EV code signing with .pfx in 202426/10/2024

ferib.dev/blog/EV-code-signing-with-pfx-in-2024

I have wanted to get my hands on an EV Code Sign for some time now. Being able to sign my own Windows drivers and running them on production Windows systems has been amazing. However, I would like to do so without having to worry about my kernel exploits being fixed, leaked certs being revoked or exploitable drivers being blacklisted.

For one to sign his own Microsoft Windows Drivers, he will need an EV Code Sign certificate. As AFAIK all these Extended Validation (EV) ones will require some sort of vetting and will only be handed out to actual organizations, and not persons.

Setting up a shell company

Sure enough, setting up a small company isn't that big of a deal. It only costs a few thousand for initial investment capital, which can be immediatly used for notary and legal fees.

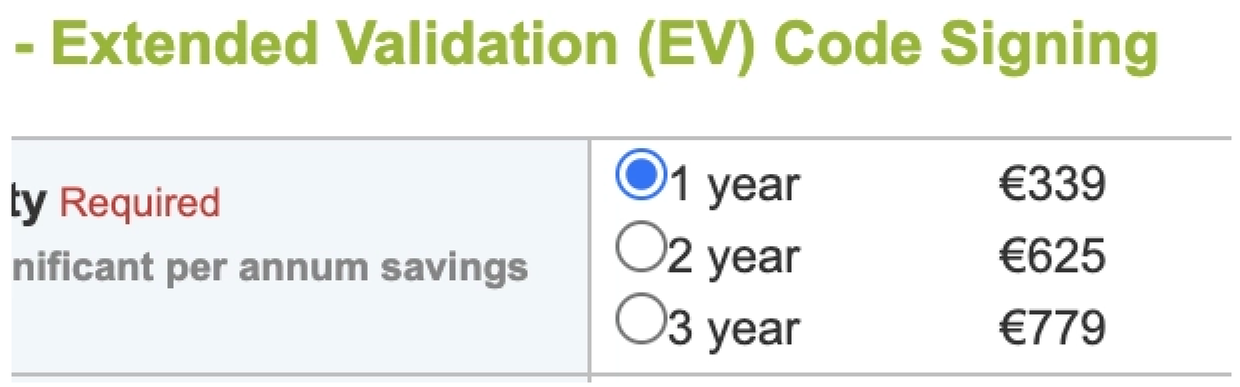
Once such a company is set up, it is time to shop for an EV Code Sign cert.

Signing up for an EV Code Sign cert

As I was reading some of the documentation on the Microsoft Developer Network (MSDN) website, I saw a list of recommended vendors to get an EV Code Sign cert from.

I have no experience with code signing certs in general, but to me, it looked like they were all selling the same type of baloney. By just looking at the Google search results, I can see how crowded the results are with corporate blog posts and articles, all trying to sell you the same product.

Anyway, I am looking for *just* the EV Code Sign cert, I have no interest in any special features or whatever. So I decided to go with the cheapest possible recommendation from the list, GlobalSign. Just looking at their website, I can see they cut a lot of corners as I navigate the 2010-style web panels and make my way to the checkout.



- Extended Validation (EV) Code Signing

ty Required	[✓ 1 year]	€339
nificant per annum savings	2 year	€625
	3 year	€779

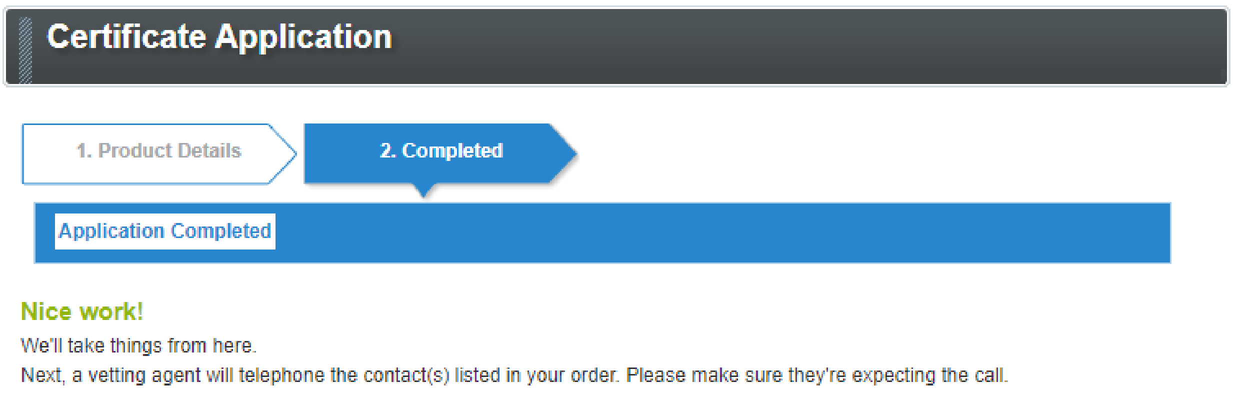
I heard from other people there are cheaper options which are in the 100 ~ 150 price range. The more expansive ones are somewhere 500+ a year.

Looking back at this, I wish I took the cheapest option possible, regardless of what's on the Microsoft recommendation list. I highly doubt I got any value for paying so much extra.

The vetting process

Okay, I had to do this weird sales thing where I cannot buy the product directly, but instead, a special account is created for me as I wait for a 'salesperson' to reach out to me over email.

At this point, I just forgot _(or couldn't be bothered)_ to complete my purchase. About 4 months later I wanted to finish what I had started, but obviously, I forgot my password. Thanks to this 2010-styled website, automated password recovery doesn't seem to be a thing. A few days later everything worked out, and I finally paid the 339 EURO to complete the process.



Certificate Application

1. Product Details

2. Completed

Application Completed

Nice work!

We'll take things from here.

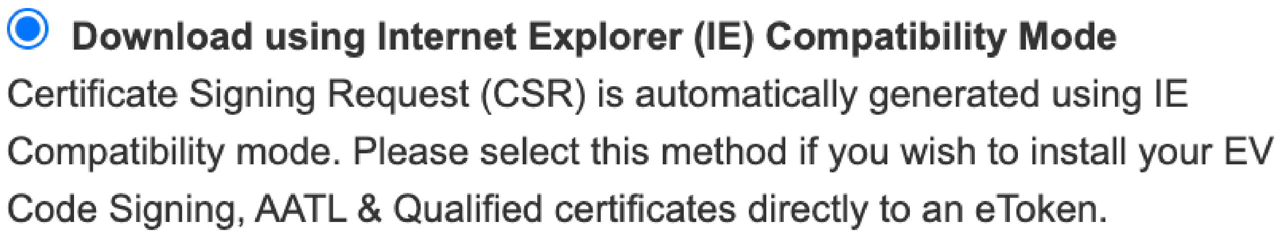
Next, a vetting agent will telephone the contact(s) listed in your order. Please make sure they're expecting the call.

Next, I receive a new email from another company regarding my order. They requested some documents as part of the ID verification, I supplied those and was told to wait for further instructions. They then informed me they would call me in the next couple of days. So about a week or so later, I got a call from some non-native-speaking guy from, a crowded call center -- judging by the background noise -- asking the same question like five times as I had such a hard time making sense of what he was asking. I think they tried to pronounce my full name, to which I said "yes yes" and the call ended shortly after.

Cheers, 2 days later, a GlobalSign email arrives informing me that my certificate download is ready!

Obtaining the certificate

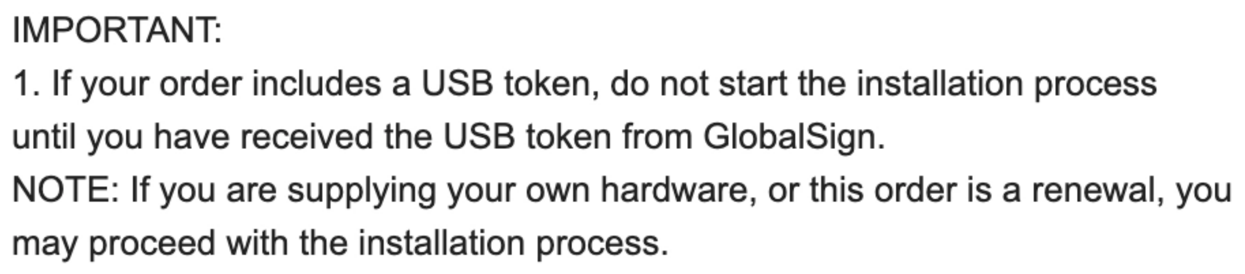
Okay good, now all that's left to do is download a `.pfx` file and start signing?



● **Download using Internet Explorer (IE) Compatibility Mode**

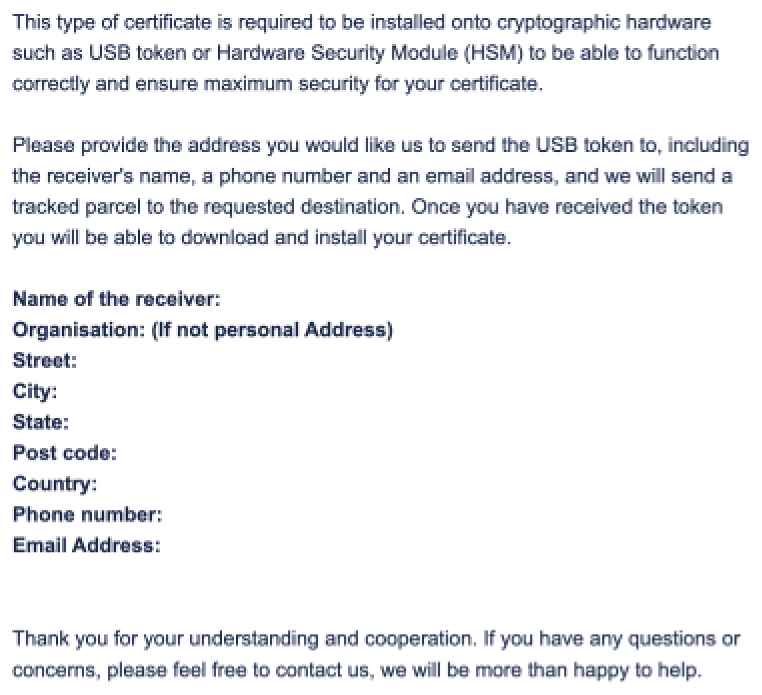
Certificate Signing Request (CSR) is automatically generated using IE Compatibility mode. Please select this method if you wish to install your EV Code Signing, AATL & Qualified certificates directly to an eToken.

Sure, let me google how to enable Internet Explorer Compatibility Mode in 2024, this shouldn't be hard.


IMPORTANT:
1. If your order includes a USB token, do not start the installation process until you have received the USB token from GlobalSign.
NOTE: If you are supplying your own hardware, or this order is a renewal, you may proceed with the installation process.

USB Token? Eh, no?

This reminds me... after I completed the vetting, this Indian call center guy sent me an email requesting the following information:


This type of certificate is required to be installed onto cryptographic hardware such as USB token or Hardware Security Module (HSM) to be able to function correctly and ensure maximum security for your certificate.

Please provide the address you would like us to send the USB token to, including the receiver's name, a phone number and an email address, and we will send a tracked parcel to the requested destination. Once you have received the token you will be able to download and install your certificate.

Name of the receiver:
Organisation: (If not personal Address)
Street:
City:
State:
Post code:
Country:
Phone number:
Email Address:

Thank you for your understanding and cooperation. If you have any questions or concerns, please feel free to contact us, we will be more than happy to help.

At this point in time, I assumed it was some malicious sales practice where they happily send you an eToken if you didn't have one already, just to send you an absurd invoice afterwards. So I refused to get a stupid enterprise eToken from a shady Indian guy...

Spoiler alert, I was wrong... the eToken was included in the price. I reached out to my 'salesperson' guy over email, and he confirmed this eToken was included, even the shipping was included!

Obtainig my eToken

Okay, let me reply to that one email with the requested information so I can get my eToken. Just a few more weeks of waiting, and it arrived at my door.

Along with the eTokens comes of course the bloatware. I installed this SafeNet Authentication Client aka SACTools as I set up the eToken, but I could immediatly see my system being bloated with whatever corporate crap that is.

A new process named 'SACMonitor.exe' is running in the background, new tray icons are showing up, and new startup processes `SACMonitor.exe` was added so it runs whenever I reboot my device...

Shame! Closed-source crapware bloating my device is a no-go. I quickly uninstalled the tools and started looking for an alternative.

Buying a better USB Token

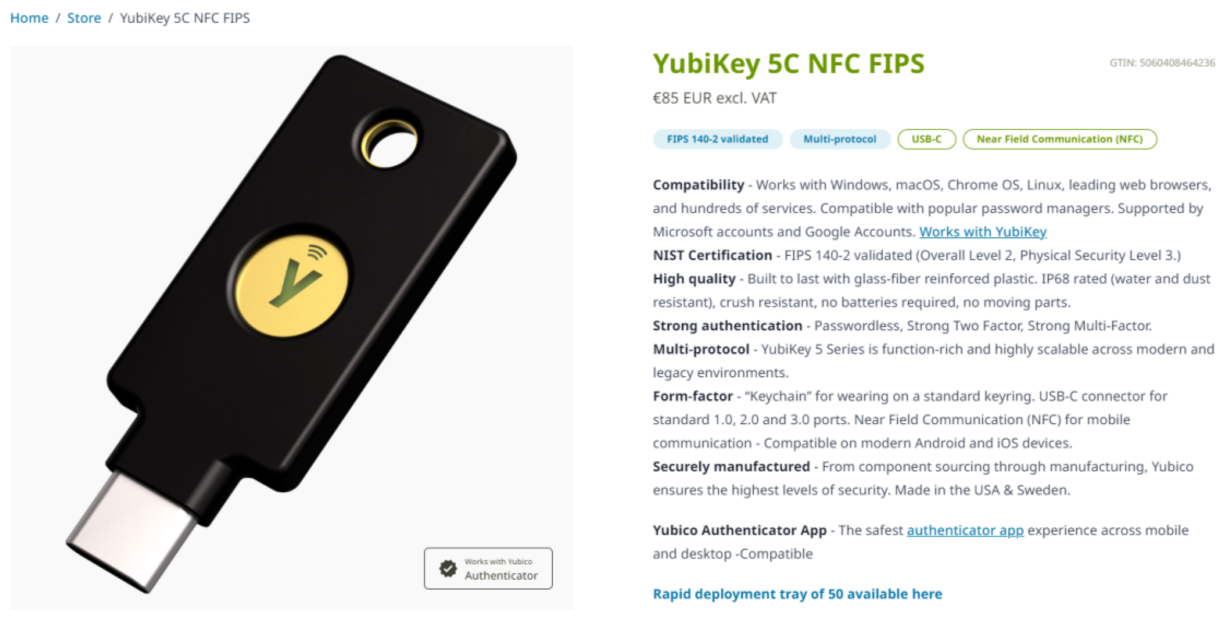
Looking back at the certificate pickup website, I was prompted with 2 options. I either use the Fortify app or Internet Explorer (compatible) browser. I decided that I already had enough bloatware on my Windows machine, so installing enterprise apps like this was a no-go.

Anyway, what I wanted to show was this little checkbox on the bottom. It forced me to check it, which meant I was forced to use a FIPS140-2 compatible device.

![fortify](Fortify_browser_pickup.webp"/>

Shopping for a FIPS-compatible Yubikey

Browsing the Yubcio I noticed the following product matching what I need. It even has NFC support, how neat, can sign my drivers without even plugging it in? 🤣


Home / Store / YubiKey 5C NFC FIPS

YubiKey 5C NFC FIPS GTIN: 5060408464236

€85 EUR excl. VAT

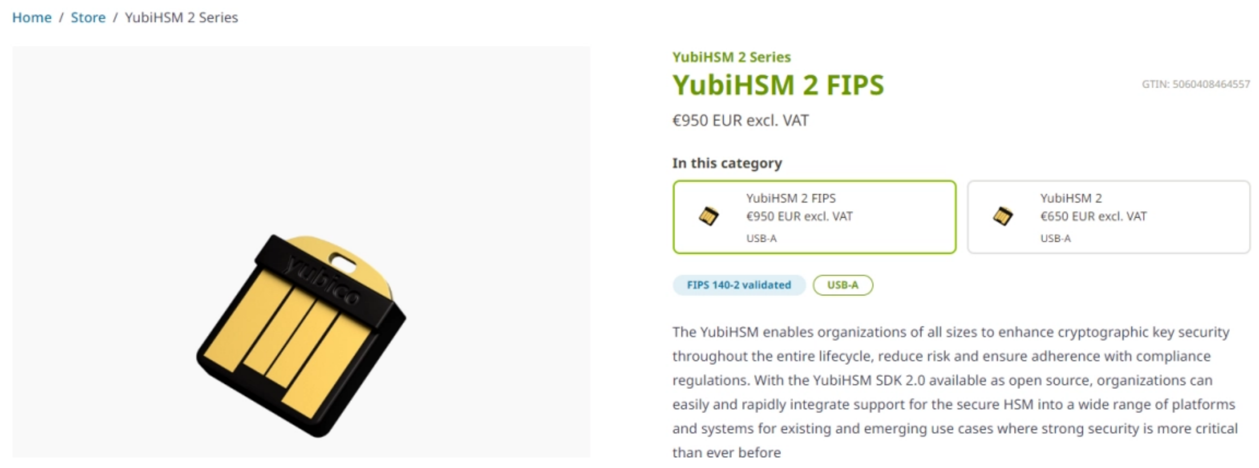
FIPS 140-2 validated · Multi-protocol · USB-C · Near Field Communication (NFC)

Compatibility - Works with Windows, macOS, Chrome OS, Linux, leading web browsers, and hundreds of services. Compatible with popular password managers. Supported by Microsoft accounts and Google Accounts. Works with YubiKey
NIST Certification - FIPS 140-2 validated (Overall Level 2, Physical Security Level 3.)
High quality - Built to last with glass-fiber reinforced plastic. IP68 rated (water and dust resistant), crush resistant, no batteries required, no moving parts.
Strong authentication - Passwordless, Strong Two Factor, Strong Multi-Factor.
Multi-protocol - YubiKey 5 Series is function-rich and highly scalable across modern and legacy environments.
Form-factor - "Keychain" for wearing on a standard keyring. USB-C connector for standard 1.0, 2.0 and 3.0 ports. Near Field Communication (NFC) for mobile communication - Compatible on modern Android and iOS devices.
Securely manufactured - From component sourcing through manufacturing, Yubico ensures the highest levels of security. Made in the USA & Sweden.

Yubico Authenticator App - The safest authenticator app experience across mobile and desktop -Compatible

Rapid deployment tray of 50 available here

I also found this other product, which looks pretty interesting, but the price point is a bit far away from my budget.


Home / Store / YubiHSM 2 Series

YubiHSM 2 Series

YubiHSM 2 FIPS

GTIN: 5060408464557

€950 EUR excl. VAT

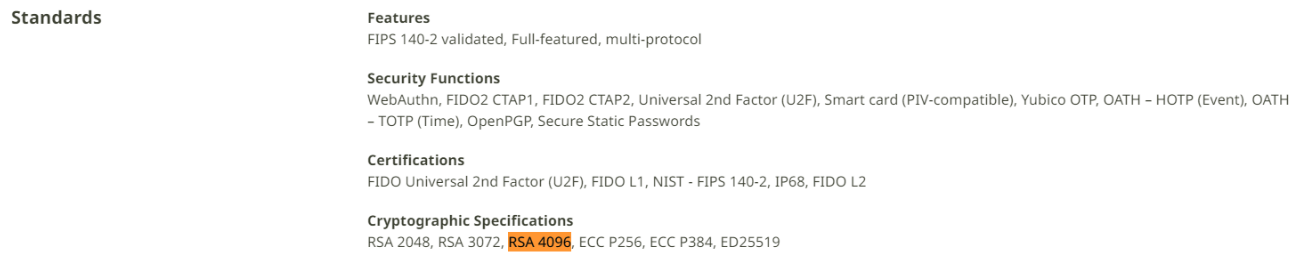
In this category

YubiHSM 2 FIPS €950 EUR excl. VAT USB-A	YubiHSM 2 €650 EUR excl. VAT USB-A

The YubiHSM enables organizations of all sizes to enhance cryptographic key security throughout the entire lifecycle, reduce risk and ensure adherence with compliance regulations. With the YubiHSM SDK 2.0 available as open source, organizations can easily and rapidly integrate support for the secure HSM into a wide range of platforms and systems for existing and emerging use cases where strong security is more critical than ever before

Anyway, before I buy the 85 EURO YubiKey I better make sure that these devices can be used for code signing as I have only seen people use them for 2FA and such.

I was also informed that RSA 4096-bit keys must be used for the EV Code Sign certs, so I also made sure this was a thing by checking the specifications of the device.


Standards

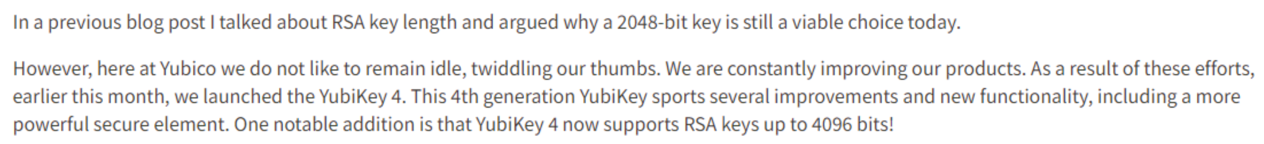
Features
FIPS 140-2 validated, Full-featured, multi-protocol

Security Functions
WebAuthn, FIDO2 CTAP1, FIDO2 CTAP2, Universal 2nd Factor (U2F), Smart card (PIV-compatible), Yubico OTP, OATH – HOTP (Event), OATH – TOTP (Time), OpenPGP, Secure Static Passwords

Certifications
FIDO Universal 2nd Factor (U2F), FIDO L1, NIST - FIPS 140-2, IP68, FIDO L2

Cryptographic Specifications
RSA 2048, RSA 3072, RSA 4096, ECC P256, ECC P384, ED25519

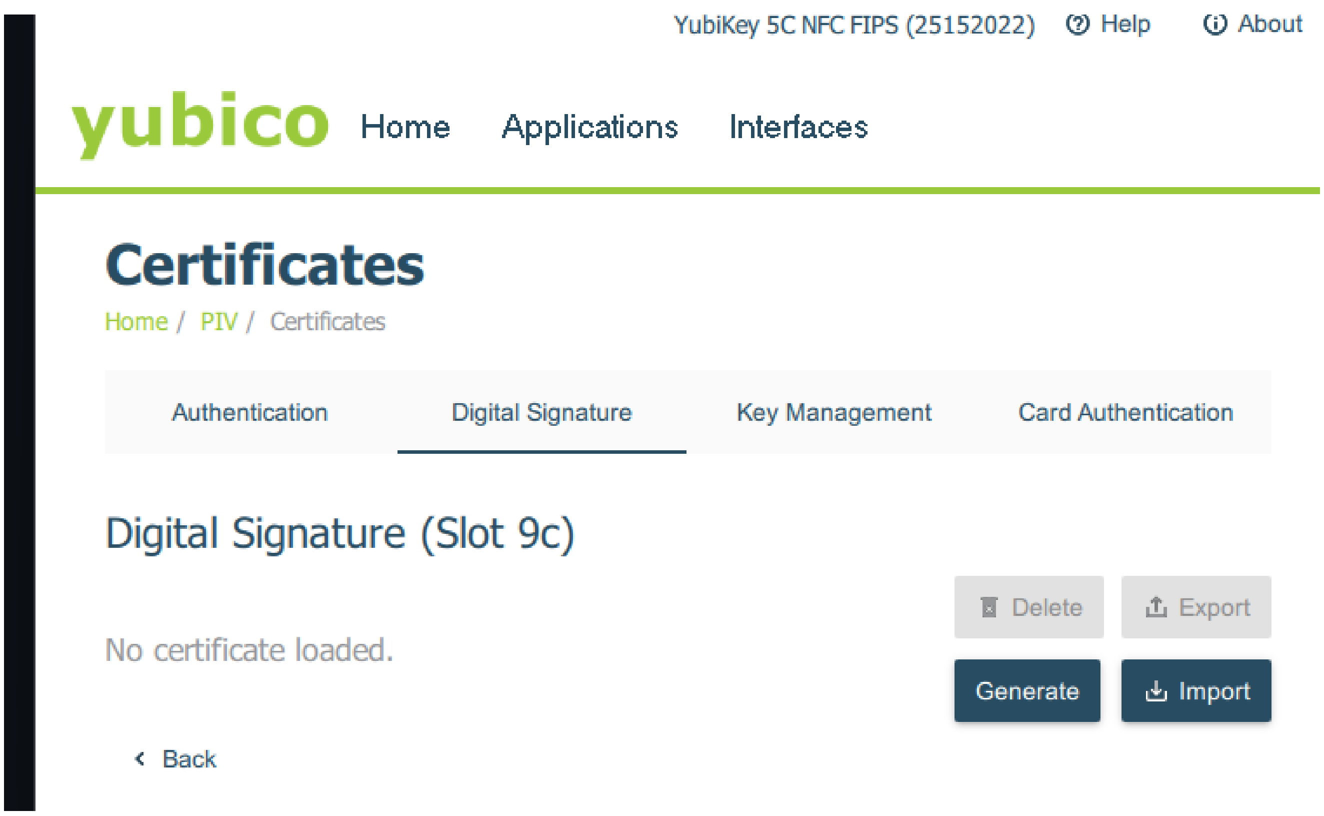
As well as looking at the community to see if this was a valid solution.


In a previous blog post I talked about RSA key length and argued why a 2048-bit key is still a viable choice today.

However, here at Yubico we do not like to remain idle, twiddling our thumbs. We are constantly improving our products. As a result of these efforts, earlier this month, we launched the YubiKey 4. This 4th generation YubiKey sports several improvements and new functionality, including a more powerful secure element. One notable addition is that YubiKey 4 now supports RSA keys up to 4096 bits!

Setting up the YubiKey

After confidently completing the purchase, I waited about a week for it to arrive. Once it arrived, I couldn't wait to set it up and install the fancy GUI it came with


YubiKey 5C NFC FIPS (25152022) Help About

yubico Home Applications Interfaces

Certificates

Home / PIV / Certificates

Authentication | Digital Signature | Key Management | Card Authentication

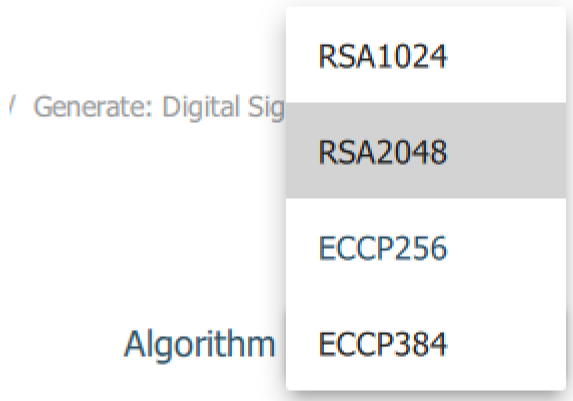
Digital Signature (Slot 9c)

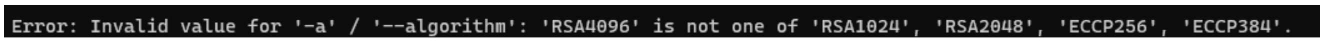
No certificate loaded.

It took me a couple of minutes, just to get disapointed as I tried to generate my first 'Digital Signature' keys.

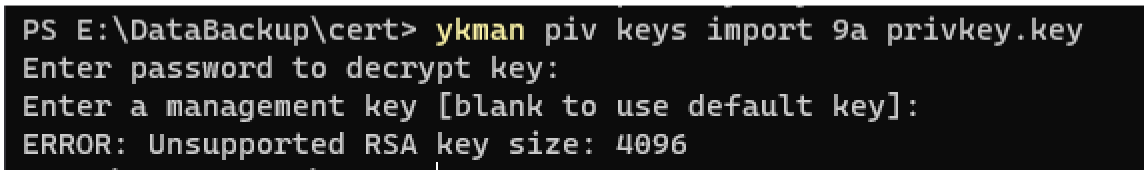
The limited selection of the algorithms doesn't include `RSA4096`, it seems that `RSA2048` is the highest option available.

So I ended up doing some research and found the CLI tool for YubiKeys. I set up this `ykman` tool and used the CLI to generate a key with algorithm `RSA4096`, but I am once again greeted with that same list from the GUI, which is missing the option I want.




```
Error: Invalid value for '-a' / '--algorithm': 'RSA4096' is not one of 'RSA1024', 'RSA2048', 'ECCP256', 'ECCP384'.
```

In my last attempt, I generated a 4096-bit RSA key myself and figured I might be able to import it into Yubikey using the CLI. But no, I once again faced disappointment.


```
PS E:\DataBackup\cert> ykman piv keys import 9a privkey.key
Enter password to decrypt key:
Enter a management key [blank to use default key]:
ERROR: Unsupported RSA key size: 4096
```

Back to the eToken

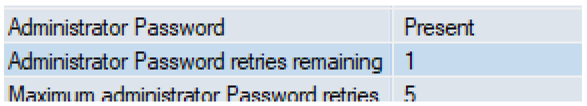
Okay, it's been a few weeks now... what was the password again for my eToken? Ah yes, I wrote it down, did I?

Enters PIN wrong 3 times

Shit, did I fatfinger the pin like 3 times in a row? Oh well, let me grab that admin password.

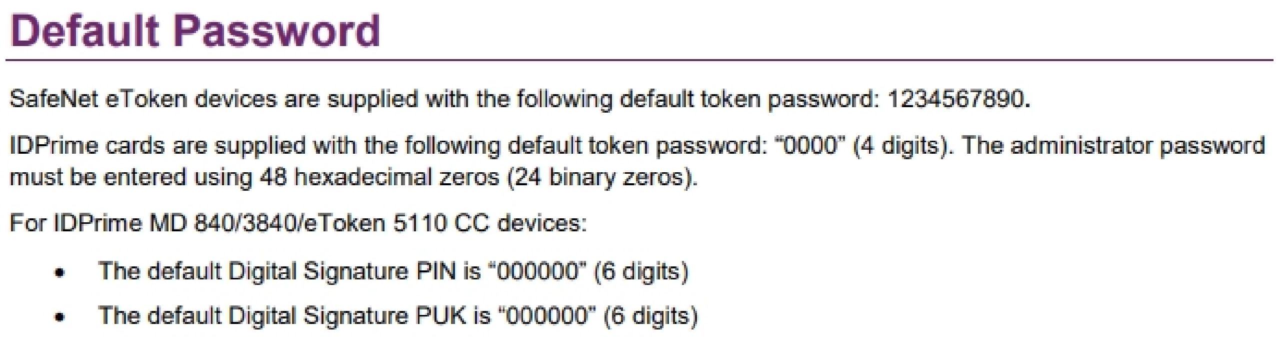
Enters Admin Password wrong 4 times

Oh shit, that pin and password I wrote down? Yeah, that was for my Yubikey, NOT my crappy eToken... But wait, come to think of it, I never set a password for the eToken!



Administrator Password	Present
Administrator Password retries remaining	1
Maximum administrator Password retries	5

Okay, then it must still be using the default password? Sure, let me look up the manual for that eToken.



Default Password

SafeNet eToken devices are supplied with the following default token password: 1234567890.

IDPrime cards are supplied with the following default token password: "0000" (4 digits). The administrator password must be entered using 48 hexadecimal zeros (24 binary zeros).

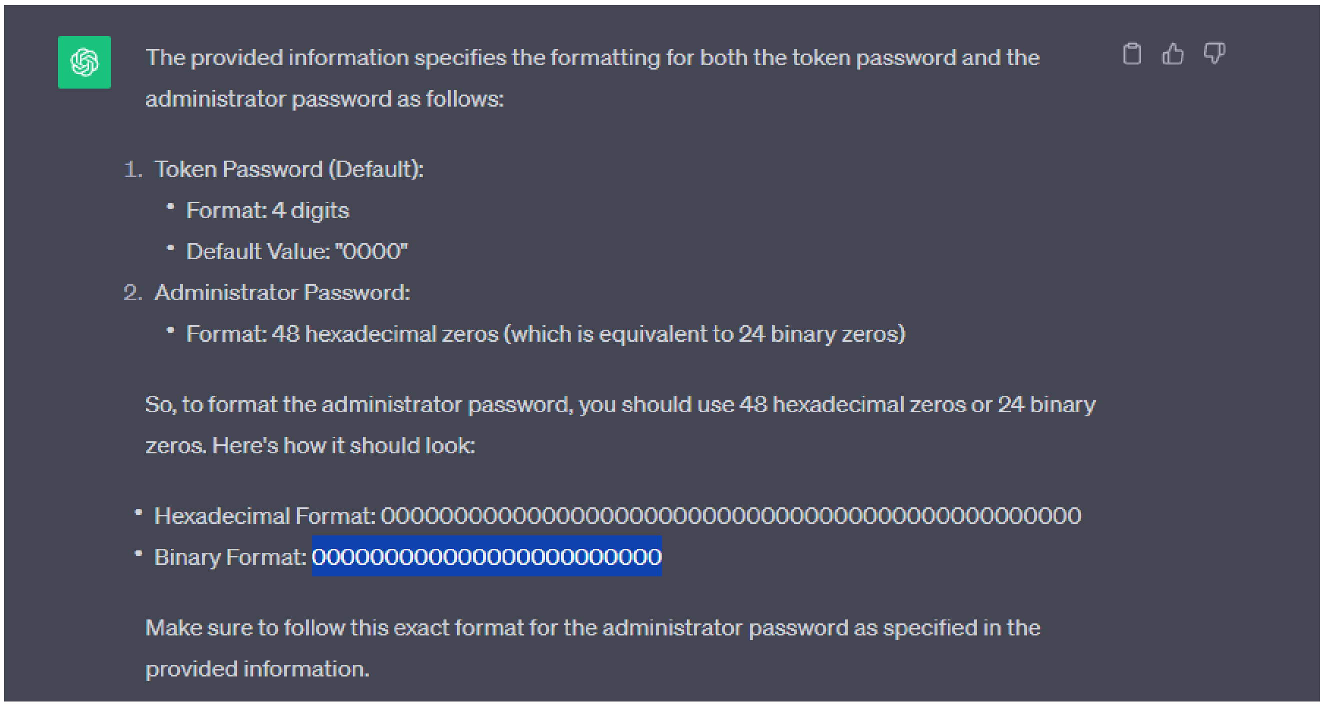
For IDPrime MD 840/3840/eToken 5110 CC devices:

- The default Digital Signature PIN is "000000" (6 digits)
- The default Digital Signature PUK is "000000" (6 digits)

Lovely, the default *"administrator password must be entered using 48 hexadecimal zeros (24 binary zeros)."*. Wait what? I have exactly 1 attempt left, how do I enter 48 hexadecimal zeros?

Do I format 48 zeros with a whitespace, like this `00 00 00 ...`? or maybe `0 0 0 ...`? or should I prefix em `0x00 0x00 0x00 ...`? or do I drop whitespaces like so `000...` or do I need to enter a double 00's for every 0 like `000000...`?

I am baffled, so I asked ChatGPT's opinion.



The provided information specifies the formatting for both the token password and the administrator password as follows:

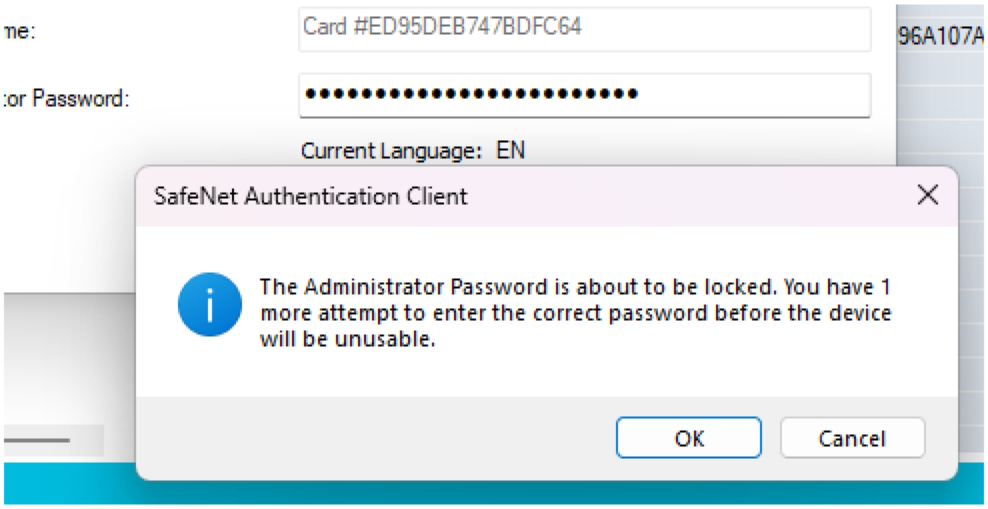
1. Token Password (Default):
   - Format: 4 digits
   - Default Value: "0000"
2. Administrator Password:
   - Format: 48 hexadecimal zeros (which is equivalent to 24 binary zeros)

So, to format the administrator password, you should use 48 hexadecimal zeros or 24 binary zeros. Here's how it should look:

- Hexadecimal Format: 000000000000000000000000000000000000000000000000
- Binary Format: 000000000000000000000000

Make sure to follow this exact format for the administrator password as specified in the provided information.

Okay, let me just blatantly copy-paste the zeros from the 'Hexadecimal Format', paste them at the login prompt, and hope for the best?



ne: Card #ED95DEB747BDFC64

or Password: ••••••••••••••••••••••••

Current Language: EN

SafeNet Authentication Client

The Administrator Password is about to be locked. You have 1 more attempt to enter the correct password before the device will be unusable.

Well... it didn't work and my eToken is now locked 😅

Unlocking my eToken

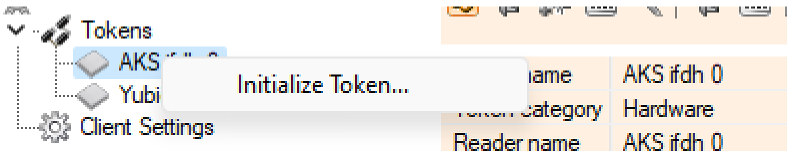
Okay... time to email support and ask how I can reset the eToken?

After a few days, I get a phone call -- in the middle of the day during actual work -- and I'm asked to join another call on MS Teams as I'm asked to share my screen. I tell the woman on the phone I have no headphones or mic available on my device, and ask her to keep talking over the phone. I then join the MS Teams calls, and she ends the phone call...

After a good 10 minutes of looking for headphones and an external microphone around the house, I managed to talk to the support woman using my desktop device.

The Support Call

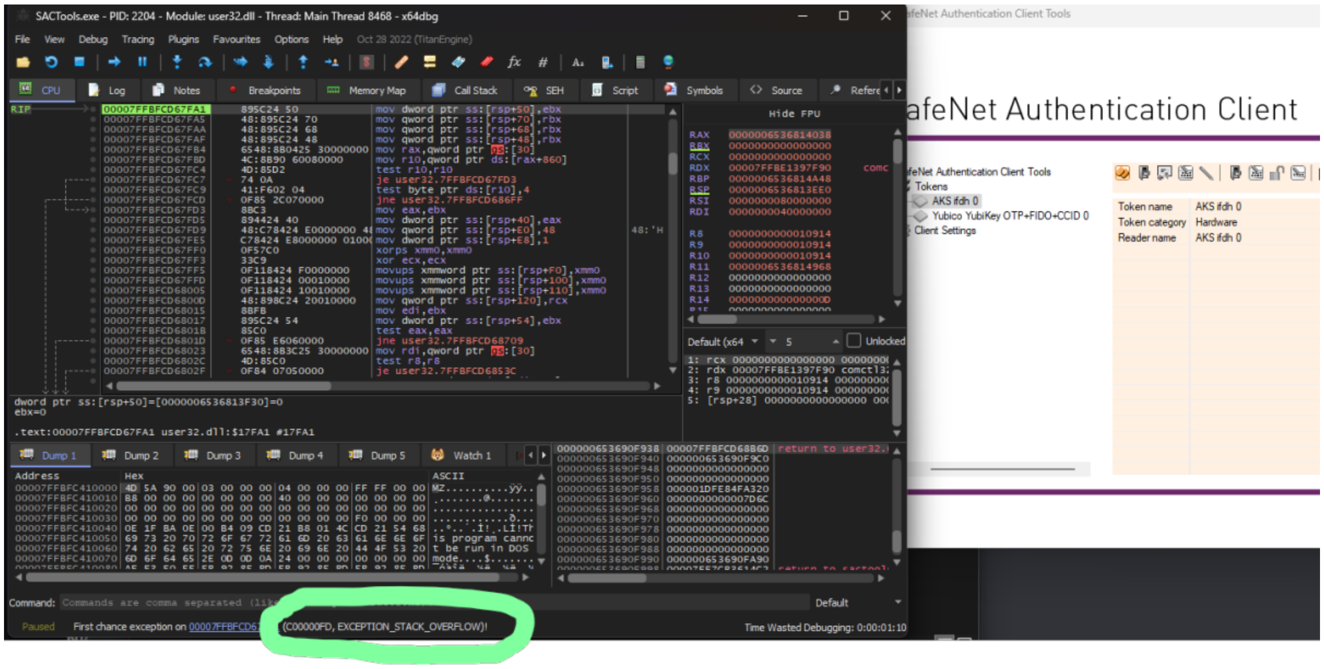
Long story short, I was told to navigate the bloatware and do "Initialize Token...", but this option was disabled in the GUI as my eToken was already initialized. She told me to downgrade my SACTools, restart the device, and try again.



Oh look, the "Initialize Token..." became visible after removing 10.8 and installing version 10.7. But once I click this, the application just closes and nothing happened...

After like half an hour of support with screen share, she suggested that I should buy a new eToken as she concluded there was no way possible to recover this.

Sure, I said I'll talk this over with my financial department -- trying not to laugh out loud as I tell her -- and exit the Teams call. Out of curiosity I opened x64dbg and attached the debugger to the SACTools.exe to see why it is closing. After clicking the "Initialize Token..." I see x64dbg caught an exception.



```
RIP  00007FFBFCD67FA1  895C24 50                 mov dword ptr ss:[rsp+50],ebx
     00007FFBFCD67FA5  48:895C24 70              mov qword ptr ss:[rsp+70],rbx
     00007FFBFCD67FAA  48:895C24 68              mov qword ptr ss:[rsp+68],rbx
     00007FFBFCD67FAF  48:895C24 48              mov qword ptr ss:[rsp+48],rbx
     00007FFBFCD67FB4  6548:8B0425 30000000      mov rax,qword ptr gs:[30]
     00007FFBFCD67FBD  4C:8B90 60080000          mov r10,qword ptr ds:[rax+860]
     00007FFBFCD67FC4  4D:85D2                   test r10,r10
     00007FFBFCD67FC7  74 0A                     je user32.7FFBFCD67FD3
     00007FFBFCD67FC9  41:F602 04                test byte ptr ds:[r10],4
     00007FFBFCD67FCD  0F85 2C070000             jne user32.7FFBFCD686FF
     00007FFBFCD67FD3  8BC3                      mov eax,ebx
     00007FFBFCD67FD5  894424 40                 mov dword ptr ss:[rsp+40],eax
     00007FFBFCD67FD9  48:C78424 E0000000 4      mov qword ptr ss:[rsp+E0],48        48:'H
     00007FFBFCD67FE5  C78424 E8000000 0100      mov dword ptr ss:[rsp+E8],1
     00007FFBFCD67FF0  0F57C0                    xorps xmm0,xmm0
     00007FFBFCD67FF3  33C9                      xor ecx,ecx
     00007FFBFCD67FF5  0F118424 F0000000         movups xmmword ptr ss:[rsp+F0],xmm0
     00007FFBFCD67FFD  0F118424 00010000         movups xmmword ptr ss:[rsp+100],xmm0
     00007FFBFCD68005  0F118424 10010000         movups xmmword ptr ss:[rsp+110],xmm0
     00007FFBFCD6800D  48:898C24 20010000        mov qword ptr ss:[rsp+120],rcx
     00007FFBFCD68015  8BFB                      mov edi,ebx
     00007FFBFCD68017  895C24 54                 mov dword ptr ss:[rsp+54],ebx
     00007FFBFCD6801B  85C0                      test eax,eax
     00007FFBFCD6801D  0F85 E6060000             jne user32.7FFBFCD68709
     00007FFBFCD68023  6548:8B3C25 30000000      mov rdi,qword ptr gs:[30]
     00007FFBFCD6802C  4D:85C0                   test r8,r8
     00007FFBFCD6802F  0F84 07050000             je user32.7FFBFCD6853C

dword ptr ss:[rsp+50]=[0000006536813F30]=0
ebx=0

.text:00007FFBFCD67FA1 user32.dll:$17FA1 #17FA1
```

```
RAX   0000006536814038
RBX   0000000000000000
RCX   0000000000000000
RDX   00007FFBE1397F90     comc
RBP   0000006536814A48
RSP   0000006536813EE0
RSI   0000000080000000
RDI   0000000040000000

R8    0000000000010914
R9    0000000000010914
R10   0000000000010914
R11   0000006536814968
R12   0000000000000000
R13   0000000000000000
R14   000000000000000D

1: rcx 0000000000000000 0000000
2: rdx 00007FFBE1397F90 comctl3
3: r8 0000000000010914 0000000
4: r9 0000000000010914 0000000
5: [rsp+28] 0000000000000000 000
```

```
Address           Hex                                              ASCII
00007FFBFC410000  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZ..........ÿÿ..
00007FFBFC410010  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ,.......@.......
00007FFBFC410020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00007FFBFC410030  00 00 00 00 00 00 00 00 00 00 00 00 F0 00 00 00  ............ð...
00007FFBFC410040  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ..º..´.Í!¸.LÍ!Th
00007FFBFC410050  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
00007FFBFC410060  74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS 
00007FFBFC410070  6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode....$.......
```

```
000000653690F938  00007FFBFCD68B6D  return to user32.
000000653690F940  000000653690F9C0
000000653690F948  0000000000000000
000000653690F950  0000000000000000
000000653690F958  000001DFE84FA320
000000653690F960  0000000000007D6C
000000653690F968  0000000000000000
000000653690F970  0000000000000000
000000653690F978  0000000000000000
000000653690F980  0000000000000000
000000653690F988  0000000000000000
000000653690F990  000000653690FA90
```

```
Command: Commands are comma separated (lik
Paused   First chance exception on 00007FFBFCD6 (C00000FD, EXCEPTION_STACK_OVERFLOW)!   Time Wasted Debugging: 0:00:01:10
```

And no surprise, that the application is having some sort of unrecoverable bug happening. Most likely it was never intended to click this option when the device is in this specific state, which is most likely why this action is disabled on the later version of 10.8.

The Reverse Engineering lifestyle

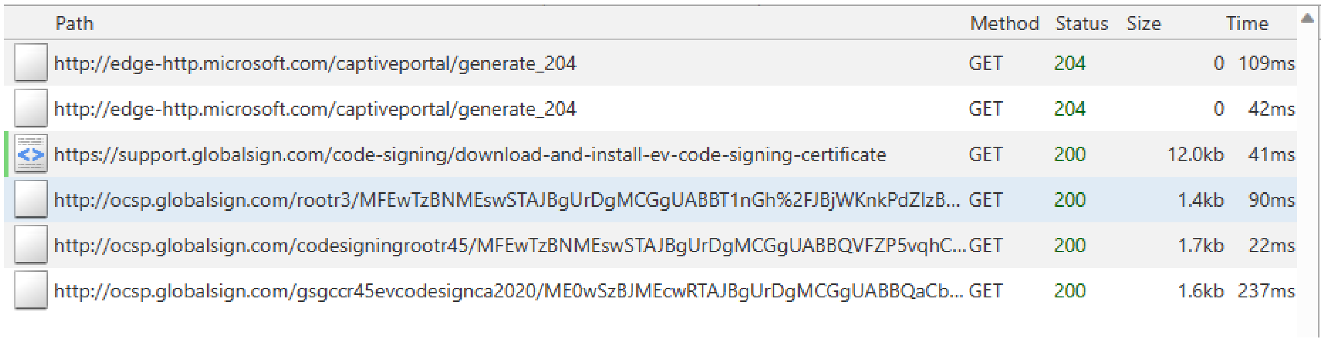
Okay, at this point, I am already poking at binaries and reversing them, so I might as well go down the rabbit hole and figure out exactly how this eToken was supposed to work.

With the loss of my eToken, I won't waste any more time on reversing the SAC tools/drivers/binaries and instead will focus on the certification pickup website. There must be some way to spoof or emulate the presence of the eToken, and send off my own DIY CSR to the remote server. I even doubt the eToken is performing cryptographic computations related to the key generation, and I assume most of it happens in either an Internet Explorer-compatible browser or whatever this 'Fortify app' is.

Reversing the Legacy IE JavaScript

Okay, I am running the Edge browser -- you know, that thing no one uses yet Microsoft is forcing it onto their users -- which, no surprise, is the only way to have Internet Explorer Legacy Compatibility mode.

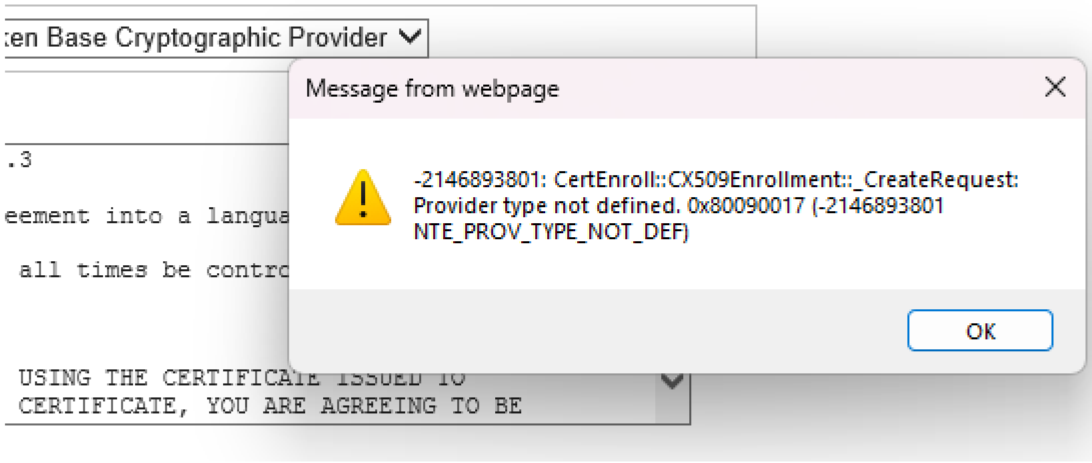
To make things worse, this thing doesn't have devtools like Element Inspection, Network Inspection, or a JS Console. To see the network traffic I would use `mitmproxy` and intercept HTML and JS files, as well as take note of requests being made.



Path	Method	Status	Size	Time
http://edge-http.microsoft.com/captiveportal/generate_204	GET	204	0	109ms
http://edge-http.microsoft.com/captiveportal/generate_204	GET	204	0	42ms
https://support.globalsign.com/code-signing/download-and-install-ev-code-signing-certificate	GET	200	12.0kb	41ms
http://ocsp.globalsign.com/rootr3/MFEwTzBNMEswSTAJBgUrDgMCGgUABBT1nGh%2FJBjWKnkPdZlzB...	GET	200	1.4kb	90ms
http://ocsp.globalsign.com/codesigningrootr45/MFEwTzBNMEswSTAJBgUrDgMCGgUABBQVFZP5vqhC...	GET	200	1.7kb	22ms
http://ocsp.globalsign.com/gsgccr45evcodesignca2020/ME0wSzBJMEcwRTAJBgUrDgMCGgUABBQaCb...	GET	200	1.6kb	237ms

I don't see too much going on here, this is most likely because of the old 2010-style website. The website is a good old html using `<form>` tags and `<input>` fields. They only seem to use JavaScript for very specific things like eToken and cryptographic (legacy) APIs.

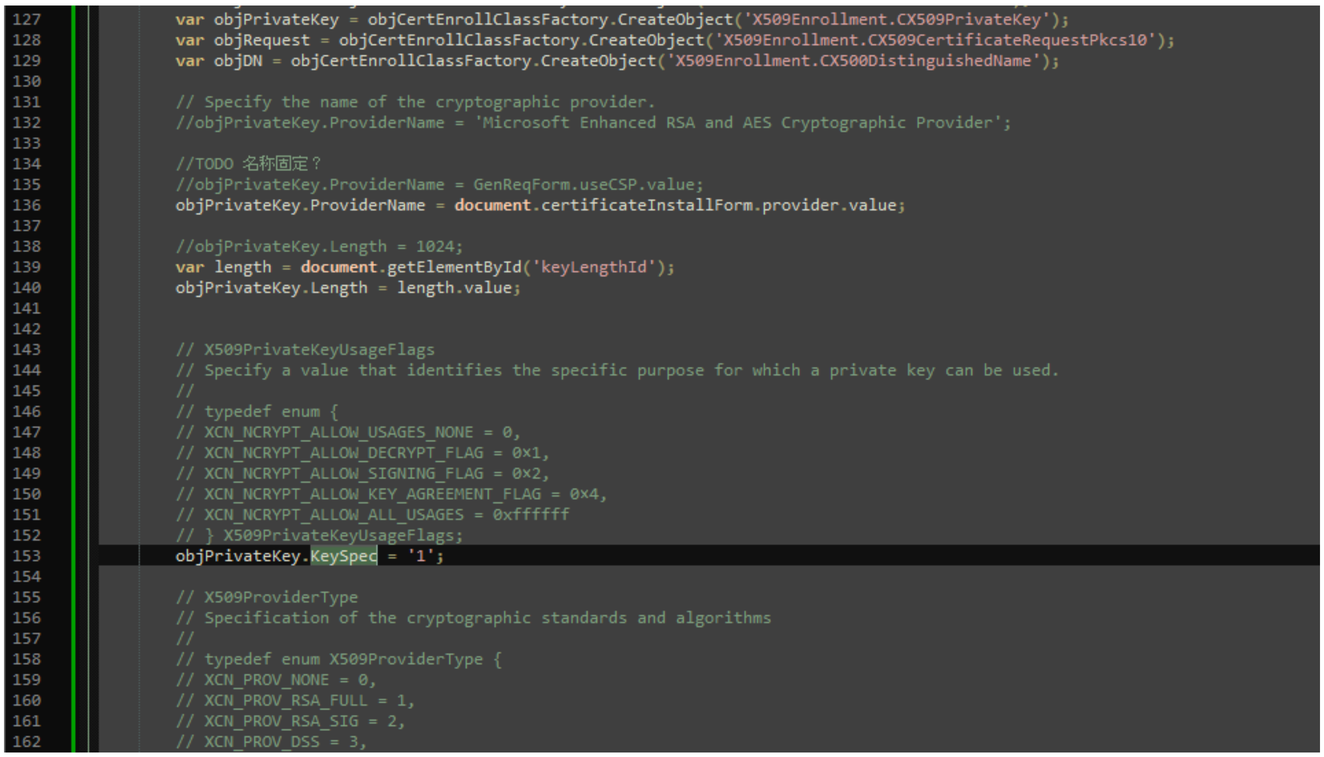
To learn a bit more about the expected workflow of the website, I use my 'broken' eToken. The website still picks it up -- that is, after installing some bloatware drivers -- and can be used to send valid requests to the server. None of these requests will complete -- and therefore won't consume my pickup cert -- as my 'broken' eToken is causing the below error:


ken Base Cryptographic Provider
.3
eement into a langua
all times be contro
USING THE CERTIFICATE ISSUED TO
CERTIFICATE, YOU ARE AGREEING TO BE
Message from webpage
OK

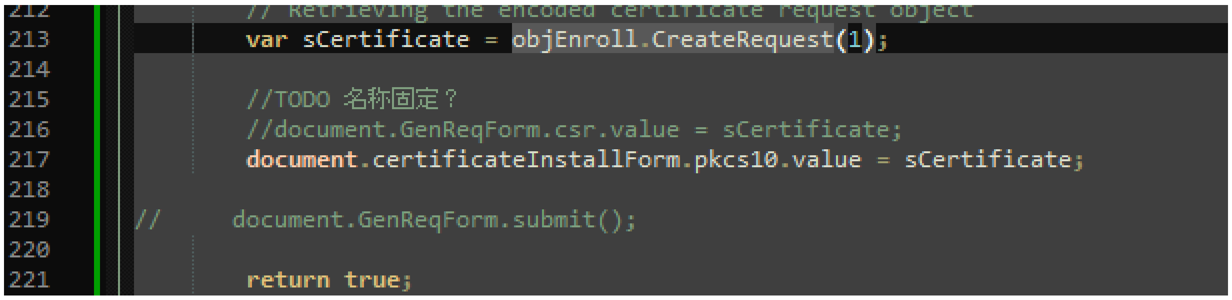
-2146893801: CertEnroll::CX509Enrollment::_CreateRequest: Provider type not defined. 0x80090017 (-2146893801 NTE_PROV_TYPE_NOT_DEF)

Extract source code

I look at the `mitmproxy` requests and dump all `.js` files to disk so I can properly look at them. The code below -- which has Chinese comments, lol -- seems to be invoking cryptographic operations to generate things.

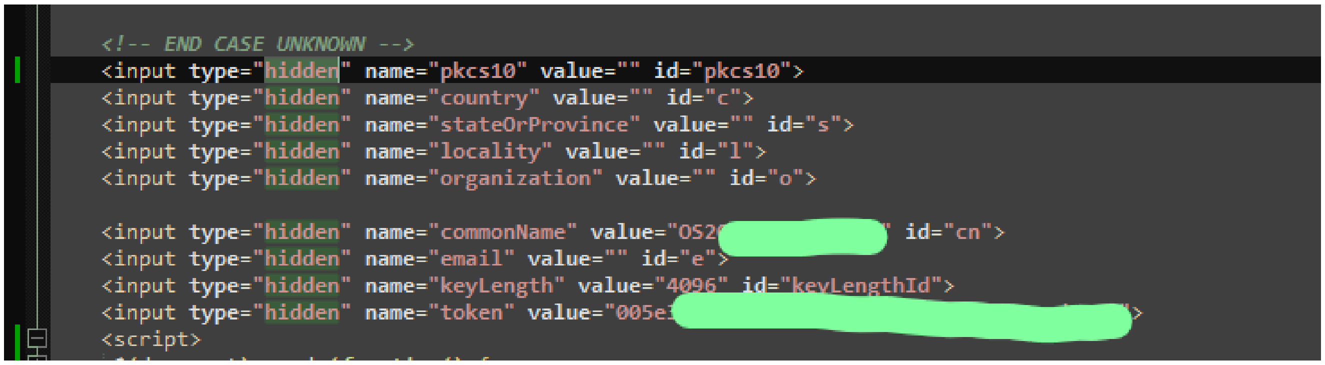


```
        var objPrivateKey = objCertEnrollClassFactory.CreateObject('X509Enrollment.CX509PrivateKey');
        var objRequest = objCertEnrollClassFactory.CreateObject('X509Enrollment.CX509CertificateRequestPkcs10');
        var objDN = objCertEnrollClassFactory.CreateObject('X509Enrollment.CX500DistinguishedName');

        // Specify the name of the cryptographic provider.
        //objPrivateKey.ProviderName = 'Microsoft Enhanced RSA and AES Cryptographic Provider';

        //TODO 名称固定？
        //objPrivateKey.ProviderName = GenReqForm.useCSP.value;
        objPrivateKey.ProviderName = document.certificateInstallForm.provider.value;

        //objPrivateKey.Length = 1024;
        var length = document.getElementById('keyLengthId');
        objPrivateKey.Length = length.value;


        // X509PrivateKeyUsageFlags
        // Specify a value that identifies the specific purpose for which a private key can be used.
        //
        // typedef enum {
        // XCN_NCRYPT_ALLOW_USAGES_NONE = 0,
        // XCN_NCRYPT_ALLOW_DECRYPT_FLAG = 0x1,
        // XCN_NCRYPT_ALLOW_SIGNING_FLAG = 0x2,
        // XCN_NCRYPT_ALLOW_KEY_AGREEMENT_FLAG = 0x4,
        // XCN_NCRYPT_ALLOW_ALL_USAGES = 0xffffff
        // } X509PrivateKeyUsageFlags;
        objPrivateKey.KeySpec = '1';

        // X509ProviderType
        // Specification of the cryptographic standards and algorithms
        //
        // typedef enum X509ProviderType {
        // XCN_PROV_NONE = 0,
        // XCN_PROV_RSA_FULL = 1,
        // XCN_PROV_RSA_SIG = 2,
        // XCN_PROV_DSS = 3,
```



```
    // Retrieving the encoded certificate request object
    var sCertificate = objEnroll.CreateRequest(1);

    //TODO 名称固定？
    //document.GenReqForm.csr.value = sCertificate;
    document.certificateInstallForm.pkcs10.value = sCertificate;

//  document.GenReqForm.submit();

    return true;
```

I decided to also dump the `.html` files to disk and then found this gem.



```
<!-- END CASE UNKNOWN -->
<input type="hidden" name="pkcs10" value="" id="pkcs10">
<input type="hidden" name="country" value="" id="c">
<input type="hidden" name="stateOrProvince" value="" id="s">
<input type="hidden" name="locality" value="" id="l">
<input type="hidden" name="organization" value="" id="o">

<input type="hidden" name="commonName" value="OS2[redacted]" id="cn">
<input type="hidden" name="email" value="" id="e">
<input type="hidden" name="keyLength" value="4096" id="keyLengthId">
<input type="hidden" name="token" value="005e[redacted]">
<script>
```

It seems to contain a series of `type="hidden"` fields which were pre-defined by the attestation/pickup server. The `keyLength` is already set to `4096`. The `commonName` aka `CN` is set to something that appears to be my account identifier. The last important one is the `token` field, which seems to be some sort of temporary password for the certificate pickup.

Generating my own CSR

Okay, I think I have enough knowledge to create my own keys and CSR to fool the attestation server!

I rush to my trashcan and take out the Yubikey from months ago as I need it back to follow all the steps in this CSR attestation with yubikey manager guide. I learned that the Yubikey cannot have 4096-bit RSA, but now that I am in control of these hidden input fields, I might be able to change the bit size.

These are the commands used on `ykman` to set up an RSA 2048-bit key.

```
& 'C:\Program Files\Yubico\YubiKey Manager\ykman.exe' piv reset -f
& 'C:\Program Files\Yubico\YubiKey Manager\ykman.exe' piv keys generate --pin-policy
ONCE --touch-policy CACHED --algorithm RSA2048 --management-key
010203040506070801020304050607080102030405060708 9a "$($env:TEMP)\junkfile"
& 'C:\Program Files\Yubico\YubiKey Manager\ykman.exe' piv certificates generate -s
"CN=Selfsigned" --pin 123456 --management-key
010203040506070801020304050607080102030405060708 9a "$($env:TEMP)\junkfile"
& 'C:\Program Files\Yubico\YubiKey Manager\ykman.exe' piv objects generate --
management-key 010203040506070801020304050607080102030405060708 chuid
& 'C:\Program Files\Yubico\YubiKey Manager\ykman.exe' piv certificates  export f9
"$($env:TEMP)\yubico_intermediate.cer"
& 'C:\Program Files\Yubico\YubiKey Manager\ykman.exe' piv keys attest 9a
"$($env:TEMP)\yubico_attestation.cer"
```

The only thing I have changed here is the `"CN=Selfsigned"` to match the `CN` value from the `"commonName"` hidden input field. I then proceed to build the Certificate Signing Request (CSR) and export it as base64.

Injecting my DIY CSR

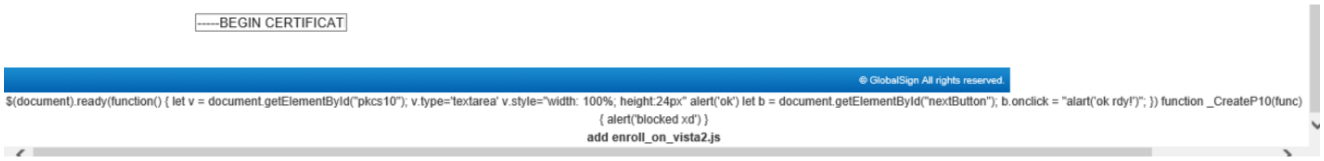
Now, weaponized with a self-made CSR, I can start thinking of a way to inject this into the form post. I will be using `mitmdump -s proxy.py` with the below `proxy.py`.

```
class FeribProxy(object):
        def __init__(self):
                pass
        def response(self, flow):
                url = flow.request.url
                if "enroll_on_vista2.js" in url:
                        flow.response.text = open("enroll_on_vista2.js", 'r').read()

addons = [
        FeribProxy()
]
```

The above Python script will look for a file named `enroll_on_vista2.js` and replace it with our malicious `enroll_on_vista2.js` file as seen below.

```
<script style="display:block">
$(document).ready(function() {
        let v = document.getElementById("pkcs10");
        v.type='test'

        let b = document.getElementById("nextButton");
        b.onclick = '';

        alert('ok')
})
</script>
```
add enroll_on_vista2.js

This file will inject this `<script style="display: block">` tag. Notice how we inject a `<script>` tag which has this `"display:block"` style property set. This means that the element is actually made visible on the website itself.



-----BEGIN CERTIFICAT

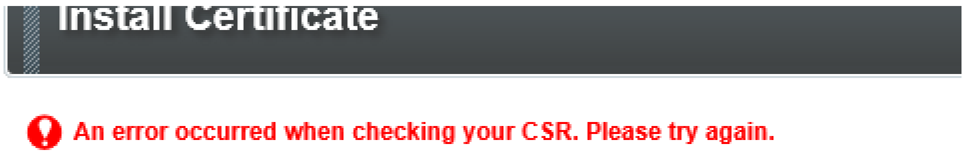
© GlobalSign All rights reserved.

$(document).ready(function() { let v = document.getElementById("pkcs10"); v.type='textarea' v.style="width: 100%; height:24px" alert('ok') let b = document.getElementById("nextButton"); b.onclick = "alart('ok rdy!')"; }) function _CreateP10(func) { alert('blocked xd') }

add enroll_on_vista2.js

It's rather annoying to not have any sort of devtools on this IE Compatible browser, so having this visual feedback of the script being injected helps so much.

Now that we have working inputs on the website, I will do a quick test and simply set the hidden `pkcs10` field to `"test"` and press the submit button.



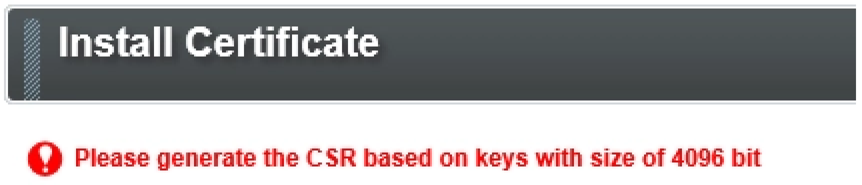
An error occurred when checking your CSR. Please try again.

Yay, I am under the assumption that this generic obscure error message indicates some completely unexpected error has happened. Most likely as it was unable to parse the expected pkcs10.

Okay, now I just force 2048 bit like so

```
let v = document.getElementById("keyLength");
v.type='2048'
```

I make the change, I see my `<script>` content changed on the webpage, and with full confidence I click submit!



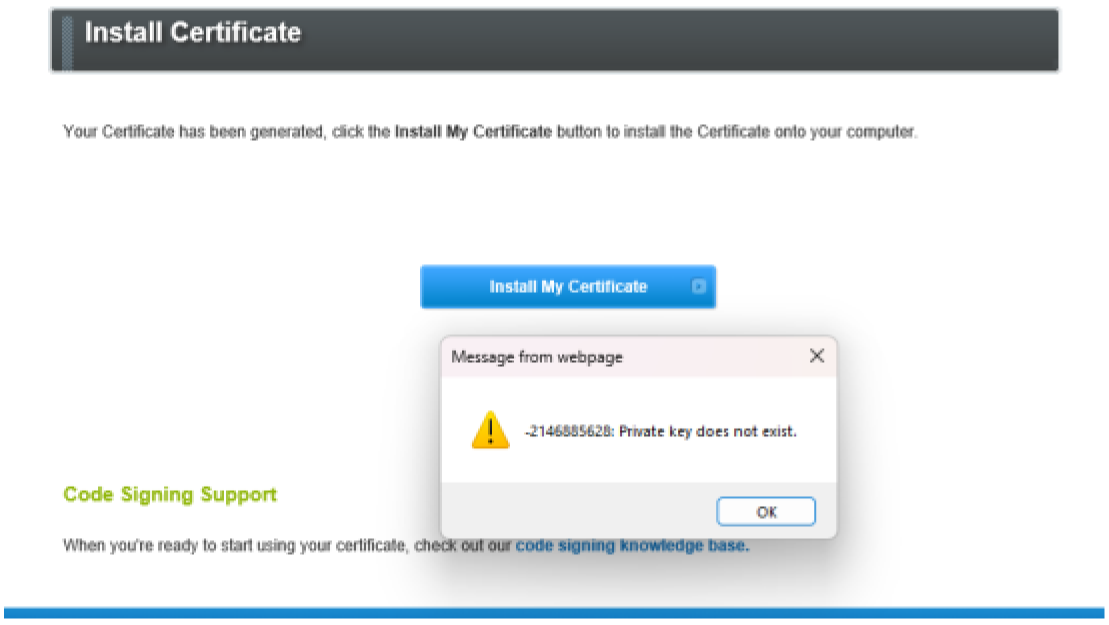
Please generate the CSR based on keys with size of 4096 bit

Sad...

Generating a CSR on disk

Okay.. threw my Yubikey across the room and finally gave up on Yubikeys. I will be using `openssl` cli commands like a normal person. Nothing new here, I follow the steps from some random article and make sure to set my `CN` correctly.

Okay, exported it to pkcs10 and pasted it into my script, hit submit, and this happened.


Install Certificate

Your Certificate has been generated, click the **Install My Certificate** button to install the Certificate onto your computer.

Message from webpage

-2146885628: Private key does not exist.

Code Signing Support

When you're ready to start using your certificate, check out our code signing knowledge base.

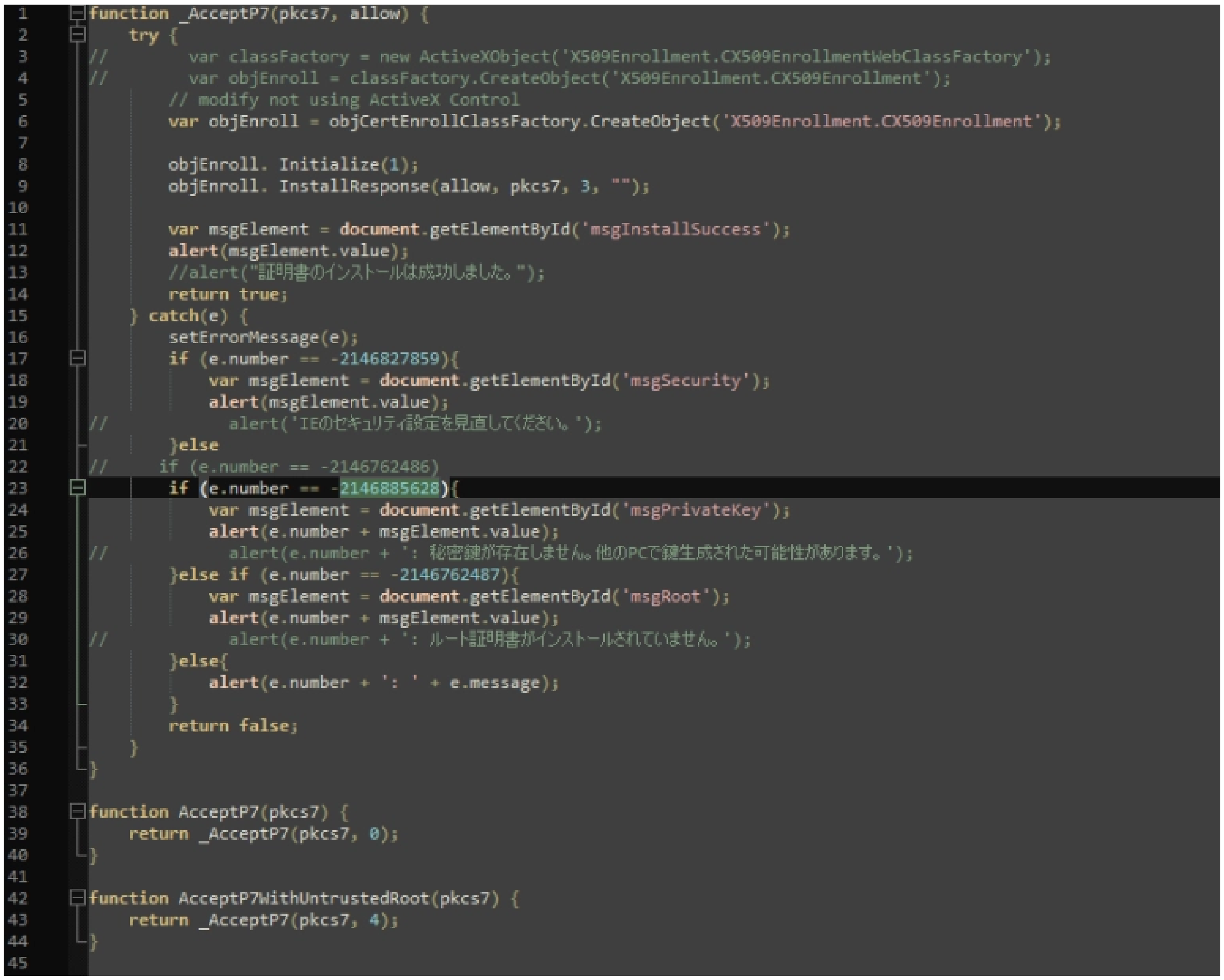
Yay? my Certificate has been generated? But I still have an error message?

Recovering the pkcs7 response

Okay, still surprised that it got accepted, but I have bigger things to worry about now. My pickup token is consumed and the response key is most likely in my Edge.exe process, somewhere in the RAM memory? I would also assume it to be some sort of base64 encoded string?

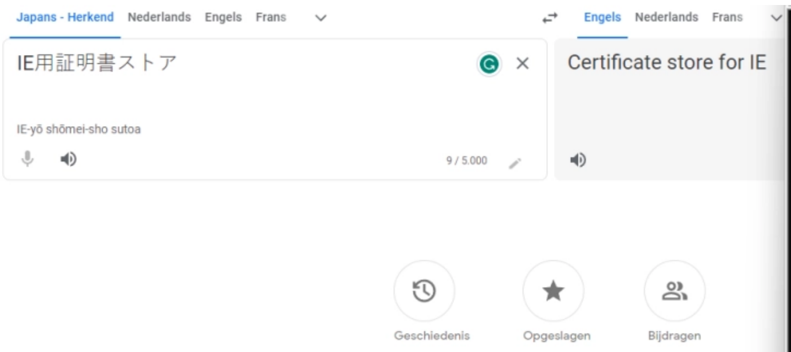
Did my `mitmproxy` log anything? Of course not, I'm running `mitmdump` with this custom script.

Okay, no panic, let's revisit my collection of `.js` files and figure out what error code that is.



```javascript
function _AcceptP7(pkcs7, allow) {
    try {
//          var classFactory = new ActiveXObject('X509Enrollment.CX509EnrollmentWebClassFactory');
//          var objEnroll = classFactory.CreateObject('X509Enrollment.CX509Enrollment');
        // modify not using ActiveX Control
        var objEnroll = objCertEnrollClassFactory.CreateObject('X509Enrollment.CX509Enrollment');

        objEnroll. Initialize(1);
        objEnroll. InstallResponse(allow, pkcs7, 3, "");

        var msgElement = document.getElementById('msgInstallSuccess');
        alert(msgElement.value);
        //alert("証明書のインストールは成功しました。");
        return true;
    } catch(e) {
        setErrorMessage(e);
        if (e.number == -2146827859){
            var msgElement = document.getElementById('msgSecurity');
            alert(msgElement.value);
//              alert('IEのセキュリティ設定を見直してください。');
        }else
//      if (e.number == -2146762486)
        if (e.number == -2146885628){
            var msgElement = document.getElementById('msgPrivateKey');
            alert(e.number + msgElement.value);
//              alert(e.number + ': 秘密鍵が存在しません。他のPCで鍵生成された可能性があります。');
        }else if (e.number == -2146762487){
            var msgElement = document.getElementById('msgRoot');
            alert(e.number + msgElement.value);
//              alert(e.number + ': ルート証明書がインストールされていません。');
        }else{
            alert(e.number + ': ' + e.message);
        }
        return false;
    }
}

function AcceptP7(pkcs7) {
    return _AcceptP7(pkcs7, 0);
}

function AcceptP7WithUntrustedRoot(pkcs7) {
    return _AcceptP7(pkcs7, 4);
}
```

There is our error code, so who is calling this `_AcceptP7` function? And what's the meaning of this Chinese comment?

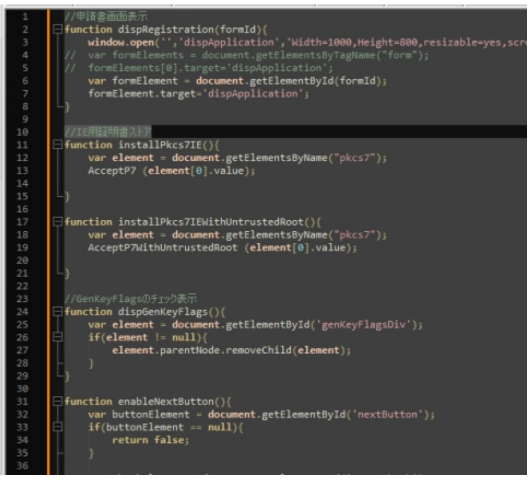


IE用証明書ストア

IE-yō shōmei-sho sutoa

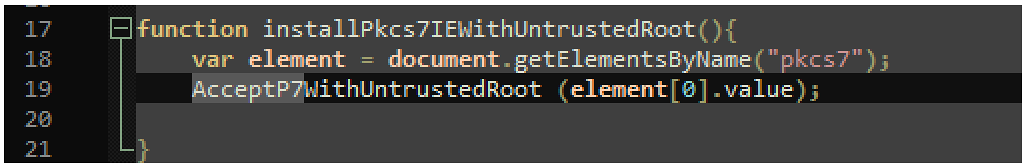
9 / 5.000

Certificate store for IE



```javascript
//申請書画面表示
function dispRegistration(formId){
    window.open('','dispApplication','Width=1000,Height=800,resizable=yes,scr
//  var formElements = document.getElementsByTagName("form");
//  formElements[0].target='dispApplication';
    var formElement = document.getElementById(formId);
    formElement.target='dispApplication';
}

//IE用証明書ストア
function installPkcs7IE(){
    var element = document.getElementsByName("pkcs7");
    AcceptP7 (element[0].value);

}

function installPkcs7IEWithUntrustedRoot(){
    var element = document.getElementsByName("pkcs7");
    AcceptP7WithUntrustedRoot (element[0].value);

}

//GenKeyFlagsのチェック表示
function dispGenKeyFlags(){
    var element = document.getElementById('genKeyFlagsDiv');
    if(element != null){
        element.parentNode.removeChild(element);
    }
}

function enableNextButton(){
    var buttonElement = document.getElementById('nextButton');
    if(buttonElement == null){
        return false;
    }
```

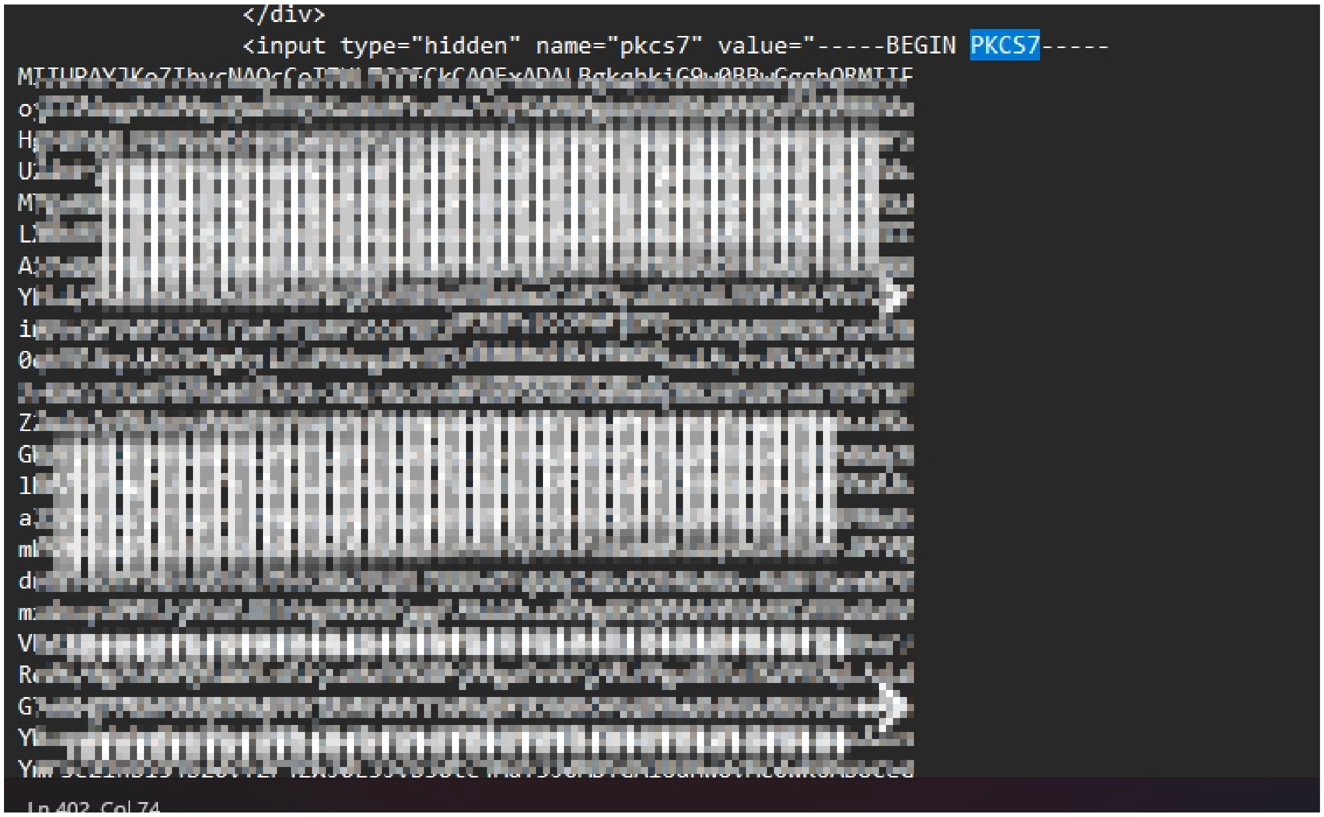
Chinese comments seem useless, so I Ctrl+F and find something calling `AcceptP7`?



```
function installPkcs7IEWithUntrustedRoot(){
    var element = document.getElementsByName("pkcs7");
    AcceptP7WithUntrustedRoot (element[0].value);

}
```

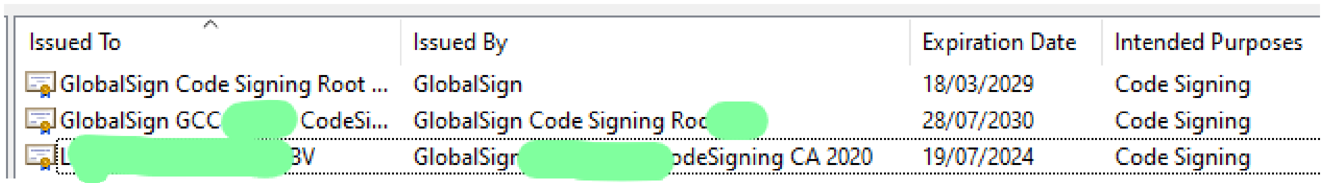
I look around and see `installPkcs7IEWithUntrustedRoot()` which calls `document.getElementByName("pkcs7")`. This `"pkcs7"` must be one of the *(hidden?)* response fields given by the webserver?

I am desperate here, so I attach CheatEngine to Edge.exe, I scan the process memory for string `"pkcs7"` to hope and find some sort of JS/HTML code.



```
                    </div>
                    <input type="hidden" name="pkcs7" value="-----BEGIN PKCS7-----
[redacted]
  Ln 402, Col 74
```

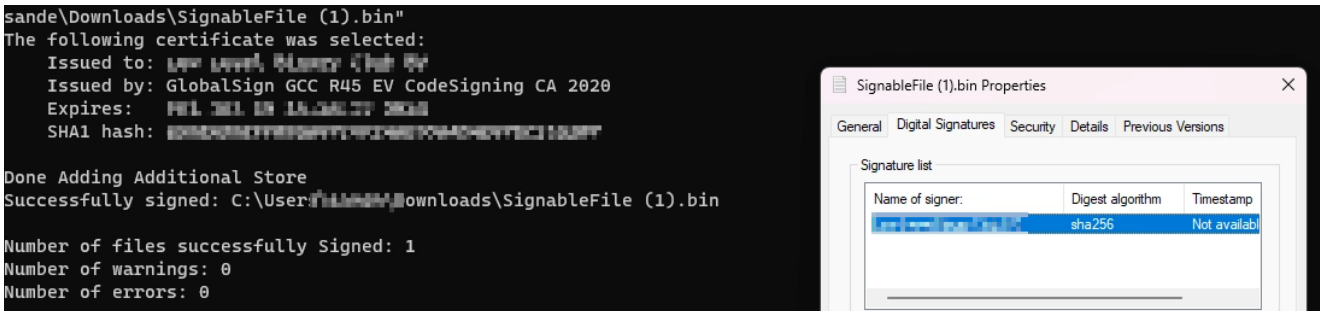
As I looked for bronze, I found gold! Thanks to the oldschool response of the webserver, I found this hidden input containing the pkcs7.



Issued To	Issued By	Expiration Date	Intended Purposes
GlobalSign Code Signing Root ...	GlobalSign	18/03/2029	Code Signing
GlobalSign GCC[redacted] CodeSi...	GlobalSign Code Signing Roo[redacted]	28/07/2030	Code Signing
L[redacted]3V	GlobalSign[redacted]odeSigning CA 2020	19/07/2024	Code Signing

Finally, I use `openssl` once more and convert my pkcs7 file into a `.pfx` file 🤩

Let's Sign Something!



```
sande\Downloads\SignableFile (1).bin"
The following certificate was selected:
    Issued to: [redacted]
    Issued by: GlobalSign GCC R45 EV CodeSigning CA 2020
    Expires:   [redacted]
    SHA1 hash: [redacted]

Done Adding Additional Store
Successfully signed: C:\User[redacted]ownloads\SignableFile (1).bin

Number of files successfully Signed: 1
Number of warnings: 0
Number of errors: 0
```

SignableFile (1).bin Properties

General | Digital Signatures | Security | Details | Previous Versions

Signature list

Name of signer:	Digest algorithm	Timestamp
[redacted]	sha256	Not availabl

Tada! Countless hours of frustration, support, emailing, vetting, reversing, debugging, and a whole lot of other bullshit, along with the cost of a useless eToken and Yubikey devices.

Finally, I can move on to the next step: requesting vetting for Microsoft Driver Signing 🙃