

Good Workaround!

Digging into Azure AD Certificate-Based Authentication

 Marius Solbakken  February 15, 2022

Azure AD Certificate-Based Authentication is now in public preview, with [a surprisingly good documentation](#). Usually I have to guess how 50% of a feature actually works, but this time they have gone all-in with technical details of just about everything. What is a blogger to do? Well, let's configure it and see if we can sneak a peek behind the scenes 😊



Comment

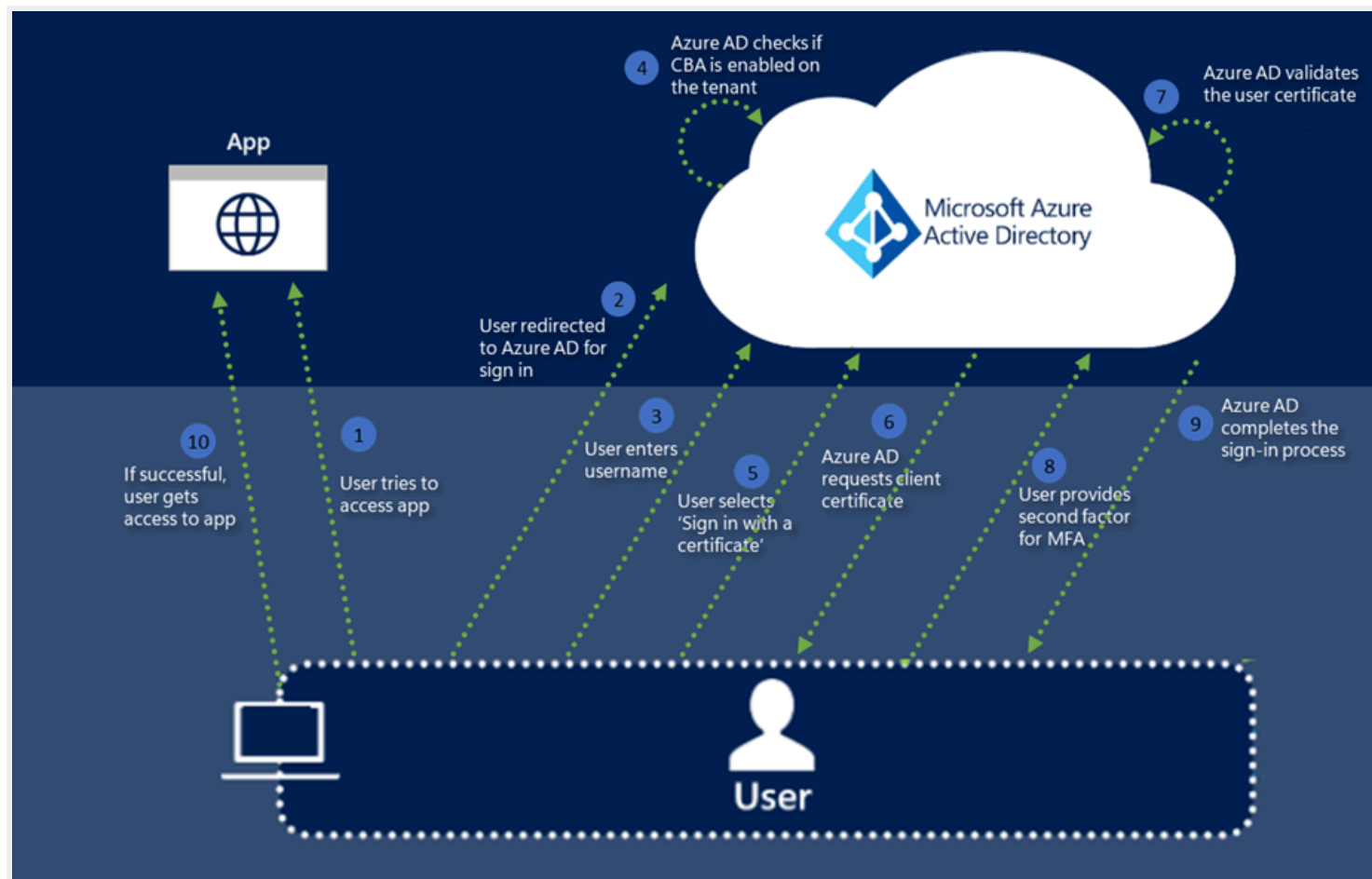


Reblog



Subscribe





So, to start, let's configure a certificate authority using plain OpenSSL. Essentially I will then have a certificate with a private key in a file locally on my computer, upload the public version of the certificate to Azure AD as a trusted root certificate authority and issue user certificates with the private key.

Please note: This is not a recommended configuration at all, as it does not have things like Certificate Revocation List, but it works great for testing.

We start by installing [OpenSSL](#) and follow [this 11 year old guide](#), apparently still relevant.

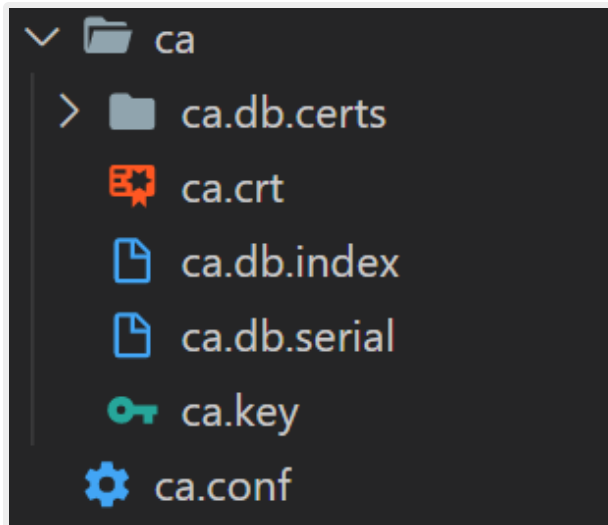
We can then use the following PowerShell script to create a CA:

```
1 | $openssl = 'C:\Program Files\OpenSSL'
```

Comment Reblog Subscribe

```
2
3 Set-Content -Path ca.conf -Value '[ ca ]
4 default_ca = ca_default
5 [ ca_default ]
6 dir = ./ca
7 certs = $dir
8 new_certs_dir = $dir/ca.db.certs
9 database = $dir/ca.db.index
10 serial = $dir/ca.db.serial
11 RANDFILE = $dir/ca.db.rand
12 certificate = $dir/ca.crt
13 private_key = $dir/ca.key
14 default_days = 365
15 default_crl_days = 30
16 default_md = md5
17 preserve = no
18 policy = generic_policy
19 [ generic_policy ]
20 countryName = optional
21 stateOrProvinceName = optional
22 localityName = optional
23 organizationName = optional
24 organizationalUnitName = optional
25 commonName = optional
26 emailAddress = optional'
27
28 # Create folders
29 !(Test-path "ca") ? (mkdir "ca" | Out-Null) : $null
30 !(Test-path "ca/ca.db.certs") ? (mkdir "ca/ca.db.certs" | Out-Null) : $
31
32 # Fill initial files
33 Set-Content -Path "ca/ca.db.index" -Value ""
34 Set-Content -Path "ca/ca.db.serial" -Value "1234"
35
36 # Generate a 1024-bit RSA private key for the CA
37 . $openssl genrsa -des3 -out ca/ca.key 4096
38
39 # Create a self-signed X509 certificate for the CA (the CSR will be sig
40 . $openssl req -new -x509 -days 10000 -key ca/ca.key -out ca/ca.crt
```

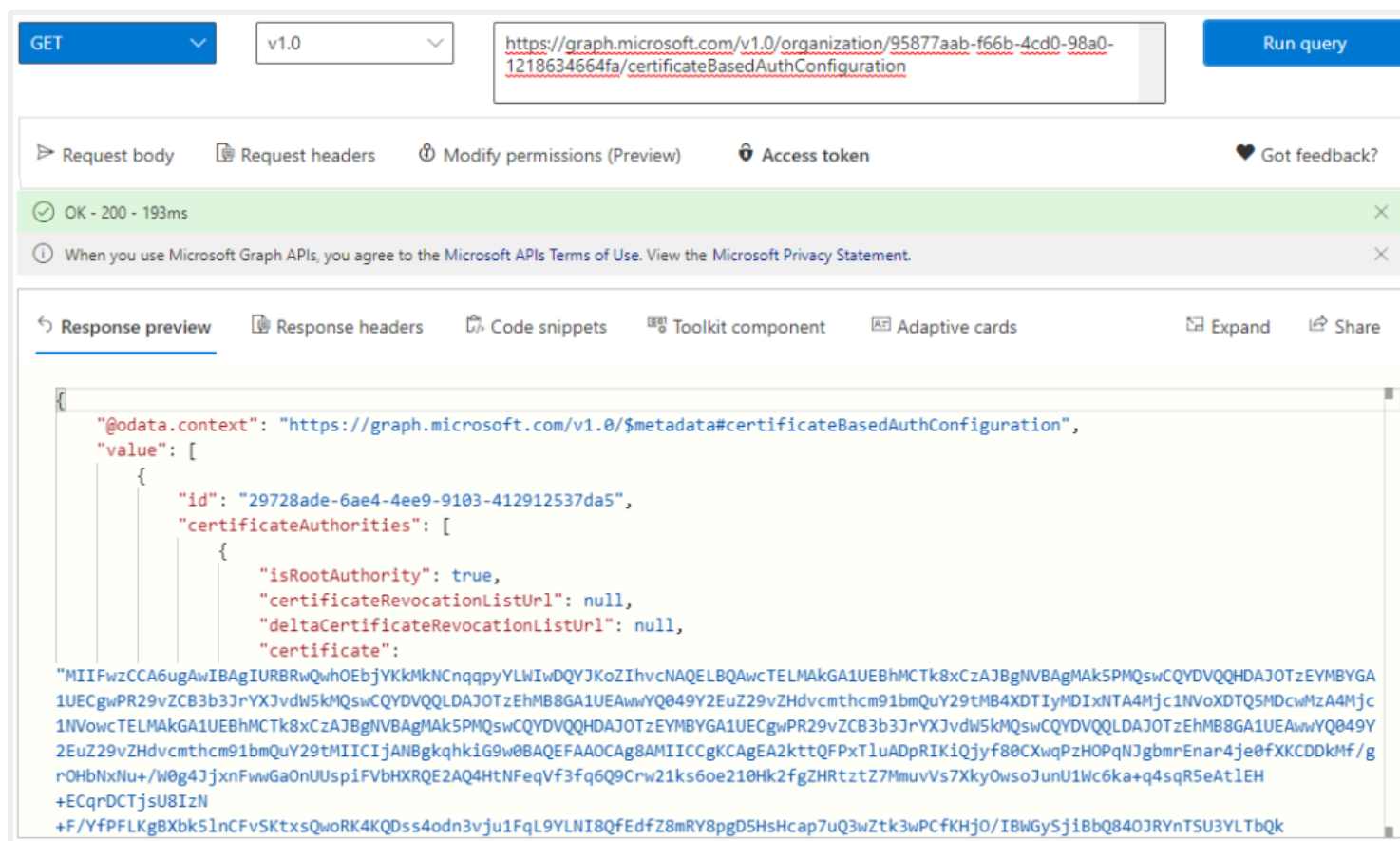
We will then have a folder structure that looks something like this:



The ca.crt file is our public key that we should upload to Azure AD. Let's do that using Microsoft Graph endpoint:

```
1 $accessToken = "eyJ0eXAiOiJKV1QiLCJub.....vtPWqjha"
2 $tenantid = "95877aab-f66b-4cd0-98a0-1218634664fa"
3 $file = "ca/ca.crt"
4
5 $body = @{
6     certificateAuthorities = @(
7         @{
8             isRootAuthority = $true
9             certificate = [Convert]::ToBase64String([System.IO.File]::R
10         }
11     )
12 } | ConvertTo-Json -Depth 10
13
14 Invoke-RestMethod -Uri "https://graph.microsoft.com/v1.0/organization/$
```

We can now see that a certificate authority has been added:



GET v1.0 <https://graph.microsoft.com/v1.0/organization/95877aab-f66b-4cd0-98a0-1218634664fa/certificateBasedAuthConfiguration> Run query

Request body Request headers Modify permissions (Preview) Access token Got feedback?

OK - 200 - 193ms

When you use Microsoft Graph APIs, you agree to the Microsoft APIs Terms of Use. View the Microsoft Privacy Statement.

Response preview Response headers Code snippets Toolkit component Adaptive cards Expand Share

```
{
  "@odata.context": "https://graph.microsoft.com/v1.0/$metadata#certificateBasedAuthConfiguration",
  "value": [
    {
      "id": "29728ade-6ae4-4ee9-9103-412912537da5",
      "certificateAuthorities": [
        {
          "isRootAuthority": true,
          "certificateRevocationListUrl": null,
          "deltaCertificateRevocationListUrl": null,
          "certificate":
            "MIIFwzCCA6ugAwIBAgIURBRwQwOEbjYKkMkNCnqppyYlWiDQYJKoZIhvcNAQELBQAwTElMAkGA1UEBhMCTk8xCzAJBgNVBAGMAk5PMQswCQYDVQQHDAJOTzEYMBYGA1UECgwPR29vZCB3b3JrYXJvdW5kMQswCQYDVQQLDAJOTzEhMB8GA1UEAwYQ049Y2EuZ29vZHVcmthcm91bmQuY29tMB4XDTIyMDIxNTA4Mjc1NVowcTElMAkGA1UEBhMCTk8xCzAJBgNVBAGMAk5PMQswCQYDVQQLDAJOTzEYMBYGA1UECgwPR29vZCB3b3JrYXJvdW5kMQswCQYDVQQLDAJOTzEhMB8GA1UEAwYQ049Y2EuZ29vZHVcmthcm91bmQuY29tMIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAGEA2kttQFPxTluADpRIKiQjyf80CXwqPzH0PqN3gbmrEnar4je0fXKCDdkMf/g
            rOHbNxnNu+/w0g4JjxnFwwGaOnUUsPiFVbHXRQE2AQ4HtNFeqVF3fq6Q9Crw21ks6oe210Hk2fgZHRtz7MmuvVs7XkyOwsoJunU1Wc6ka+q4sqR5eAt1EH
            +ECqrDCTjsU8IzN
            +F/YfPFLKgBxbk51nCFvSKtxsQwoRK4KQDss4odn3vju1FqL9YLNi8QfEdfZ8mRY8pgD5HsHcap7uQ3wZtk3wPCFKHjO/IBWgySj1BbQ840JRYnTSU3YLTbQk
        }
      ]
    }
  ]
}
```

Quick question – If I were to add an additional trusted CA to your tenant, would you notice?

We can now configure the CBA feature:

Home > Kakao AS > Security >

Authentication methods | Policies

Kakao AS - Azure AD Security

Search (Ctrl+/) << Got feedback?

Manage

- Policies**
- Password protection
- Registration campaign

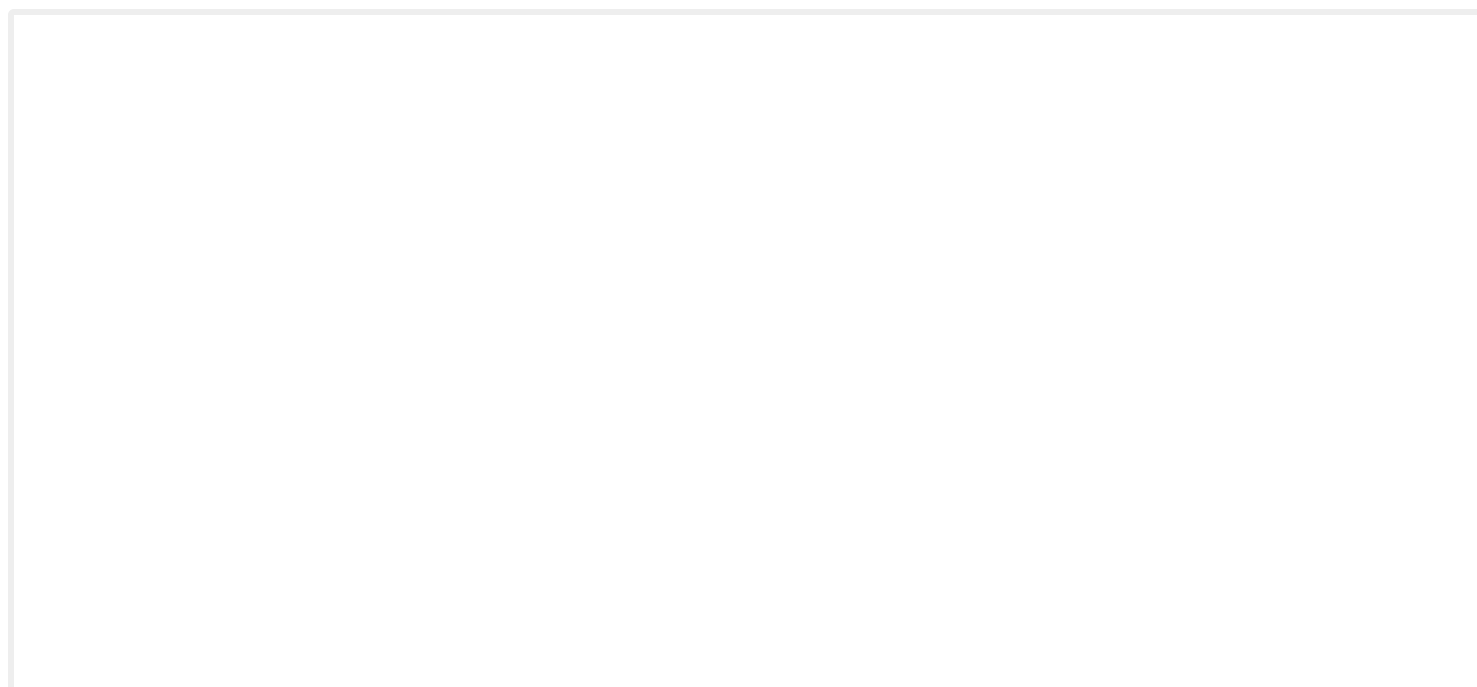
Monitoring

- Activity
- User registration details
- Registration and reset events
- Bulk operation results

Configure your users in the authentication methods policy to enable passwordless authentication. Once configured your users for the enhanced registration preview so they can register these authentication methods and use the

Method	Target	Enabled
FIDO2 Security Key		No
Microsoft Authenticator		No
Text message (preview)		No
Temporary Access Pass (preview)		No
Certificate-based authentication (preview)	All users	Yes

Since this is a test tenant, I simply enable for all users:



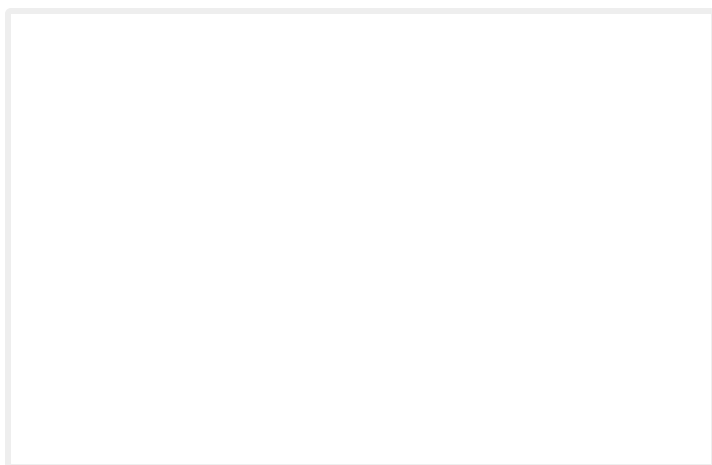
And, now we are ready to create a user certificate:

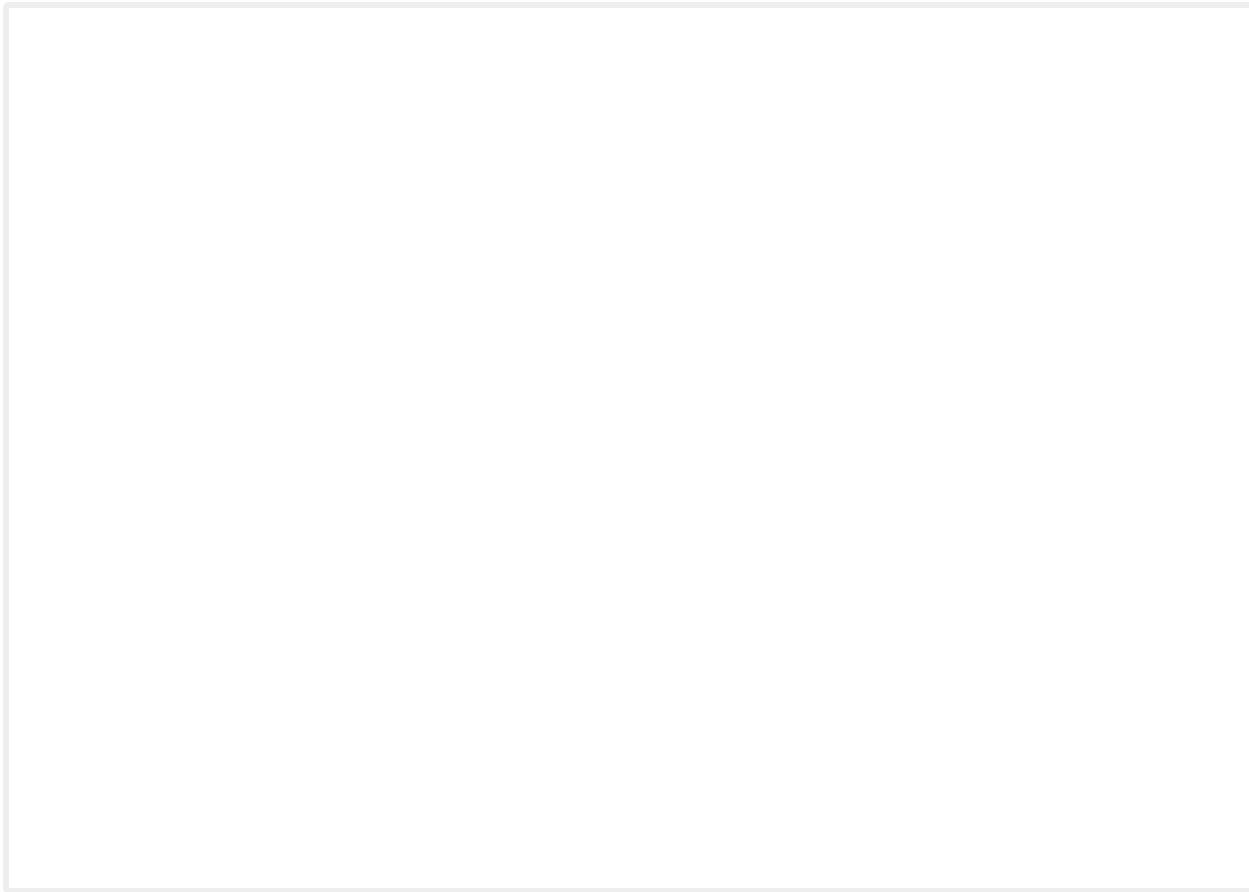
```
1 | $userPrincipalName = "AllanD@M365x912454.OnMicrosoft.com"
2 |
3 | $openssl = 'C:\Program Files\OpenSSL'
```

Comment Reblog Subscribe ...

```
4  
5 # Create CSR  
6 #. $openssl req -new -sha256 -newkey rsa:4096 -nodes -keyout "$userPrin  
7 . $openssl req -new -sha256 -newkey rsa:4096 -nodes -keyout "$userPrinc  
8  
9 # Sign CSR  
10 . $openssl ca -md sha256 -config ca.conf -out "$userPrincipalName-certi  
11  
12 # Create PFX  
13 . $openssl pkcs12 -inkey "$userPrincipalName-key.pem" -in "$userPrincip
```

After running this, we now have a pfx file that we can import locally:





We should see that the SAN field contains “Principal Name”:



Comment

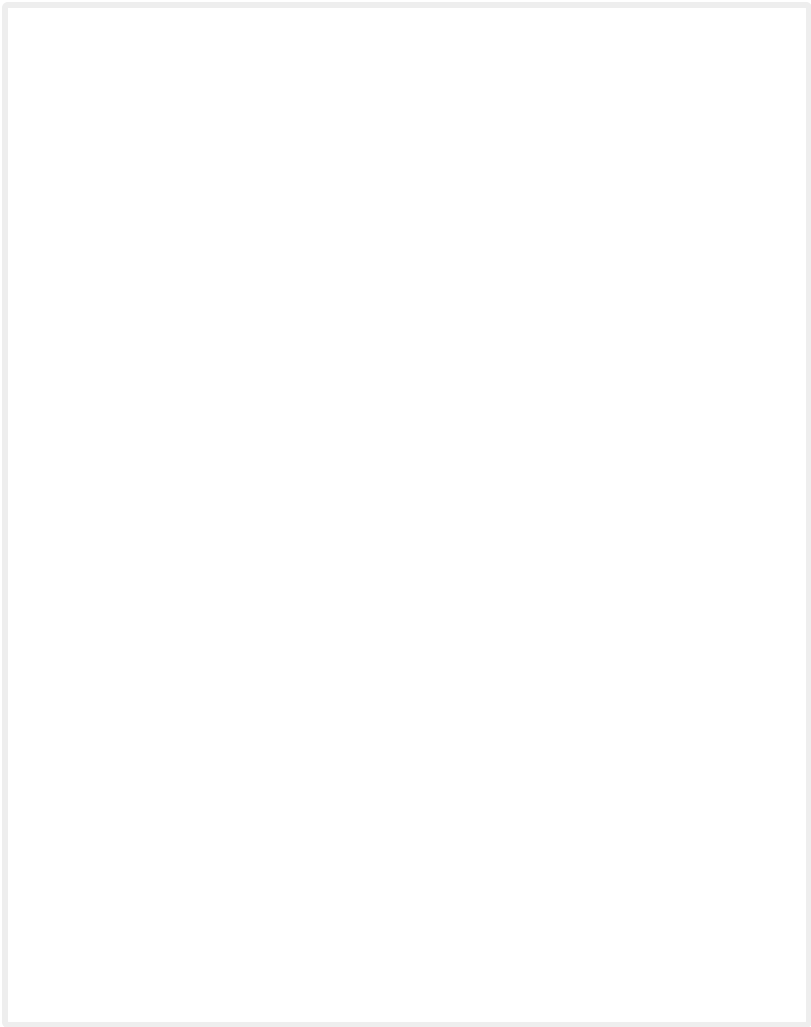


Reblog

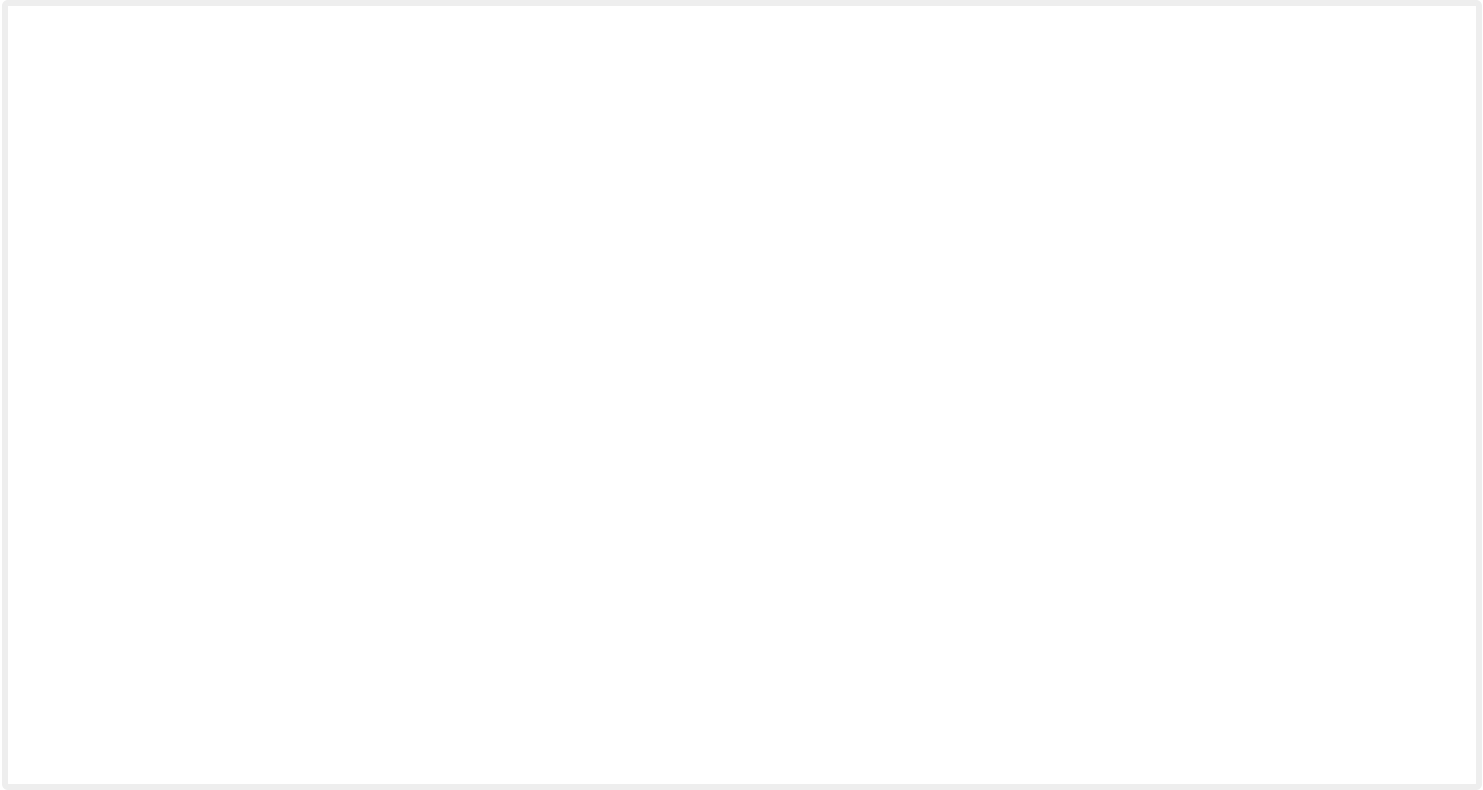


Subscribe



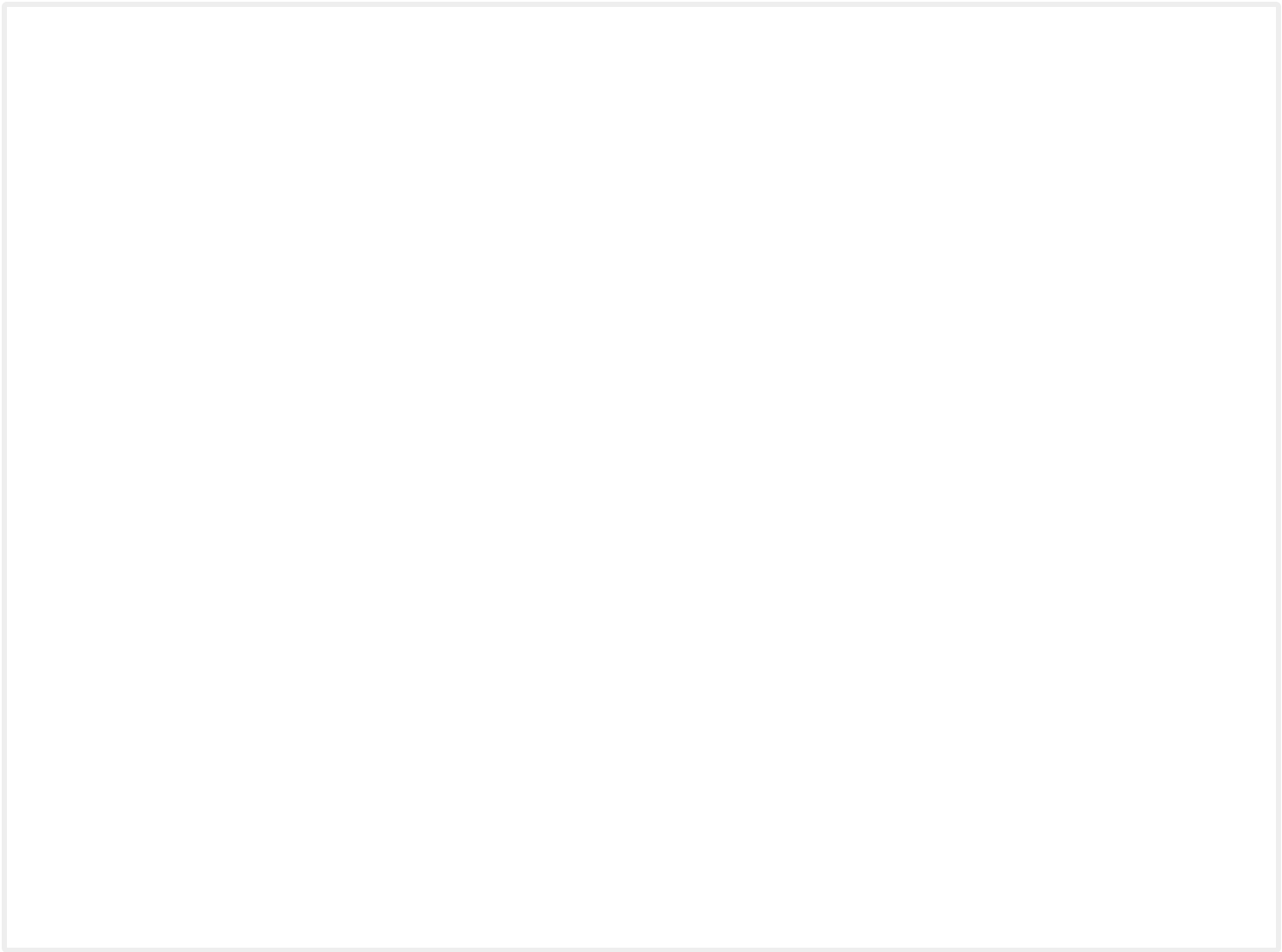


This should match our CBA configuration:



Let’s test!

After we type our username, we get **Sign in with a certificate** as an option.



After clicking, we are told to choose which certificate to use when authenticating towards **certauth.login.microsoftonline.com**:



And for some reason it fails the first time:



Before it works after clicking **Sign in with a certificate** the second time...



And now it works:

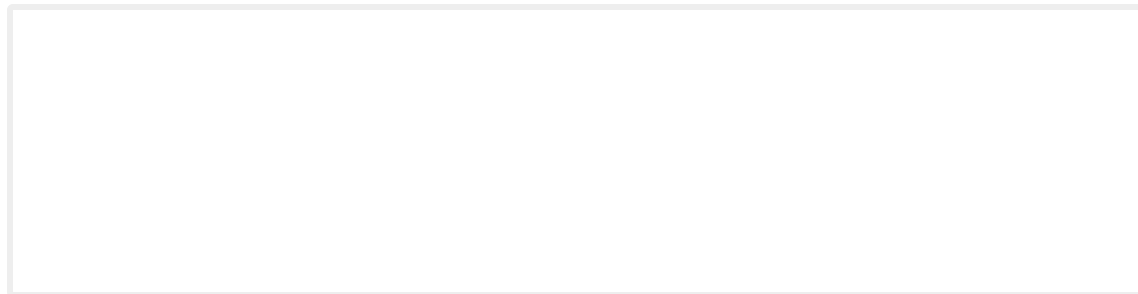


That might be a public preview issue, or something to do with my certificate configuration – I don't know, but we got it working!

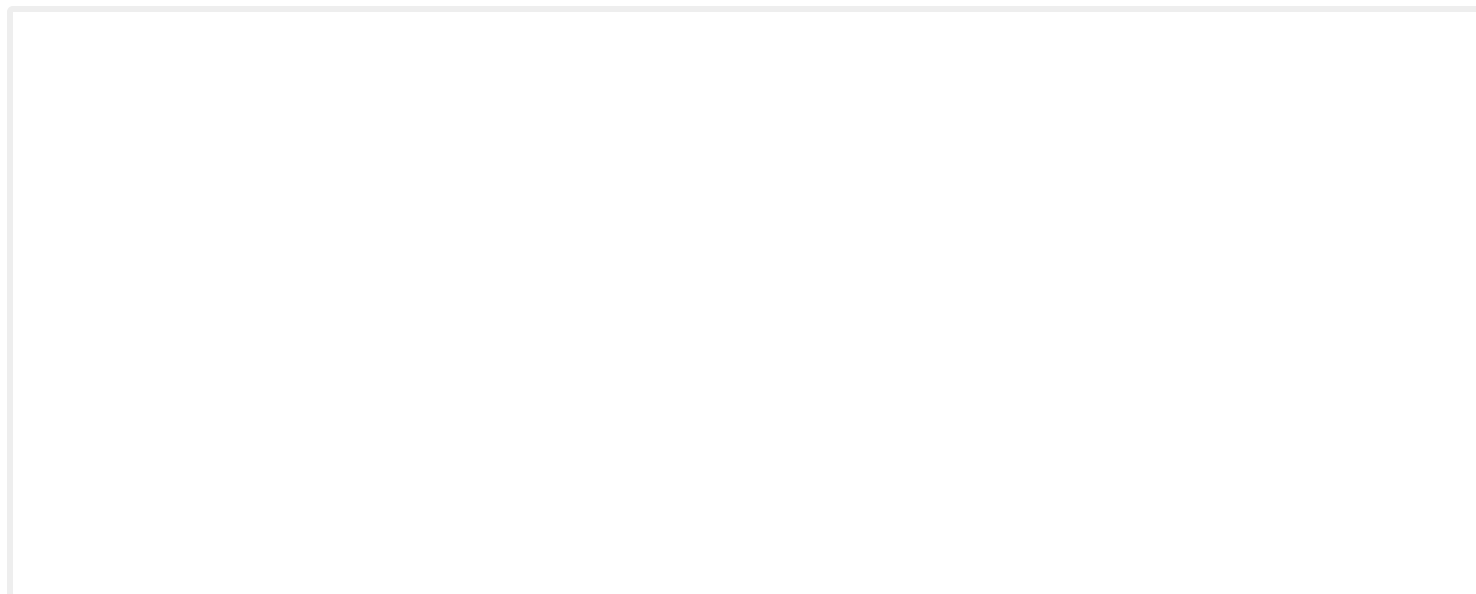
Let us now have a look at what happens behind the scenes. When clicking the **Sign in with a certificate** link, the following happens:

First, the url

<https://login.microsoftonline.com/common/GetCredentialType?mkt=en-US> is invoked:



There are a lot of parameters sent *out* to this endpoint, already discovered apparently:



And we get information about the certificate authentication, and a lot of other things:

```
1  {
2  "Username": "alland@m365x912454.onmicrosoft.com",
3  "Display": "alland@m365x912454.onmicrosoft.com",
4  "IfExistsResult": 0,
5  "IsUnmanaged": false,
6  "ThrottleStatus": 0,
7  "Credentials": {
8    "PrefCredential": 1,
9    "HasPassword": true,
10   "HasCertAuth": true,
11   "RemoteNgcParams": null,
12   "FidoParams": null,
13   "SasParams": null,
14   "CertAuthParams": {
15     "CertAuthUrl": "https://
```



Comment



Reblog

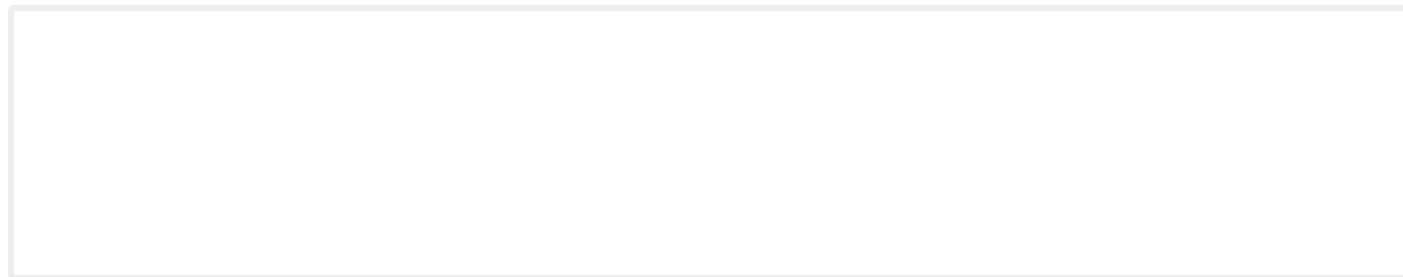


Subscribe

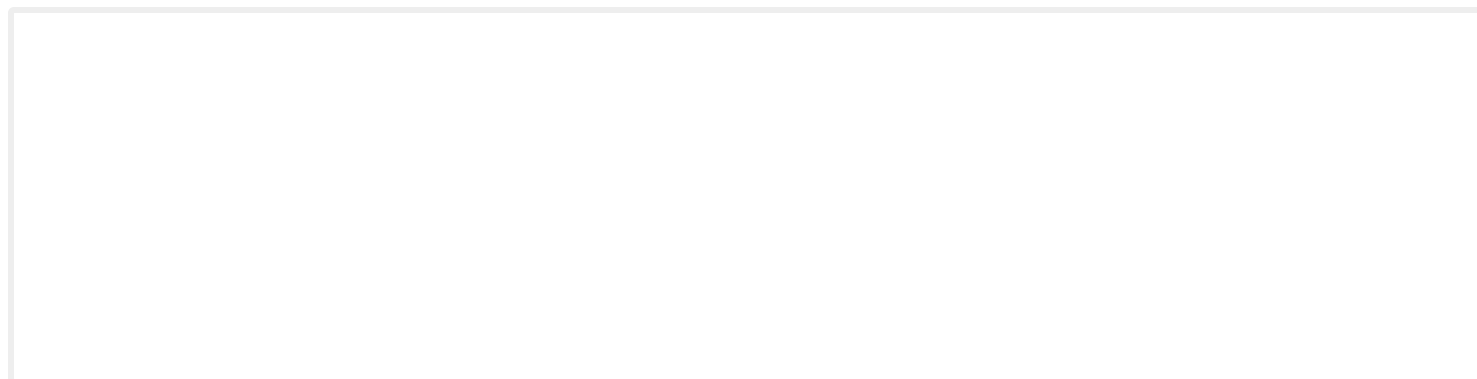



```
16      },
17      "GoogleParams": null,
18      "FacebookParams": null
19    },
20    "EstsProperties": {
21      "UserTenantBranding": [
22        {
23          "Locale": 0,
24          "BannerLogo": "https://aadcdn.msauthimages.net/c1c6b6c8-7",
25          "TileLogo": "https://aadcdn.msauthimages.net/c1c6b6c8-7",
26          "TileDarkLogo": "https://aadcdn.msauthimages.net/c1c6b6c8-7",
27          "Illustration": "https://aadcdn.msauthimages.net/c1c6b6c8-7",
28          "BoilerPlateText": "<p>Contoso</p>\n",
29          "KeepMeSignedInDisabled": false,
30          "UseTransparentLightBox": false,
31          "LayoutTemplateConfig": {
32            "showHeader": false,
33            "headerLogo": "",
34            "layoutType": 0,
35            "hideCantAccessYourAccount": false,
36            "hideForgotMyPassword": false,
37            "hideResetItNow": false,
38            "showFooter": true,
39            "hideTOU": false,
40            "hidePrivacy": false
41          },
42          "CustomizationFiles": {
43            "strings": {
44              "adminConsent": "",
45              "attributeCollection": "",
46              "authenticatorNudgeScreen": "",
47              "conditionalAccess": ""
48            },
49            "customCssUrl": ""
50          }
51        }
52      ],
53      "DomainType": 3
54    },
55    "FlowToken": "AQABAAEAAAD--DLA3V07Q....D8VmBUct-YdmhU7iLRpSAA",
56    "IsSignupDisallowed": true,
57    "apiCanary": "AQABAAAAAAD--DLA3V07Q....36TiDv19674vTOi3TiAA"
58  }
```

The **FlowToken** is then sent to the url found in **CertAuthUrl**, as well as a ctx token (for tracking purposes, I believe):



A POST to <https://certauth.login.microsoftonline.com/TENANTID/certauth> happens, with two parameters:



There is no response at all, other than **200 OK**, so I am guessing there is some kind of backend call happening that informs Azure AD about the successful authentication, and the flowToken is what identifies our browser session.

Using PowerShell we can test the

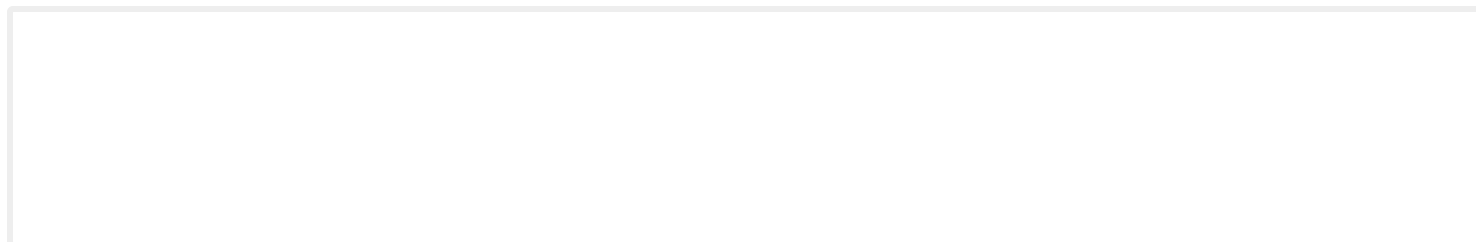
<https://login.microsoftonline.com/common/GetCredentialType?mkt=en-US> endpoint:

```
1 $url = "https://login.microsoftonline.com/common/GetCredentialType?mkt=en-US"
2 $body = @{
3     username = "alland@m365x912454.onmicrosoft.com"
4     flowtoken = "marius"
5 } | ConvertTo-Json
6
```

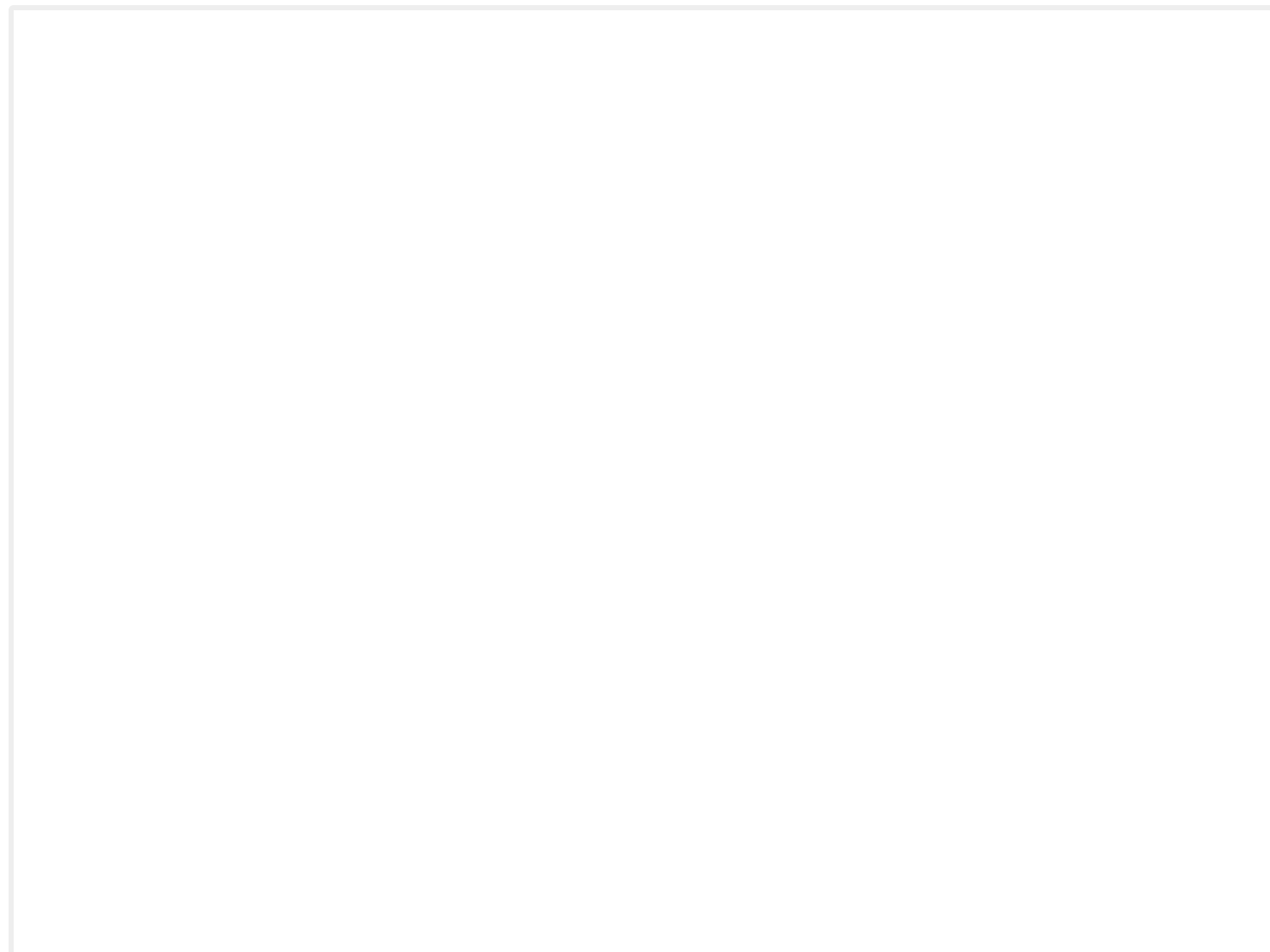
 Comment  Reblog  Subscribe ...

```
7 | $result = Invoke-RestMethod -Method Post -Uri $url -Body $body -ContentT  
8 | "FlowToken: $($result.FlowToken)"  
9 | "CertAuthUrl: $($result.Credentials.CertAuthParams.CertAuthUrl)"
```

Using this test I can find that the flowtoken returned is simply whatever it sent *to* the endpoint:



But where does the FlowToken stem from? Well, I found that it comes from a subset of the **buid** cookie returned from the authorize endpoint:



So I guess it would be technically possible to trigger the authorize endpoint from PowerShell, reading the **buid** cookie, like this:

```
1 $userPrincipalName = "AllanD@M365x912454.OnMicrosoft.com"
2 $CertificateThumbprint = "456b0789b3b4f0f9980ea10c3472a6c3e97419d0"
3
4 # Extract flowtoken
5 $r = Invoke-WebRequest -Uri "https://login.microsoftonline.com/organiza
6 $cookie = $r.RawContent -split "`n" | Where-Object {$_.like "set-Cooki
7 $flowtoken = ($cookie -replace "set-Cookie: " -split ";" | Where-object
8
9 # Get credential type
10 $url = "https://login.microsoftonline.com/common/GetCredentialType?mkt=
11 $body = @{
```

 Comment  Reblog  Subscribe 

```
12     username = $userPrincipalName
13     flowtoken = $flowtoken
14 } | ConvertTo-Json
15
16 $result = Invoke-RestMethod -Method Post -Uri $url -Body $body -Content
17 "FlowToken: $($result.FlowToken)"
18 "CertAuthUrl: $($result.Credentials.CertAuthParams.CertAuthUrl)"
19
20 # Auth using cert
21 $body = "flowToken=$($result.FlowToken)"
22 $certauth = Invoke-RestMethod $result.Credentials.CertAuthParams.CertAu
```

But for some reason I just cannot get this to work, as it errors out with **AADSTS9002313: Invalid request. Request is malformed or invalid.** But anyway, this feature is not really targeted towards script scenarios.

That's it, the feature is really well documented and works great, and it seems really simple from the customer side. One thing to really note here, is that one should definitely monitor the trusted root authorities of Azure AD, as they can be used to add backdoors to your system.

Have a good one!

Share this:



Loading...

Related



Comment



Reblog



Subscribe



Authenticating to Azure AD as an application using certificate based client credential grant

July 7, 2020
Liked by 1 person

Automating certificate rollover for Azure AD applications using Azure Functions and KeyVault

August 15, 2021
Liked by 2 people

Azure DevOps ARM Service Connection using certificate

February 19, 2023

Published by Marius Solbakken

[View all posts by Marius Solbakken](#)

Published

February 15, 2022

< Finding Azure AD domain from tenant id

Terraform module for automatically maintaining Azure DevOps variable group with app secret >

One thought on “Digging into Azure AD Certificate-Based Authentication”

Xiaoyue D (@p_orange_kitty)

December 15, 2022 at 3:29 pm

Your explanation is much more clear than Azure official doc! I followed your step and successfully setup a local signed cert to login.

 Comment

 Reblog

 Subscribe



There are two minor changes during my setup are:

1. When creating CSR by command ``openssl req -new -sha256 -newkey rsa:4096 -nodes -keyout "$userPrincipalName-key.pem" -out "$userPrincipalName-req.pem" -subj "/C=NO/ST=Oslo/L=Oslo/O=Good Workaround/OU=IT/CN=$userPrincipalName" -addext "subjectAltName=otherName:1.3.6.1.4.1.311.20.2.3;UTF8:$userPrincipalName"``, the CN= in -sub should be same as name of root CA.

2. When I using the command ``.$openssl ca -md sha256 -config ca.conf -out "$userPrincipalName-certificate.pem.crt" -infile "$userPrincipalName-req.pem"`` to sign the cert, the SAN field disappeared. So I used a workaround with an extension files which explicitly defines a SAN.

– command: `openssl x509 -req -days 365 -in "$userPrincipalName.csr" -CA ./ca/ca.crt -CAkey ./ca/ca.key -CAcreateserial -out "$userPrincipalName.crt" -extfile '$userPrincipalName.conf' -extensions v3_req`

– Part of \$userPrincipalName.conf file:

“`

...

[v3_req]

keyUsage = keyEncipherment, dataEncipherment, digitalSignature

extendedKeyUsage = serverAuth, clientAuth

subjectAltName =

otherName:msUPN;UTF8:firsthonoreduser@xiaoyueduangmail.onmicrosoft.com

“`

Besides, it should be carefully that the cert ``keyUsage`` must contain ``digitalSignature``.

Otherwise when logging in, web browser will not show up the option of the cert.

[↩ Reply](#)

Leave a comment

[Comment](#) [↕ Reblog](#) [📄 Subscribe](#) [...](#)

Marius Solbakken

Co-founder and Principal Cloud Engineer @ [Fortytwo](#), Microsoft Security MVP, Host of the CloudFirst



Podcast

 Comment

 Reblog

 Subscribe





Tags

[.NET](#) [Active Directory](#) [ADAL](#) [ADFS](#) [API authentication](#) [Azure](#) [Azure AD](#) [C#](#) [cloud](#) [entra-id](#) [Exchange](#)


[Exchange Online](#) [FIM](#) [Full IGA using Azure AD](#) [microsoft](#) [microsoft-graph](#) [microsoft-graph-sdk](#) [Office 365](#)


[PowerShell](#) [radius](#) [Reporting](#) [Scripting](#) [Security](#) [SharePoint 2013](#) [Single Sign-On](#) [SSO](#) [Time saving Tools](#)


 [Comment](#)  [Reblog](#)  [Subscribe](#) 

My Tweets

[Blog at WordPress.com.](#)

 Comment

 Reblog

 Subscribe

