me/system_status`

The state of the node will be returned as a string, with "OK" being the optimal value received. The value deviates from being "OK" when there are failures detected by the node, such as a degradation hard drive or flash device, and will result in showing "BAD". A periodic health poll of the node can be leveraged to determine when to remove or re-add a node to the virtual pool. It is advised to treat any condition other than "OK" as a health check failure since the node may also be undergoing maintenance, bootstrap, or an upgrade.

CONCLUSION

The usage of an API Token that is available across a logical namespace of clustered nodes provides a tangible benefit for customers looking to leverage automation against the Rubrik APIs and SDKs. For those that also wish to simplify their namespace with the usage of round robin DNS or a third party load balancing solution, there exists opportunities to provide load distribution to nodes for UI and API requests.

ABOUT THE AUTHORS

Chris Wahl is the Chief Technologist at Rubrik. He is the author of the award winning Wahl Network blog and host of the Datanauts Podcast. Chris focuses on creating content that revolves around hybrid clouds, CI pipelines, workflow automation, building operational excellence, and evangelizing solutions that benefit the technology community. In addition to co-authoring "Networking for VMware Administrators" for VMware Press, he has published hundreds of articles and continues to enable others on a plethora of open source initiatives, including Rubrik Build. You can reach him on Twitter via @ChrisWahl.

Ben Reisner is a Member of Technical Staff at Rubrik on the security team focusing on certificate management, session management, and secure communication. He also holds a patent for the design of Polaris Radar.

Ronald Gil is a Member of Technical Staff at Rubrik. As part of the security team, he works on a variety of product security features such as session management, authentication, and encryption at rest.