# Amazon Web Services:
## Browser-Based Uploads Using POST and Signature Version 4

**Created by Zarsha Mian**

# Amazon Web Services
## Browser-Based Uploads Using POST and Signature Version 4

*Table of Contents*

## Browser-Based Upload Using HTTP POST

At this point, you should have a functional website that is available to the public; you can create and upload HTML files to your root domain bucket to add pages to your website, make documents, images, and other files available for users to download, and perform various other functions. What if you want to give users the ability to upload objects on your website into your S3 bucket? To do this, you will use an HTTP POST request to upload content directly into Amazon S3. You will also learn about and use the Signature Version 4 signing process. Using HTTP POST to upload content simplifies uploads and reduces upload latency where users upload data to store in Amazon S3.

This process will require you to obtain your IAM security credentials, create a POST policy, encode it in Base64, calculate a signature, and create an HTML form for your website that can be used to upload objects into the bucket.

## Signature Version 4

Signature Version 4 is the process to add authentication information to AWS requests sent by HTTP. For security, most requests to AWS must be signed with an access key. This access key consists of two parts: an access key ID and secret access key, also known as your security credentials.

Similar to a username and password, you can use your access key ID and secret access key together to authenticate your requests. By providing your access keys to a third party, you might give someone full access to your account. For this reason, it is important to manage your access keys as securely as possible.

When you create access keys, you create the access key ID and secret access key as a set. During creation, AWS gives you one opportunity to view and download the secret access key part of the access key. If you fail to download it or lose it, you have the option to delete the access key and create a new one. For more information about access keys and obtaining your security credentials, go to **Identity and Access Management (IAM)**.

Signature Version 4 has four steps:

1. Create a canonical request.
2. Use the canonical request and additional metadata to create a string for signing. This string is Base64-encoded.
3. Derive a signing key from your AWS secret access key. Then use the signing key, and the string from the previous step, to create a signature.
4. Add the resulting signature to the HTTP request in a header or as a query string parameter.

When an AWS service receives the request, it performs the same steps that you did to calculate the signature you sent in your request. AWS compares its calculated signature to the one you sent with the request. If the signatures match, the request is processed. If the signatures don’t match, the request is denied.

## Identity and Access Management (IAM)

AWS Identity and Access Management, or IAM, is used to manage access to Amazon S3 resources. It controls the type of access that users have to specific parts of an S3 bucket. You must use IAM to obtain the security credentials (access key ID and secret access key) needed to calculate a signature for sending a HTTP request. Security credentials are associated with a user, but first we will create a policy that will be attached to the user. To get your security credentials, follow these steps:

1. Go to the **IAM Management console**.
2. Under Access Management, go to **Policies**.
3. Click *Create Policy*.
4. Click *JSON*.
5. Enter the policy you wish to associate with the user you will later create in the text editor. If you do not have a policy of your own, copy and paste the policy below into the text editor. In the **Resource** section, replace *your-bucket-name* with the name of your root domain bucket.

```
{
   “Version”: “2012-10-17”,
   “Statement”: [
      {
         “Effect”: “Allow”,
         “Action”: [
            “s3:Put*”
         ],
         “Resource”: [
            “arn:aws:s3:::your-bucket-name/*”
         ]
      }
   ]
}
```

6. Click *Review Policy*.
7. Enter the name of your policy in the *Name* field.
8. Add a description of your policy’s function in the *Description* field. **NOTE:** This step is optional.

9. Read the summary information. The policy mentioned in this tutorial provides limited access, namely write, permissions management, and tagging.
10. Click *Create Policy*. You will be returned to the **Policies** page of the IAM console, and will receive an alert saying *"policy-name has been created"*.

Once the policy is created and saved, you will be able to find it by going under Access Management to **Policies**. You can filter the policies or search for a specific policy by name in the *search box*. Your policy will be shown to be Customer Managed.

Now that you have created the policy that you want to attach to your user, you will create the new user by doing the following:

11. Go to the **IAM Management console**.
12. Under Access Management, go to **Users**.
13. Click *Add User*.
14. Enter a user name in the *User name* field. **NOTE:** You can add multiple users at once with the same access type and permissions.
15. Under **Select AWS access type**, select *Programmatic Access*. **NOTE:** This enables an access key ID and secret access key for the AWS API, CLI, SDK, and other development tools.
16. Click *Next: Permissions*.
17. Click the *Attach Existing Policies Directly* tab.
18. Search the name of the policy you previously created during step 5 in the search bar.
19. Click the *check mark* next to the policy's name.
20. Click *Next: Tags*.
21. Add *IAM key-value pairs* to your user. These tags can include user information, such as an email address, or can be descriptive, such as a job title. These tags can be used to organize, track, or control access for this user. **NOTE:** This step is not required.
22. Click *Next: Review*.
23. Review the information under **User details** to confirm the information is correct.
24. Review the information under **Permissions summary** to confirm that you attached the correct policy.
25. Review any tags that were added in the **Tags** section.

26. Click *Create User*.

You will get an alert that says that the user creation was successful, and you will be assigned your security credentials. You will be given the option to view and download these new user security credentials, which include your access key ID and secret access key. This will be your only chance to download your credentials, but you can create new credentials at any time. Click *Download .csv* to download your credentials in an Excel spreadsheet.

The security credentials that you have been assigned are necessary when calculating your signature for the HTTP request. Make sure you have this information available to you and stored in a safe place.

If you wish to see any information related to your user:

1. Go to the **IAM console**.
2. Under Access Management, go to **Users**.
3. Click the *name* of the user. This will provide a general summary of the user, and will also have **Permission, Group, Tags, Security Credentials, and Access Advisor** tabs for you to go through in order to find more information about your user and any attached policies.

## Creating a POST Policy

The policy is the Base64-encoded security policy that describes what is permitted in the request. For signature calculation, this policy is the string to sign value. Amazon S3 must get this policy so that it can re-calculate the signature. As mentioned previously, the first step in the Signature Version 4 signing process is creating a canonical request, which requires you to arrange the contents of your request (host, action, headers, etc.) into a standard or canonical format. This canonical request is one of the inputs used to create a string to sign, which is used in the next step of the process.

A POST policy should include certain fields to provide signature and relevant information that Amazon S3 can use to re-calculate the signature upon receiving the request. It should include specific conditions, such as:

- Expiration: the upload must occur before the date and time specified in this condition. The expiration is a deadline that marks the end of the policy's availability.
- Bucket: the name of the bucket the content can be uploaded into. The bucket must be in the region that is specified in the credential (x-amz-credential form parameter) that will be explained later on. The signature you provide is valid only within this scope.
- Starts with: sets a condition for what the file or upload name must begin with. For example, an upload can be performed if the key name starts with user/user1, like user/user1/MyPhoto.jpg.
- ACL: access control lists enable you to manage access to buckets and objects. For example, the ACL may be set to "public-read", giving the public read permissions.
- Successful action redirect: if the upload succeeds, the user's browser will be redirected to the mentioned page or URL. For example, you may have a redirection to a successful_upload.html page that is stored in your bucket.
- Content type: describes the kind of file that may be uploaded. For example, the object may be uploaded only if it is an image file.
- Credential: in addition to your access key ID, this provides scope information you used in calculating the signing key for signature calculation. It is a string in the following form: **<your-access-key-id>/<date>/<aws-region>/<aws-service>/aws4_request**

The date refers to the date of the policy’s creation, and the region must match the region of the root domain bucket. For Amazon S3, the aws-service string is s3.

- Algorithm: the algorithm used to calculate the signature. You must provide this value when you use AWS Signature Version 4 for authentication. The signing algorithm for AWS Signature Version 4 is **AWS4-HMAC-SHA256**.
- Date: date value in ISO8601 format. For example, 20130728T000000Z, represents the year (YYYY), month (MM), date (DD), hour (HH), minute (MM), and second (SS). This is the same date you use in creating your signing key. It must also be the same value you provide in the policy that you sign.

## Sample POST Policy

You are able to create a policy based on the criteria you want your policy to have. For example, you may want a policy that allows only .pdf uploads until midnight on December 31st, 2021 into a bucket named example.com. If you do not have your own policy, you may use the following sample POST policy for the remainder of this tutorial. Highlighted in red are the values that you would need to change according to your own specifications.

```
{ "expiration": "2020-12-31T12:00:00.000Z",
  "conditions": [
    {"bucket": "sigv4examplebucket"},
    ["starts-with", "$key", "user/user1/"],
    {"acl": "public-read"},
    {"success_action_redirect":
"http://sigv4examplebucket.s3.amazonaws.com/successful_upload.html"},
    ["starts-with", "$Content-Type", "image/"],
    {"x-amz-meta-uuid": "14365123651274"},
    {"x-amz-server-side-encryption": "AES256"},
    ["starts-with", "$x-amz-meta-tag", ""],

    {"x-amz-credential": "AKIAIOSFODNN7EXAMPLE/20200101/us-east-
2/s3/aws4_request"},
    {"x-amz-algorithm": "AWS4-HMAC-SHA256"},
    {"x-amz-date": "20200101T000000Z" }
  ]
}
```

This POST policy sets the following conditions on the request:

- The upload must occur before noon UTC on December 31, 2020. Since the policy was created on January 1, 2020 (check the date and credential conditions that contain the policy's creation date), that means that the policy is valid for the duration of one year.
- The content can only be uploaded into the bucket sigv4examplebucket. The region of the bucket is us-east-2, which is why the region us-east-2 is mentioned in the credential condition.
- You can provide any key name that starts with user/user1. **NOTE:** If you want to be the only person who can use the HTML form and the policy associated with it to upload directly into your bucket, you could set this value to the name of the user account that you use on your computer, such as user/*your-name*.

- The ACL is set to public read, granting the public read permissions.
- If the upload succeeds, the user's browser is redirected to sigv4examplebucket.s3.amazonaws.com/successful_upload.html.
- The object must be an image file.
- The x-amz-meta-uuid tag must be set to 14365123641274.
- The x-amz-meta-tag can contain any value.

**NOTE:** When creating a POST policy, it is vital that you replace the example access key ID mentioned above with your user's access key ID.

## String to Sign: Base64-Encode

---

The string to sign includes meta information about the request you created earlier in **Creating a POST Policy**. You will use the string to sign and a derived key that you create later as inputs to calculate the request signature later on.

In order to encode the POST policy into string to sign, you can use an online encoder such as the one found at www.base64encode.org. In order to do so, follow these steps:

1. Copy and paste your POST policy into the text editor. When you copy and paste the policy, it should have carriage returns and new lines for your computed hash to match the correct Base64-encoded value. **NOTE:** If you were using the above mentioned policy, for example, you would copy the policy from the beginning "{" to the ending "}". Make sure there are no new lines at the end of the policy when copying it into the editor, because this will change the encoded result.
2. Set the *Destination character set* value. By default, it is set to UTF-8. **NOTE:** Leave this default setting if you are using the sample POST policy provided in this tutorial.
3. Set the *Destination newline separator* value. In this tutorial, we are using CRLF (Windows). **NOTE:** The Base64-encoded version changes based on these settings, but we need to use the string to sign that Amazon will calculate and match to our value in order to accept our request. The setting we are using will work even for Macs.
4. Click the *Encode* button. The Base64-encoded result is output in the *Results* box. The string to sign value of the sample POST policy mentioned earlier (as in, without any changes made to any field, including the credential parameter with the **example** access key ID) is the following:

**eyAiZXhwaXJhdGlvbiI6ICIyMDIwLTEyLTMxVDEyOjAwOjAwLjAwMFoiLA0K
ICAiY29uZGl0aW9ucyI6IFsNCiAgICB7ImJ1Y2tldCI6ICJzaWd2NGV4YW1wbG
VidWNrZXQifSwNCiAgICBbInN0YXJ0cy13aXRoIiwgIiRrZXkiLCAidXNlci91c2
VyMS8iXSwNCiAgICB7ImFjbCI6ICJwdWJsaWMtcmVhZCJ9LA0KICAgIHsic3
VjY2Vzc19hY3Rpb25fcmVkaXJlY3QiOiAiaHR0cDovL3NpZ3Y0ZXhhbXBsZWJ1
Y2tldC5zMy5hbWF6b25hd3MuY29tL3N1Y2Nlc3NmdWxfdXBsb2FkLmh0bWwif
SwNCiAgICBbInN0YXJ0cy13aXRoIiwgIiRDb250ZW50LVR5cGUiLCAiaW1hZ2U
vIl0sDQogICAgeyJ4LWFtei1tZXRhLXV1aWQiOiAiMTQzNjUxMjM2NTEyNzQif
SwNCiAgICB7IngtYW16LXNlcnZlci1zaWRlLWVuY3J5cHRpb24iOiAiQUVTMjU
2In0sDQogICAgWyJzdGFydHMtd2l0aCIsICIkeC1hbXotbWV0YS10YWciLCAiIl0
sDQoNCiAgICB7IngtYW16LWNyZWRlbnRpYWwiOiAiQUtJQUlPU0ZPRE5ON0**

**VYQU1QTEUvMjAyMDAxMDEvdXMtZWFzdC0yL3MzL2F3czRfcmVxdWVzdC
J9LA0KICAgIHsieC1hbXotYWxnb3JpdGhtIjogIkFXUzQtSE1BQy1TSEEyNTYif
SwNCiAgICB7IngtYW16LWRhdGUiOiAiMjAyMDAxMDFUMDAwMDAwWiIgf
Q0KICBdDQp9**

For more information on how to calculate the string to sign value manually without the use of an online Base64 encoder, go to the following link: https://docs.aws.amazon.com/general/latest/gr/sigv4-create-string-to-sign.html.

## Calculating the Signature

---

Before you calculate the signature, you will derive a signing key from your AWS secret access key. Because the derived signing key is specific to the date, service, and region, it offers a greater degree of protection. You then use the signing key and the string to sign that you created earlier as the inputs to a keyed hash function. The hex-encoded result from the keyed hash function is the signature. The Ruby code for signature calculation in its entirety will be shown at the end of this section.

1. Derive your signing key. To do this, use your secret access key to create a series of hash-based messaging authentication codes (HMACs). The following Ruby function will create your signing key.

   **NOTE:** The dateStamp used in the hashing process is in the format YYYYMMDD (for example, 20200101) and does not include the time. It must match the date from your POST policy's credential and date fields. The key parameter refers to your secret access key, regionName refers to the region specified in your POST policy, and the serviceName refers to the service which, in this case, is S3.

   OpenSSL is a gem used by Ruby to provide SSL, TLS, and general-purpose cryptography. It is generally included in a Ruby installation. You must load OpenSSL with "**require 'openssl'**" in your code in order for this function to run properly.

```
def getSignatureKey(key, dateStamp, regionName, serviceName)
  kDate      = OpenSSL::HMAC.digest('sha256', "AWS4" + key, dateStamp)
  kRegion   = OpenSSL::HMAC.digest('sha256', kDate, regionName)
  kService = OpenSSL::HMAC.digest('sha256', kRegion, serviceName)
  kSigning = OpenSSL::HMAC.digest('sha256', kService, "aws4_request")
  kSigning
end
```

2. Calculate the signature. Use the signing key that you derived in step 1 and the string to sign you calculated in **String to Sign: Base64-Encode** as inputs to the keyed hash function.
3. After you calculate the signature, convert the binary value to a hexadecimal representation. The following Ruby code will accomplish steps 2 and 3.

```
signature = OpenSSL::HMAC.hexdigest('sha256', signing_key, string_to_sign)
```

The following Ruby code will perform signature calculation:

```
require 'uri'
require 'openssl'
require 'net/http'
require 'cgi'

def getSignatureKey(key, dateStamp, regionName, serviceName)
  kDate    = OpenSSL::HMAC.digest('sha256', "AWS4" + key, dateStamp)
  kRegion  = OpenSSL::HMAC.digest('sha256', kDate, regionName)
  kService = OpenSSL::HMAC.digest('sha256', kRegion, serviceName)
  kSigning = OpenSSL::HMAC.digest('sha256', kService, "aws4_request")
  kSigning
end

secret_key = ‘enter your secret access key here’
datestamp = '20200101' # enter date (from policy) in YYYYMMDD format
region = 'us-east-2' # enter the region (from policy) here
service = 's3'

string_to_sign = ‘enter your string to sign value here’

signing_key = getSignatureKey(secret_key, datestamp, region, service)
signature = OpenSSL::HMAC.hexdigest('sha256', signing_key, string_to_sign)
print signature
```

Copy and paste the above code into an editor, like Xcode or Emacs, replace the values in red with your own values (your secret access key, and the date, region, and string to sign associated with your POST policy), and save as a .rb (Ruby) file. Run it to have your signature calculated.  Your signature will be unique based on the information you provide, but here is simply an example of what a signature looks like:

**8afdbf4008c03f22c2cd3cdb72e4afbb1f6a588f3255ac628749a66d7f09699e**

This Ruby code will provide you with the calculated signature that you need for your request. As mentioned earlier, Amazon S3 will recalculate the signature on its own and determine if it matches the signature you calculated in this section.

## HTML Form for Uploads to Amazon S3

Now that you have calculated the signature, you can add it to the request. The following example form specifies the preceding POST policy and supports a POST request to the sigv4examplebucket. Copy and paste the content into a text editor and save it as exampleform.html, and then upload it into your bucket (If you don't remember how to upload content into your Amazon S3 bucket, go to the **Uploading Index and Website Content** section). You can then upload to the specific bucket using the exampleform.html. Your request will succeed if the signature you provide matches the signature Amazon S3 calculates. **NOTE:** You must update the bucket name, dates, credential, policy, and signature with valid values for this to successfully upload to S3.

```
<html>
  <head>

    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

  </head>
  <body>

  <form action="http://sigv4examplebucket.s3.amazonaws.com/" method="post"
enctype="multipart/form-data">
    Key to upload:
    <input type="input"  name="key" value="user/user1/${filename}" /><br />
    <input type="hidden" name="acl" value="public-read" />
    <input type="hidden" name="success_action_redirect"
value="http://sigv4examplebucket.s3.amazonaws.com/successful_upload.html" />
    Content-Type:
    <input type="input"  name="Content-Type" value="image/jpeg" /><br />
    <input type="hidden" name="x-amz-meta-uuid" value="14365123651274" />
    <input type="hidden" name="x-amz-server-side-encryption" value="AES256" />
    <input type="text"   name="X-Amz-Credential"
value="AKIAIOSFODNN7EXAMPLE/20200101/us-east-2/s3/aws4_request" />
    <input type="text"   name="X-Amz-Algorithm" value="AWS4-HMAC-SHA256" />
    <input type="text"   name="X-Amz-Date" value="20200101T000000Z" />

    Tags for File:
    <input type="input"  name="x-amz-meta-tag" value="" /><br />
    <input type="hidden" name="Policy" value='<Base64-encoded policy string>' />
    <input type="hidden" name="X-Amz-Signature" value="<signature-value>" />
    File:
```

```
   <input type="file"   name="file" /> <br />
   <!-- The elements after this will be ignored -->
   <input type="submit" name="submit" value="Upload to Amazon S3" />
  </form>

</html>
```

In the *Policy* field, be sure to enter in your string to sign value, also known as your Base64-encoded policy string. In the *Signature* field, replace <signature-value> with your calculated signature.

When the exampleform.html file is in your bucket, you need to provide a link to your website's users so that they may see the form and use it to upload into your bucket. To do this, you can create a link on one of your website's pages to the exampleform.html page. For example, you may add the following line somewhere inside your index.html file:

**<p><a href=“exampleform.html”>Upload to S3 Form</a></p>**

Doing so will allow the user to navigate to exampleform.html once they click the Upload to S3 Form link. There, they will be able to choose a file and upload it to Amazon S3. If the signature Amazon S3 calculates matches the signature provided in the HTML form, the request will be approved and the upload will be found in your Amazon S3 bucket. It also must match your specifications, such as the content type in order to be uploaded. If you allow only image files and the user tries to upload a Microsoft Word document, the upload will fail.

For more information on Browser-Based Uploads using HTTP Post and AWS Signature Version 4, go to https://docs.aws.amazon.com/AmazonS3/latest/API/sigv4-post-example.html. This will provide you with another similar example, and will walk through the process of uploading content directly to Amazon S3.