

Introduction

3 minutes

A shared access signature (SAS) is a URI that grants restricted access rights to Azure Storage resources. You can provide a shared access signature to clients that you want to grant delegate access to certain storage account resources.

After completing this module, you'll be able to:

- Identify the three types of shared access signatures
- Explain when to implement shared access signatures
- Create a stored access policy

Next unit: Discover shared access signatures

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

100 XP

Discover shared access signatures

3 minutes

A shared access signature (SAS) is a signed URI that points to one or more storage resources and includes a token that contains a special set of query parameters. The token indicates how the resources may be accessed by the client. One of the query parameters, the signature, is constructed from the SAS parameters and signed with the key that was used to create the SAS. This signature is used by Azure Storage to authorize access to the storage resource.

Types of shared access signatures

Azure Storage supports three types of shared access signatures:

- **User delegation SAS:** A user delegation SAS is secured with Azure Active Directory credentials and also by the permissions specified for the SAS. A user delegation SAS applies to Blob storage only.
- **Service SAS:** A service SAS is secured with the storage account key. A service SAS delegates access to a resource in the following Azure Storage services: Blob storage, Queue storage, Table storage, or Azure Files.
- **Account SAS:** An account SAS is secured with the storage account key. An account SAS delegates access to resources in one or more of the storage services. All of the operations available via a service or user delegation SAS are also available via an account SAS.

📌 Note

Microsoft recommends that you use Azure Active Directory credentials when possible as a security best practice, rather than using the account key, which can be more easily compromised. When your application design requires shared access signatures for access to Blob storage, use Azure Active Directory credentials to create a user delegation SAS when possible for superior security

How shared access signatures work

When you use a SAS to access data stored in Azure Storage, you need two components. The first is a URI to the resource you want to access. The second part is a SAS token that you've created to authorize access to that resource.

In a single URI, such as `https://medicalrecords.blob.core.windows.net/patient-images/patient-116139-nq8z7f.jpg?sp=r&st=2020-01-20T11:42:32Z&se=2020-01-20T19:42:32Z&spr=https&sv=2019-02-02&sr=b&sig=SrW1HZ5Nb6MbRzTbXCaPm%2BJiSEn15tC91Y4umMPwVZs%3D`, you can separate the URI from the SAS token as follows:

- **URI:** `https://medicalrecords.blob.core.windows.net/patient-images/patient-116139-nq8z7f.jpg?`
- **SAS token:** `sp=r&st=2020-01-20T11:42:32Z&se=2020-01-20T19:42:32Z&spr=https&sv=2019-02-02&sr=b&sig=SrW1HZ5Nb6MbRzTbXCaPm%2BJiSEn15tC91Y4umMPwVZs%3D`

The SAS token itself is made up of several components.

Component	Description
<code>sp=r</code>	Controls the access rights. The values can be <code>a</code> for add, <code>c</code> for create, <code>d</code> for delete, <code>l</code> for list, <code>r</code> for read, or <code>w</code> for write. This example is read only. The example <code>sp=acd1rw</code> grants all the available rights.
<code>st=2020-01-20T11:42:32Z</code>	The date and time when access starts.
<code>se=2020-01-20T19:42:32Z</code>	The date and time when access ends. This example grants eight hours of access.
<code>sv=2019-02-02</code>	The version of the storage API to use.
<code>sr=b</code>	The kind of storage being accessed. In this example, <code>b</code> is for blob.
<code>sig=SrW1HZ5Nb6MbRzTbXCaPm%2BJiSEn15tC91Y4umMPwVZs%3D</code>	The cryptographic signature.

Best practices

To reduce the potential risks of using a SAS, Microsoft provides some guidance:

- To securely distribute a SAS and prevent man-in-the-middle attacks, always use HTTPS.
- The most secure SAS is a user delegation SAS. Use it wherever possible because it removes the need to store your storage account key in code. You must use Azure Active Directory to manage credentials. This option might not be possible for your solution.
- Try to set your expiration time to the smallest useful value. If a SAS key becomes compromised, it can be exploited for only a short time.
- Apply the rule of minimum-required privileges. Only grant the access that's required. For example, in your app, read-only access is sufficient.
- There are some situations where a SAS isn't the correct solution. When there's an unacceptable risk of using a SAS, create a middle-tier service to manage users and their access to storage.

The most flexible and secure way to use a service or account SAS is to associate the SAS tokens with a stored access policy.

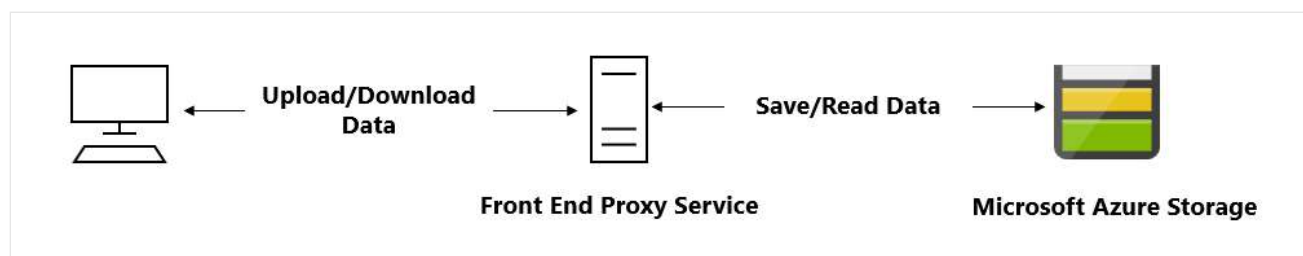
Choose when to use shared access signatures

3 minutes

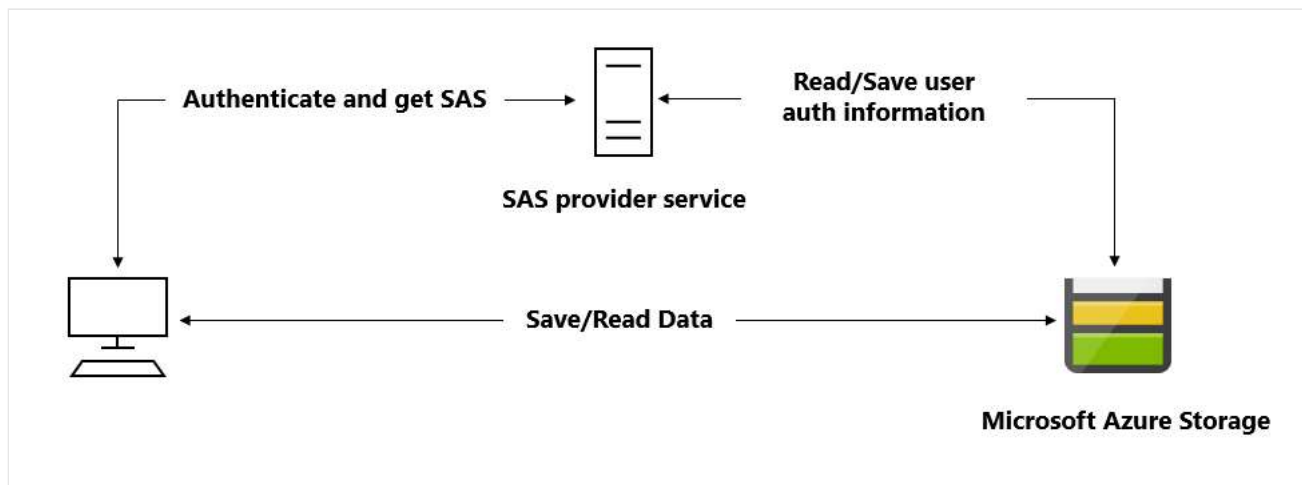
Use a SAS when you want to provide secure access to resources in your storage account to any client who does not otherwise have permissions to those resources.

A common scenario where a SAS is useful is a service where users read and write their own data to your storage account. In a scenario where a storage account stores user data, there are two typical design patterns:

- Clients upload and download data via a front-end proxy service, which performs authentication. This front-end proxy service has the advantage of allowing validation of business rules, but for large amounts of data or high-volume transactions, creating a service that can scale to match demand may be expensive or difficult.



- A lightweight service authenticates the client as needed and then generates a SAS. Once the client application receives the SAS, they can access storage account resources directly with the permissions defined by the SAS and for the interval allowed by the SAS. The SAS mitigates the need for routing all data through the front-end proxy service.



Many real-world services may use a hybrid of these two approaches. For example, some data might be processed and validated via the front-end proxy, while other data is saved and/or read directly using SAS.

Additionally, a SAS is required to authorize access to the source object in a copy operation in certain scenarios:

- When you copy a blob to another blob that resides in a different storage account, you must use a SAS to authorize access to the source blob. You can optionally use a SAS to authorize access to the destination blob as well.
- When you copy a file to another file that resides in a different storage account, you must use a SAS to authorize access to the source file. You can optionally use a SAS to authorize access to the destination file as well.
- When you copy a blob to a file, or a file to a blob, you must use a SAS to authorize access to the source object, even if the source and destination objects reside within the same storage account.

Next unit: Explore stored access policies

[Continue >](#)

How are we doing? ☆ ☆ ☆ ☆ ☆

Explore stored access policies

3 minutes

A stored access policy provides an additional level of control over service-level shared access signatures (SAS) on the server side. Establishing a stored access policy groups shared access signatures and provides additional restrictions for signatures that are bound by the policy. You can use a stored access policy to change the start time, expiry time, or permissions for a signature, or to revoke it after it has been issued.

The following storage resources support stored access policies:

- Blob containers
- File shares
- Queues
- Tables

Creating a stored access policy

The access policy for a SAS consists of the start time, expiry time, and permissions for the signature. You can specify all of these parameters on the signature URI and none within the stored access policy; all on the stored access policy and none on the URI; or some combination of the two. However, you cannot specify a given parameter on both the SAS token and the stored access policy.

To create or modify a stored access policy, call the `Set ACL` operation for the resource (see [Set Container ACL](#), [Set Queue ACL](#), [Set Table ACL](#), or [Set Share ACL](#)) with a request body that specifies the terms of the access policy. The body of the request includes a unique signed identifier of your choosing, up to 64 characters in length, and the optional parameters of the access policy, as follows:


ⓘ Note

When you establish a stored access policy on a container, table, queue, or share, it may take up to 30 seconds to take effect. During this time requests against a SAS associated with the

stored access policy may fail with status code 403 (Forbidden), until the access policy becomes active. Table entity range restrictions (`startpk`, `startrk`, `endpk`, and `endrk`) cannot be specified in a stored access policy.

Below are examples of creating a stored access policy by using C# .NET and the Azure CLI.

C#

 Copy

```
BlobSignedIdentifier identifier = new BlobSignedIdentifier
{
    Id = "stored access policy identifier",
    AccessPolicy = new BlobAccessPolicy
    {
        ExpiresOn = DateTimeOffset.UtcNow.AddHours(1),
        Permissions = "rw"
    }
};

blobContainer.SetAccessPolicy(permissions: new BlobSignedIdentifier[] { identifier
});
```

Azure CLI

 Copy

```
az storage container policy create \
  --name <stored access policy identifier> \
  --container-name <container name> \
  --start <start time UTC datetime> \
  --expiry <expiry time UTC datetime> \
  --permissions <(a)dd, (c)reate, (d)eleate, (l)ist, (r)ead, or (w)rite> \
  --account-key <storage account key> \
  --account-name <storage account name> \
```

Modifying or revoking a stored access policy

To modify the parameters of the stored access policy you can call the access control list operation for the resource type to replace the existing policy. For example, if your existing policy grants read and write permissions to a resource, you can modify it to grant only read permissions for all future requests.

To revoke a stored access policy you can delete it, rename it by changing the signed identifier, or change the expiry time to a value in the past. Changing the signed identifier breaks the associations between any existing signatures and the stored access policy. Changing the expiry

time to a value in the past causes any associated signatures to expire. Deleting or modifying the stored access policy immediately affects all of the SAS associated with it.

To remove a single access policy, call the resource's `Set ACL` operation, passing in the set of signed identifiers that you wish to maintain on the container. To remove all access policies from the resource, call the `Set ACL` operation with an empty request body.

Next unit: Knowledge check

[Continue >](#)

How are we doing? ☆ ☆ ☆ ☆ ☆

Knowledge check

3 minutes

Check your knowledge

1. Which of the following types of shared access signatures (SAS) applies to Blob storage only?

☒ Account SAS

✗ That's incorrect. An account SAS delegates access to resources in one or more of the storage services. All of the operations available via a service or user delegation SAS are also available via an account SAS.

☐ Service SAS

☐ User delegation SAS

✓ That's correct. A user delegation SAS is secured with Azure Active Directory credentials and also by the permissions specified for the SAS. A user delegation SAS applies to Blob storage only.

2. Which of the following best practices provides the most flexible and secure way to use a service or account shared access signature (SAS)?

☒ Associate SAS tokens with a stored access policy.

✓ That's correct. The most flexible and secure way to use a service or account SAS is to associate the SAS tokens with a stored access policy.

☐ Always use HTTPS

☐ Implement a user delegation SAS

Next unit: Summary

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

[< Previous](#)

Unit 6 of 6

✓ 100 XP



Summary

3 minutes

In this module, you learned how to:

- Identify the three types of shared access signatures
- Explain when to implement shared access signatures
- Create a stored access policy

Module complete:

[Unlock achievement](#)

How are we doing? ☆ ☆ ☆ ☆ ☆

