black hat®

USA 2016

JULY 30 – AUGUST 4, 2016 / MANDALAY BAY / LAS VEGAS

Cunning with CNG: Soliciting Secrets from Schannel

“Black Hat Sound Bytes”

What you get out of this talk

- ❑ Ability to decrypt Schannel TLS connections that use ephemeral key exchanges
- ❑ Ability to decrypt and extract private certificate and session ticket key directly from memory
- ❑ Public Cert/SNI to PID/Logon Session Mapping

Agenda

- A very short SSL/TLS Review
- A background on Schannel & CNG
- The Secret Data
- The Forensic Context
- Demo >.>

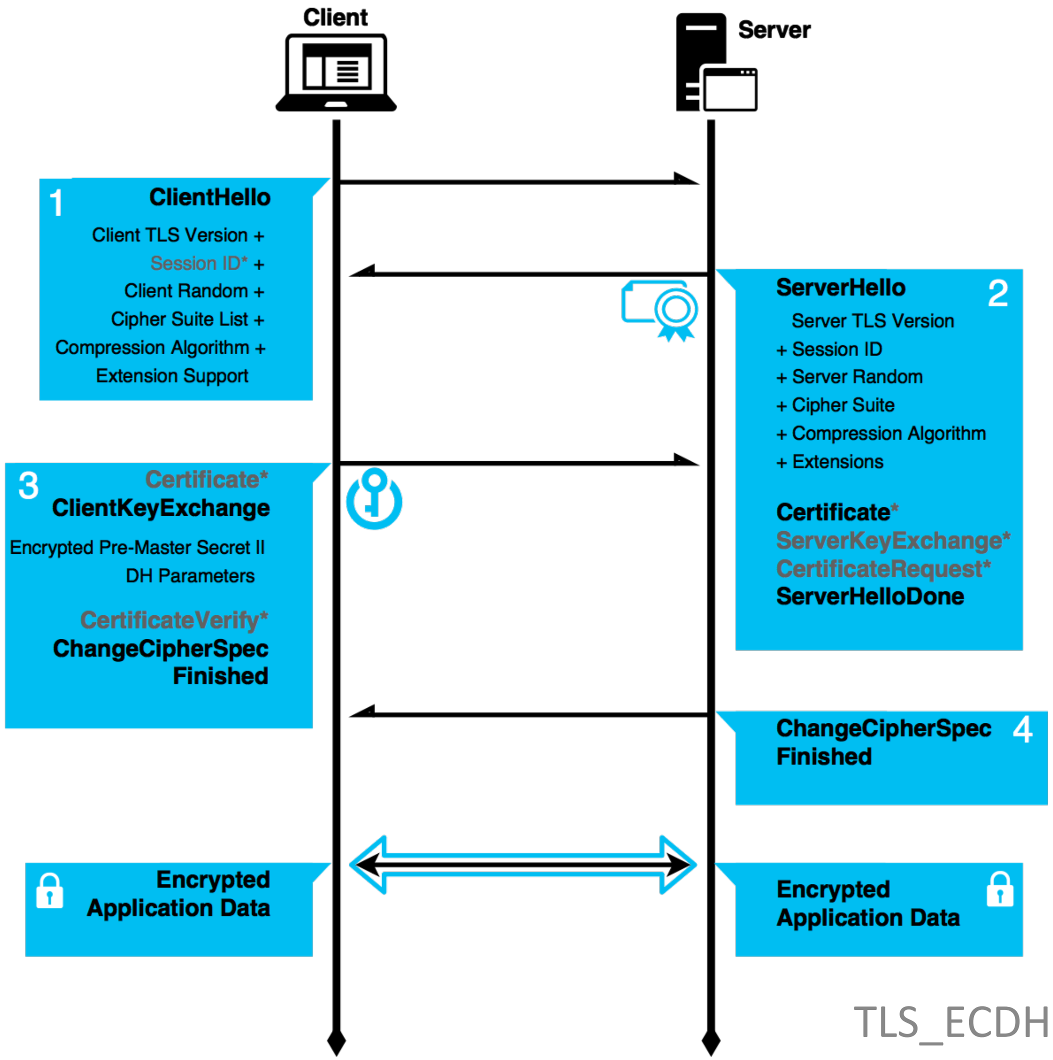
Disclaimer

- This is NOT an exploit
  - It’s just the spec :D
  - …and some implementation specific oddities
- Microsoft has done nothing [especially] wrong
  - To the contrary, their documentation was actually pretty great
- Windows doesn’t track sessions for processes that load their own TLS libs
  - I’m looking at you Firefox and Chrome
- Windows doesn’t track sessions for process that don’t use TLS…
  - That’d be you TeamViewer...

Background

TLS, Schannel, and CNG

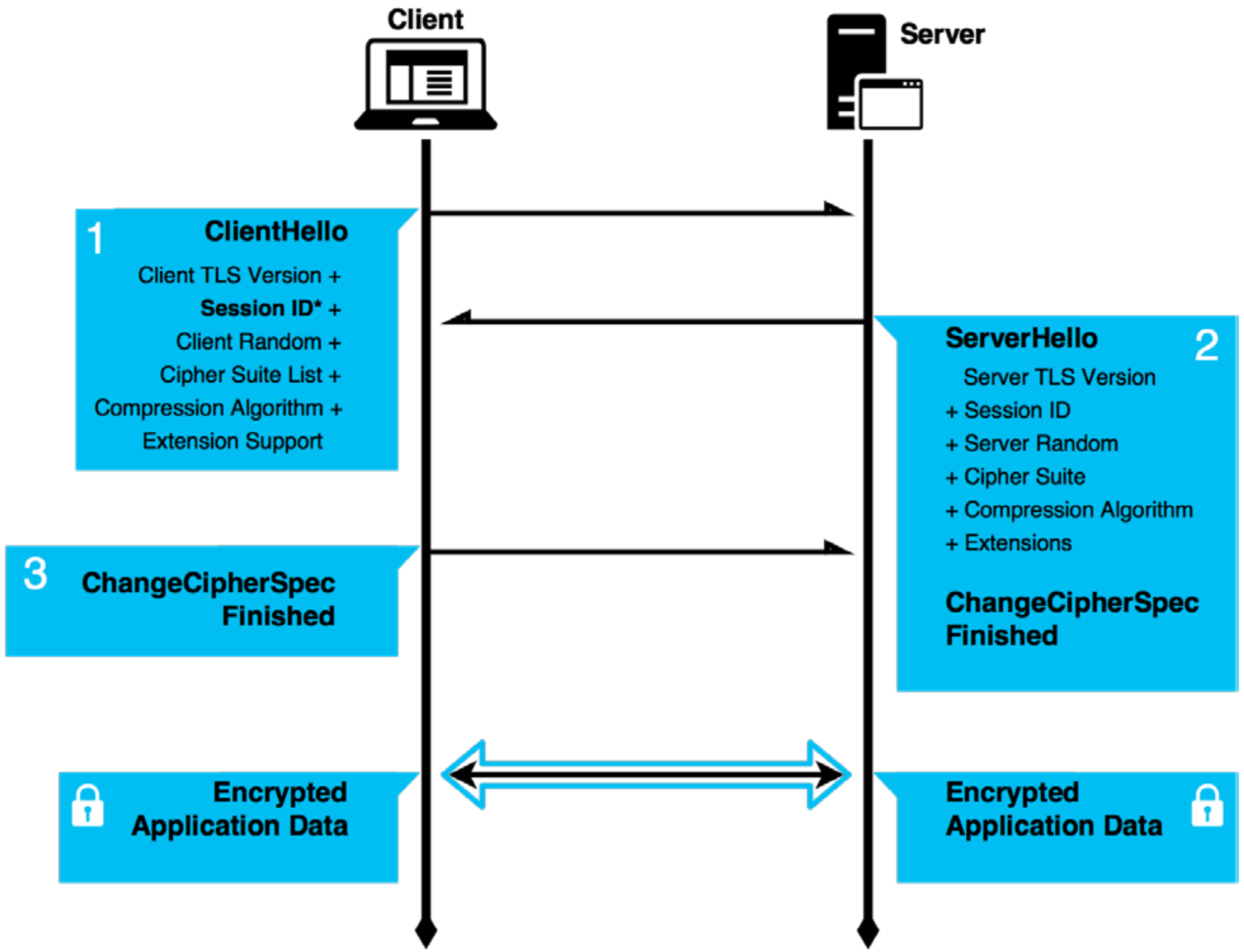
The infamous TLS Handshake



Initial Connection

TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256

The ~~infamous~~ TL~~S~~DR; Handshake



Session Resumption

Perfect Forward Secrecy

What we *want* to do

- ❑ One time use keys, no sending secrets!

What TLS *actually* does

- ❑ Caches values to enable resumption
  - ❑ recommends `An upper limit of 24 hours is suggested for session ID lifetimes`
- ❑ When using session ticket extension, sends the encrypted state over the network
  - ❑ basically returning to the issue with RSA, but using a more ephemeral key...

What implementations *also* do

- ❑ Store symmetric key schedules (so you can find the otherwise random keys...)
- ❑ Cache ephemeral keys and reuse for a while...

Schannel & CNG

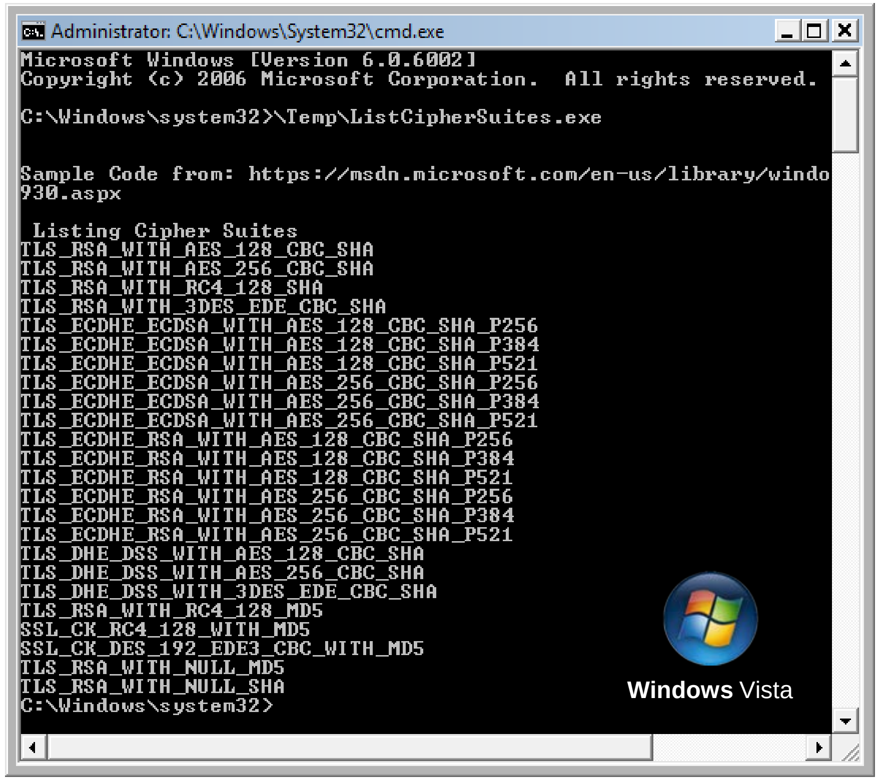
Secure Channel

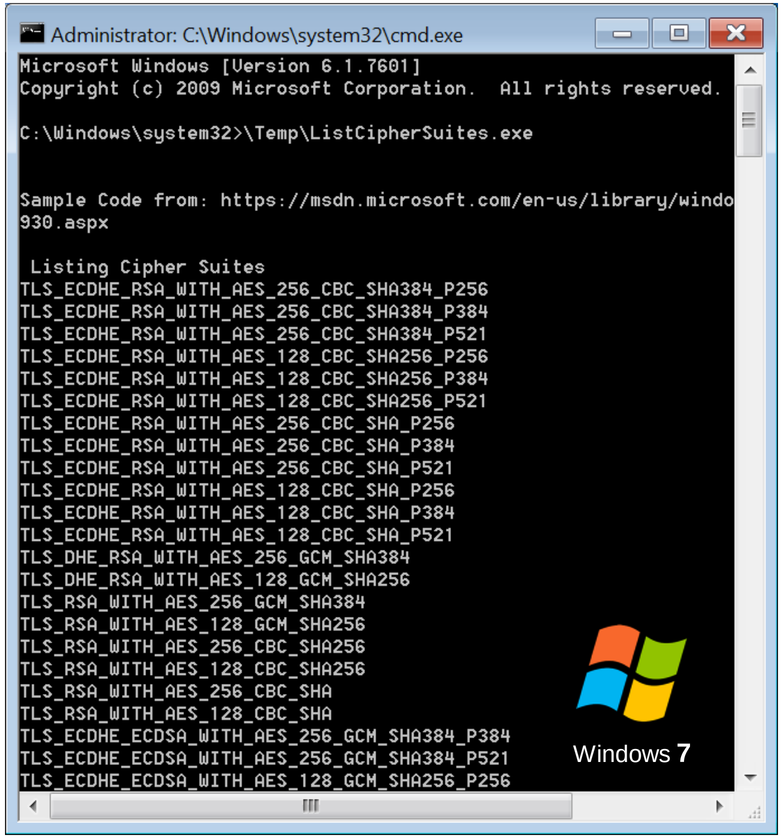
- It's TLS -> the Secure Channel for Windows!
- A library that gets loaded into the "key isolation process" and the "client" process
  - Technically a Security Support Provider (SSP)
- Spoiler: the Key Isolation process is LSASS

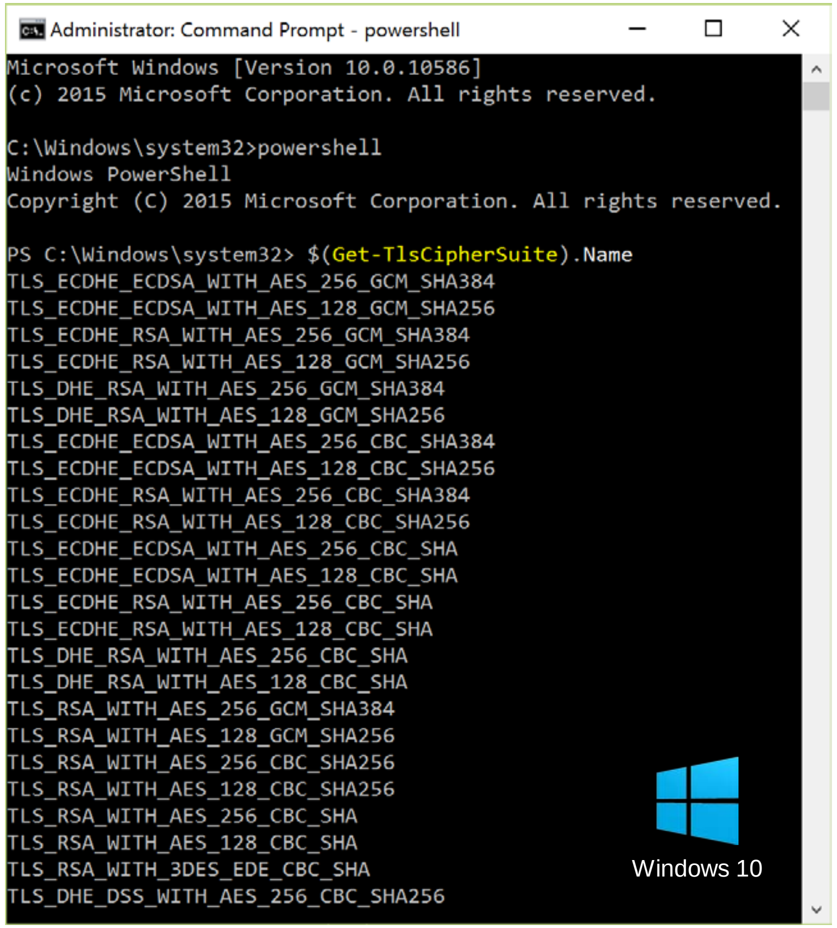
The CryptoAPI-Next Generation (CNG)

- Introduced in Vista (yes you read correctly)
- Provides Common Criteria compliance
- Used to store secrets and 'crypt them
  - Storage via the Key Storage Providers (KSPs)
  - Generic data encryption via DPAPI
  - Also brings modern ciphers to Windows (AES for example) and ECC
- Importantly, `ncrypt` gets called out as the "key storage router" and gateway to the CNG Key Isolation service

Schannel Prefered Cipher Suites







*ListCipherSuites sample code found here: https://technet.microsoft.com/en-us/library/bb870930.aspx

Microsoft’s TLS/SSL Docs

- **ClientCacheTime:** “The first time a client connects to a server through the Schannel SSP, a full TLS/SSL handshake is performed.”
- “When this is complete, **the master secret, cipher suite, and certificates are stored** in the session cache on the respective client and server.”*
- **ServerCacheTime:** “…Increasing ServerCacheTime above the default values **causes Lsass.exe to consume additional memory**. Each session cache element typically requires 2 to 4 KB of memory”*
- **MaximumCacheSize**: “This entry controls the maximum number of cache elements. […] **The default value is 20,000 elements**.” *

*TLS/SSL Settings quoted from here: https://technet.microsoft.com/en-us/library/dn786418(v=ws.11).aspx

Schannel Ops

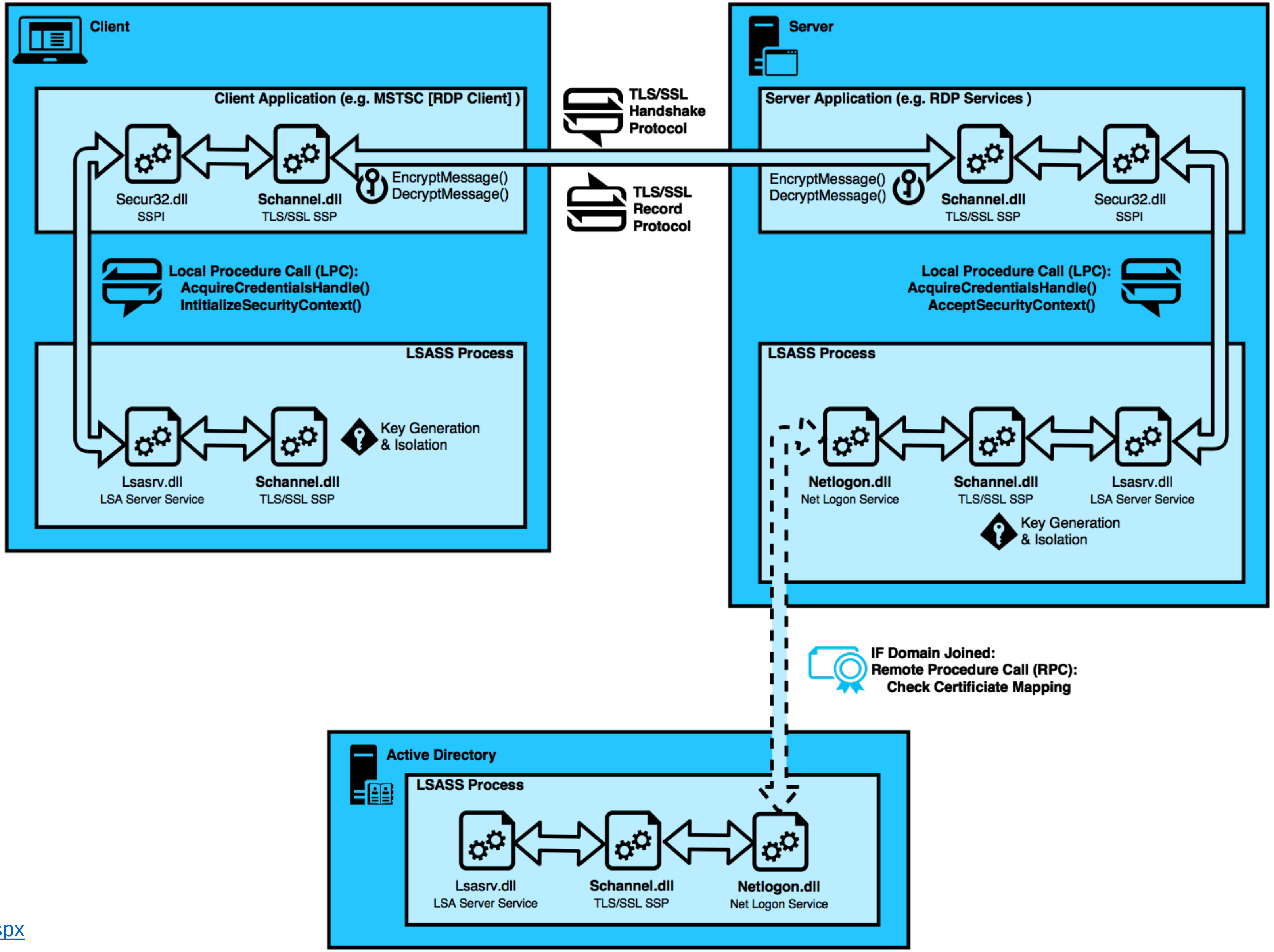


Diagram based on: https://technet.microsoft.com/en-us/library/dn786429.aspx

CNG Key Isolation

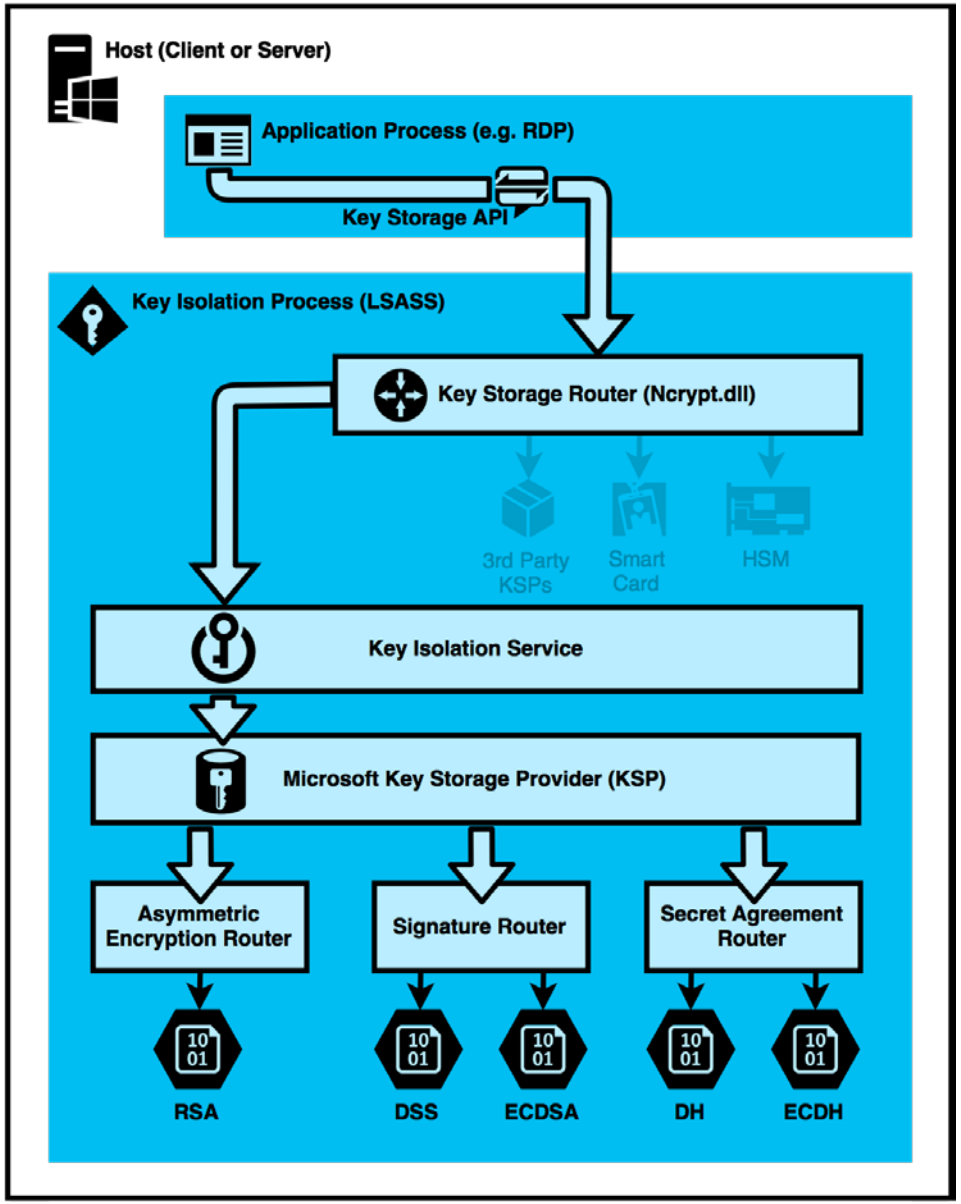
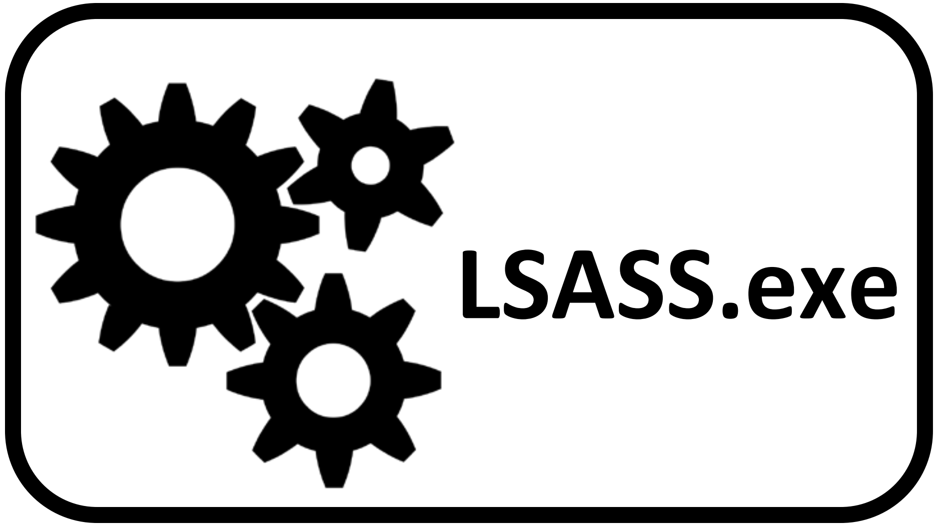


Diagram based on: https://msdn.microsoft.com/en-us/library/windows/desktop/bb204778.aspx

Background Summary

Were Looking Here



For These

Because of That

What are we trying to accomplish?

We want to be able to see data that has been protected with TLS/SSL and subvert efforts at implementing Perfect Forward Secrecy

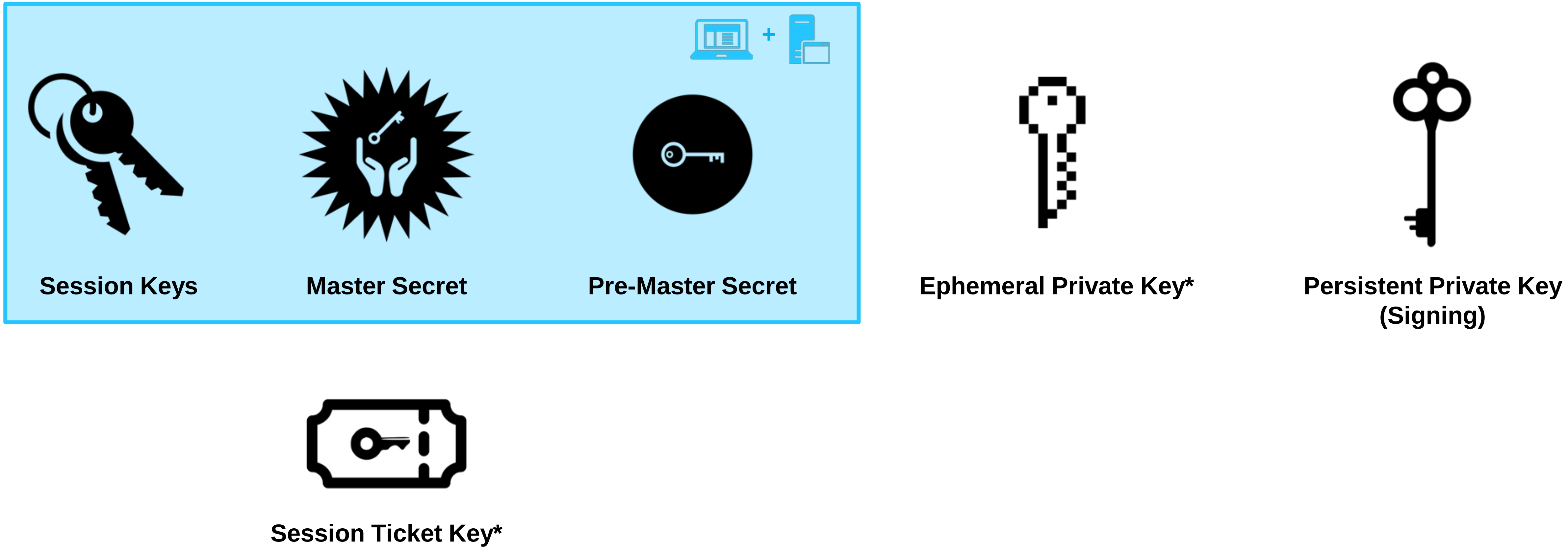
We want to gather any contextual information that we can use for forensic purposes, regardless of whether or not we can accomplish the above

We (as an adversary) want to be able to get access to a single process address space and be able to dump out things that would enable us to monitor/modify future traffic, or possibly impersonate the target

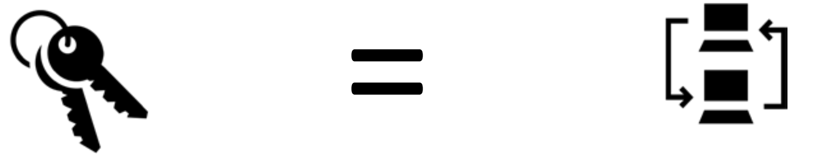
- We want to do this without touching disk

Secrets

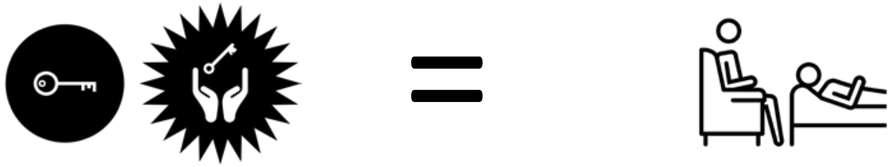
The Keys


Session Keys
Master Secret
Pre-Master Secret
Ephemeral Private Key*
Persistent Private Key (Signing)
Session Ticket Key*

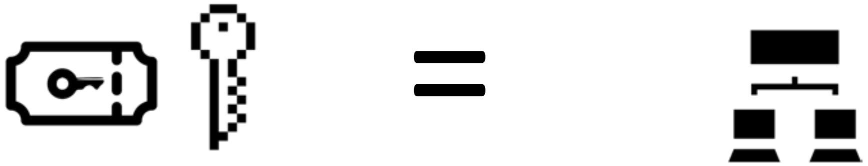
The Keys? What do they get us?



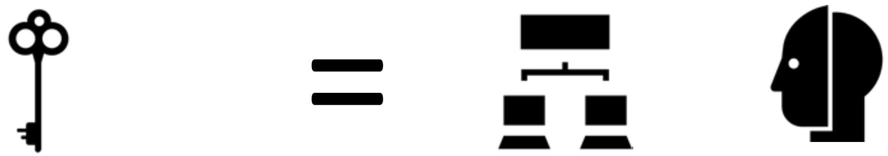
a single connection



a single session

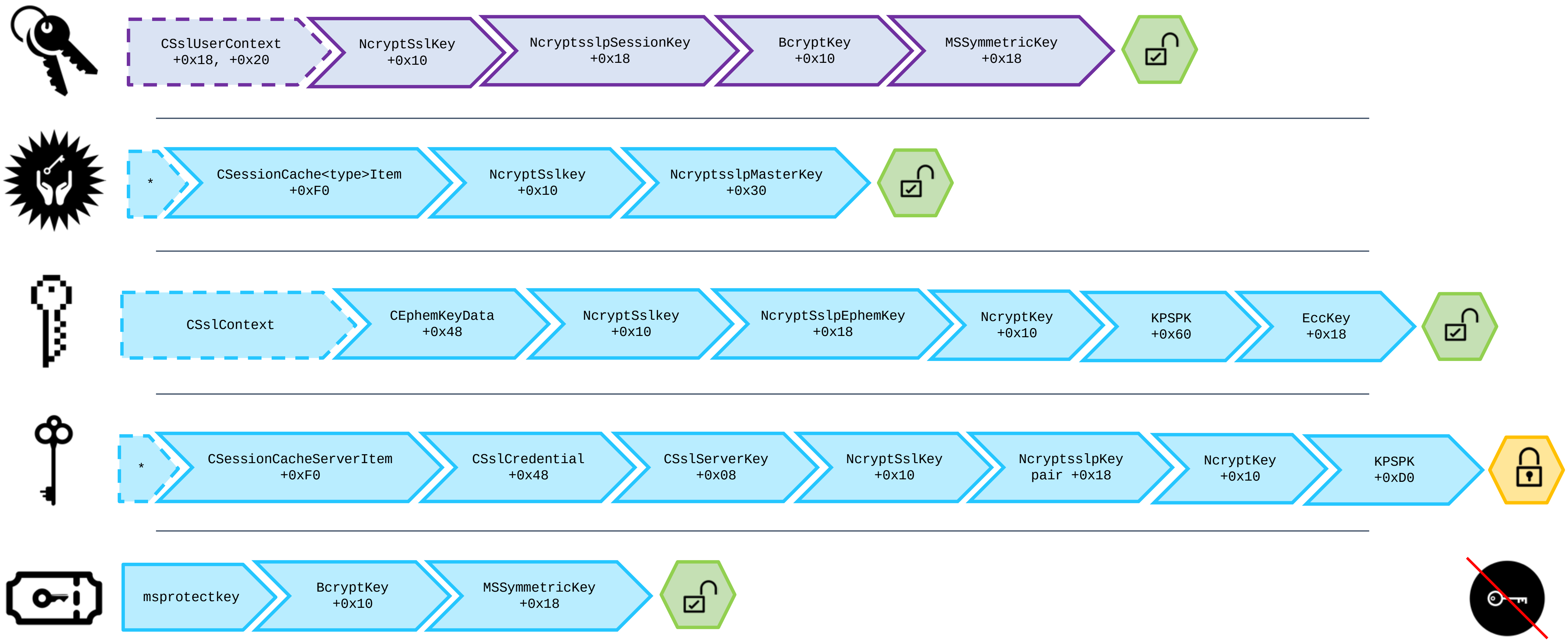


multiple sessions



multiple sessions + identity

The Keys? We got 'em...all.



Session Keys

- Smallest scope / most ephemeral
- Required for symmetric encrypted comms
- Not going to be encrypted

Approach Premise:

- Start with AES
- AES keys are relatively small and pseudo-random
- AES key schedules are larger and deterministic
  - … they are a "schedule" after all.
- Key schedules usually calculated once and stored*

- Let's scan for matching key schedules on both hosts

C:\Windows\system32\cmd.exe

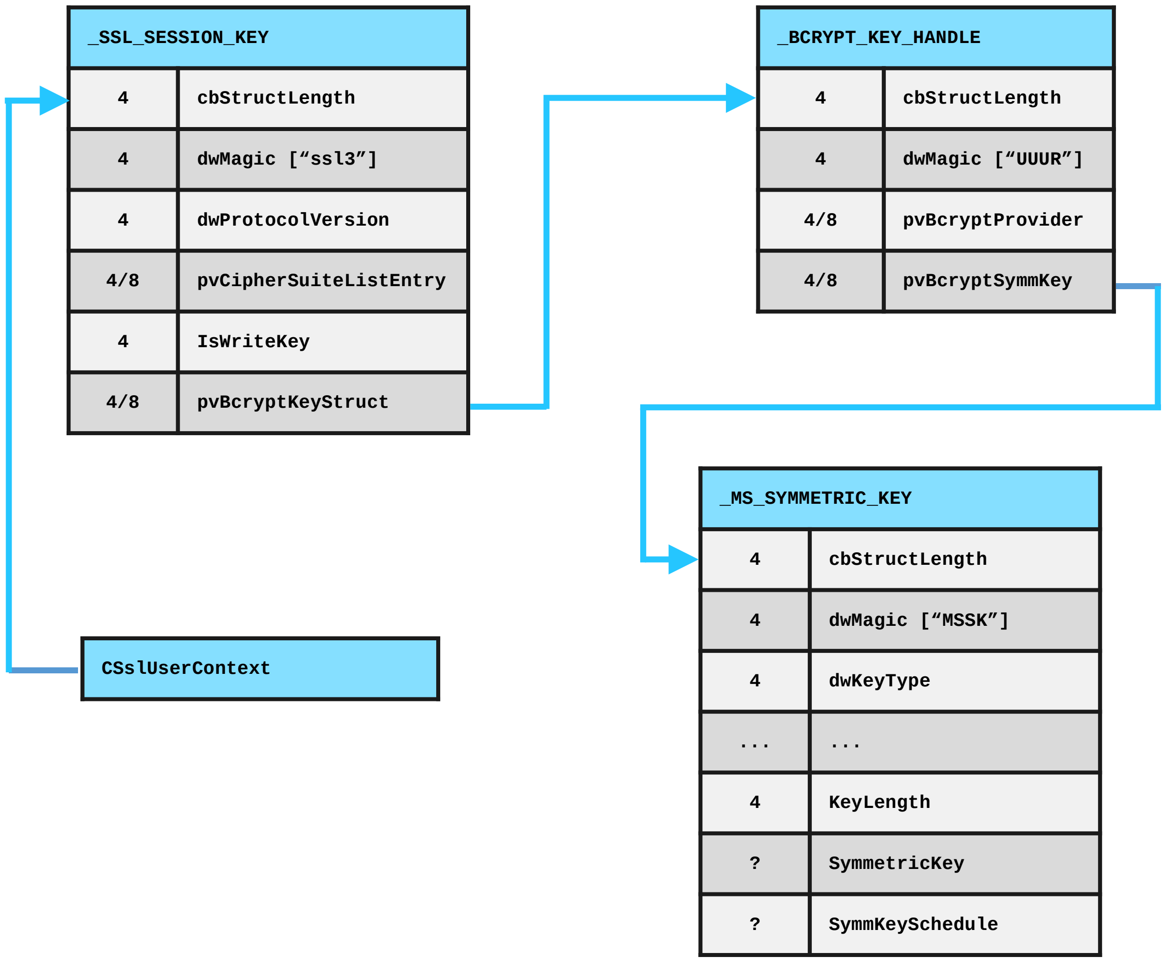
```
C:\TMP>findaes.exe rdp_mstsc.DMP
Searching rdp_mstsc.DMP
Found AES-256 key schedule at offset 0x3158ac:
b9 f0 65 ef 0a 27 33 62 0d 92 3d 2a 1e ba 24 3b 9a 1d 94 a8 70 d4 b5 ab 08 18 d6 f8 d8 04 1d 07
Found AES-256 key schedule at offset 0x3162ac:
b9 f0 65 ef 0a 27 33 62 0d 92 3d 2a 1e ba 24 3b 9a 1d 94 a8 70 d4 b5 ab 08 18 d6 f8 d8 04 1d 07
Found AES-256 key schedule at offset 0xcd71dc:
08 57 03 03 d2 71 5c bb 23 b1 69 1d b8 9a 2a ea 00 83 13 78 75 a5 84 a3 0f 21 af 5d 5c 4b 8c f9
Found AES-256 key schedule at offset 0xcd7bdc:
08 57 03 03 d2 71 5c bb 23 b1 69 1d b8 9a 2a ea 00 83 13 78 75 a5 84 a3 0f 21 af 5d 5c 4b 8c f9
Found AES-256 key schedule at offset 0xcfeadc:
b0 75 6f 15 5c 70 a5 ec 8e 4c e3 c9 f3 b3 ff 33 80 04 ed 43 d4 a6 36 b7 6e 41 8f aa df 6c e1 b9
Found AES-256 key schedule at offset 0xcff4dc:
b0 75 6f 15 5c 70 a5 ec 8e 4c e3 c9 f3 b3 ff 33 80 04 ed 43 d4 a6 36 b7 6e 41 8f aa df 6c e1 b9
Found AES-256 key schedule at offset 0x171571c:
da 37 46 b4 a7 db e9 f5 b6 7f 27 ea a2 d3 26 c6 cc 65 30 42 f1 68 74 bb fb f5 c9 ef 64 f7 30 9c
Found AES-256 key schedule at offset 0x171611c:
da 37 46 b4 a7 db e9 f5 b6 7f 27 ea a2 d3 26 c6 cc 65 30 42 f1 68 74 bb fb f5 c9 ef 64 f7 30 9c

C:\TMP>_
```

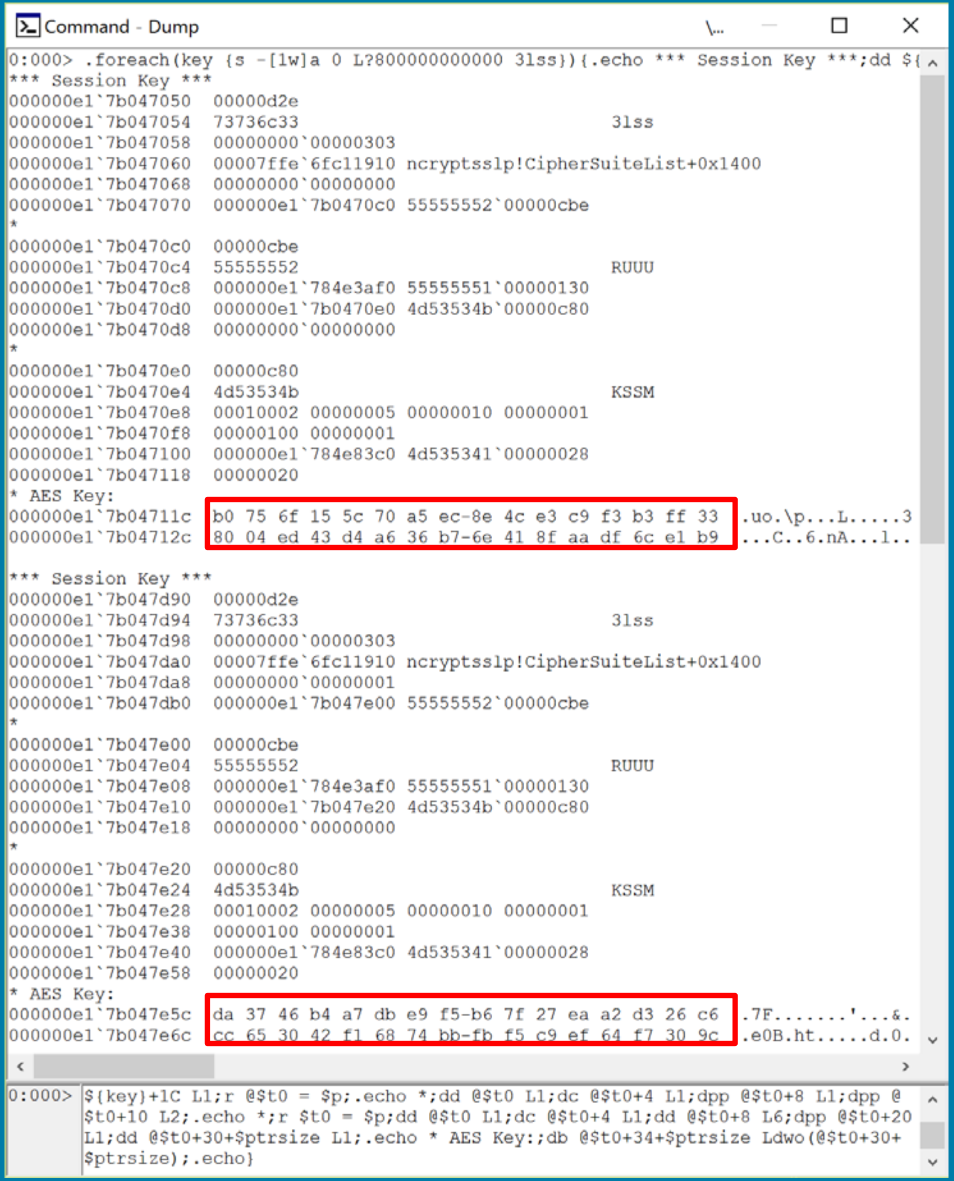
Select C:\Windows\system32\cmd.exe

```
C:\TMP>findaes.exe rdp_svchost.DMP
Searching rdp_svchost.DMP
Found AES-256 key schedule at offset 0x9bd7f50:
b9 f0 65 ef 0a 27 33 62 0d 92 3d 2a 1e ba 24 3b 9a 1d 94 a8 70 d4 b5 ab 08 18 d6 f8 d8 04 1d 07
Found AES-256 key schedule at offset 0x9c9b1c0:
b0 75 6f 15 5c 70 a5 ec 8e 4c e3 c9 f3 b3 ff 33 80 04 ed 43 d4 a6 36 b7 6e 41 8f aa df 6c e1 b9
Found AES-256 key schedule at offset 0x9c9bbc0:
b0 75 6f 15 5c 70 a5 ec 8e 4c e3 c9 f3 b3 ff 33 80 04 ed 43 d4 a6 36 b7 6e 41 8f aa df 6c e1 b9
Found AES-256 key schedule at offset 0x9c9bf00:
da 37 46 b4 a7 db e9 f5 b6 7f 27 ea a2 d3 26 c6 cc 65 30 42 f1 68 74 bb fb f5 c9 ef 64 f7 30 9c
Found AES-256 key schedule at offset 0x9c9c900:
da 37 46 b4 a7 db e9 f5 b6 7f 27 ea a2 d3 26 c6 cc 65 30 42 f1 68 74 bb fb f5 c9 ef 64 f7 30 9c
Found AES-256 key schedule at offset 0x9ca8740:
b9 f0 65 ef 0a 27 33 62 0d 92 3d 2a 1e ba 24 3b 9a 1d 94 a8 70 d4 b5 ab 08 18 d6 f8 d8 04 1d 07
Found AES-256 key schedule at offset 0x9ca9140:
b9 f0 65 ef 0a 27 33 62 0d 92 3d 2a 1e ba 24 3b 9a 1d 94 a8 70 d4 b5 ab 08 18 d6 f8 d8 04 1d 07
Found AES-256 key schedule at offset 0x9cb97b0:
08 57 03 03 d2 71 5c bb 23 b1 69 1d b8 9a 2a ea 00 83 13 78 75 a5 84 a3 0f 21 af 5d 5c 4b 8c f9
Found AES-256 key schedule at offset 0x9cba1b0:
08 57 03 03 d2 71 5c bb 23 b1 69 1d b8 9a 2a ea 00 83 13 78 75 a5 84 a3 0f 21 af 5d 5c 4b 8c f9

C:\TMP>
```

FindAES from: http://jessekornblum.com/tools/

Session Keys




Command - Dump

```
0:000> .foreach(key {s -[1w]a 0 L?800000000000 3lss}){.echo *** Session Key ***;dd ${
*** Session Key ***
000000e1`7b047050  00000d2e
000000e1`7b047054  73736c33                           3lss
000000e1`7b047058  00000000`00000303
000000e1`7b047060  00007ffe`6fc11910 ncryptsslp!CipherSuiteList+0x1400
000000e1`7b047068  00000000`00000000
000000e1`7b047070  000000e1`7b0470c0 55555552`00000cbe
*
000000e1`7b0470c0  00000cbe
000000e1`7b0470c4  55555552                           RUUU
000000e1`7b0470c8  000000e1`784e3af0 55555551`00000130
000000e1`7b0470d0  000000e1`7b0470e0 4d53534b`00000c80
000000e1`7b0470d8  00000000`00000000
*
000000e1`7b0470e0  00000c80
000000e1`7b0470e4  4d53534b                           KSSM
000000e1`7b0470e8  00010002 00000005 00000010 00000001
000000e1`7b0470f8  00000100 00000001
000000e1`7b047100  000000e1`784e83c0 4d535341`00000028
000000e1`7b047118  00000020
* AES Key:
000000e1`7b04711c  b0 75 6f 15 5c 70 a5 ec-8e 4c e3 c9 f3 b3 ff 33  .uo.\p...L.....3
000000e1`7b04712c  80 04 ed 43 d4 a6 36 b7-6e 41 8f aa df 6c e1 b9  ...C..6.nA...l..

*** Session Key ***
000000e1`7b047d90  00000d2e
000000e1`7b047d94  73736c33                           3lss
000000e1`7b047d98  00000000`00000303
000000e1`7b047da0  00007ffe`6fc11910 ncryptsslp!CipherSuiteList+0x1400
000000e1`7b047da8  00000000`00000001
000000e1`7b047db0  000000e1`7b047e00 55555552`00000cbe
*
000000e1`7b047e00  00000cbe
000000e1`7b047e04  55555552                           RUUU
000000e1`7b047e08  000000e1`784e3af0 55555551`00000130
000000e1`7b047e10  000000e1`7b047e20 4d53534b`00000c80
000000e1`7b047e18  00000000`00000000
*
000000e1`7b047e20  00000c80
000000e1`7b047e24  4d53534b                           KSSM
000000e1`7b047e28  00010002 00000005 00000010 00000001
000000e1`7b047e38  00000100 00000001
000000e1`7b047e40  000000e1`784e83c0 4d535341`00000028
000000e1`7b047e58  00000020
* AES Key:
000000e1`7b047e5c  da 37 46 b4 a7 db e9 f5-b6 7f 27 ea a2 d3 26 c6  .7F.......'...&.
000000e1`7b047e6c  cc 65 30 42 f1 68 74 bb-fb f5 c9 ef 64 f7 30 9c  .e0B.ht.....d.0.

0:000> ${key}+1C L1;r @$t0 = $p;.echo *;dd @$t0 L1;dc @$t0+4 L1;dpp @$t0+8 L1;dpp @
$t0+10 L2;.echo *;r $t0 = $p;dd @$t0 L1;dc @$t0+4 L1;dd @$t0+8 L6;dpp @$t0+20
L1;dd @$t0+30+$ptrsize L1;.echo * AES Key:;db @$t0+34+$ptrsize Ldwo(@$t0+30+
$ptrsize);.echo}
```

Look familiar? Bcrypt keys are used a lot: think Mimikatz

The Ncrypt SSL Provider (ncryptsslp.dll)

Ncryptsslp Validation function Symbols

Command - Dump \\vmware-host\Share...

```
0:000> x /1 ncryptsslp!*Validate*
ncryptsslp!SslpValidateEphemeralHandle
ncryptsslp!SslpValidateMasterKeyHandle
ncryptsslp!SslpValidateProvHandle
ncryptsslp!SslpValidateHashHandle
ncryptsslp!SslpValidateKeyPairHandle

0:000>
```

These functions do three things:

- ❑ Check the first dword for a size value
- ❑ Check the second dword for a magic ID
- ❑ Return the passed handle* if all is good

*Handles are always a pointer here

Ncryptsslp Validation function Symbols

Command - Dump \\vmware-host\Shared Folders\Documents\Challenges\zzTm...

```
0:000> uf ncryptsslp!SslpValidateMasterKeyHandle
ncryptsslp!SslpValidateMasterKeyHandle:
00007fff`df75b5b8 4885c9          test    rcx,rcx
00007fff`df75b5bb 7412            je      ncryptsslp!SslpValidateMasterKeyHandle+0x17

ncryptsslp!SslpValidateMasterKeyHandle+0x5:
00007fff`df75b5bd 833950          cmp     dword ptr [rcx],50h
00007fff`df75b5c0 720d            jb      ncryptsslp!SslpValidateMasterKeyHandle+0x17

ncryptsslp!SslpValidateMasterKeyHandle+0xa:
00007fff`df75b5c2 817904356c7373  cmp     dword ptr [rcx+4],73736C35h
00007fff`df75b5c9 7504            jne     ncryptsslp!SslpValidateMasterKeyHandle+0x17

ncryptsslp!SslpValidateMasterKeyHandle+0x13:
00007fff`df75b5cb 488bc1          mov     rax,rcx
00007fff`df75b5ce c3              ret

ncryptsslp!SslpValidateMasterKeyHandle+0x17:
00007fff`df75b5cf 33c0            xor     eax,eax
00007fff`df75b5d1 c3              ret

0:000>
```

The Ncrypt SSL Provider (ncryptsslp.dll)

SSL Magic	Size (x86)	Size (x64)	Validation Functions
ssl1	0xE4	0x130	SslpValidateProvHandle
ssl2	0x24	0x30	SslpValidateHashHandle
ssl3	?	?	<none>
ssl4	0x18	0x20	SslpValidate**KeyPair**Handle
ssl5	0x48	0x50	SslpValidate**MasterKey**Handle
ssl6	0x18	0x20	SslpValidate**Ephemeral**Handle
ssl7	?	?	<none>

ssl3 was already discussed, appears in the following functions:

```
TlsGenerateSessionKeys+0x251
SPSslDecryptPacket+0x43
SPSslEncryptPacket+0x43
SPSslImportKey+0x19a
SPSslExportKey+0x76
Ssl2GenerateSessionKeys+0x22c
```

Pre-Master Secret (PMS)

- The ‘**ssl7**’ struct appears to be used specifically for the RSA PMS
- As advised by the RFC, it gets destroyed quickly, once the Master Secret (MS) has been derived
- Client generates random data, populates the ssl7 structure, and encrypts
- In ECC the PMS is x-coordinate of the shared secret derived (which is a point on the curve), so this doesn’t /seem/ to get used in that case

Functions where ssl7 appears:

```
ncryptsslp!SPSslGenerateMasterKey+0x75
ncryptsslp!SPSslGenerateMasterKey+0x5595
ncryptsslp!SPSslGeneratePreMasterKey+0x15e
ncryptsslp!TlsDecryptMasterKey+0x6b
```

Bottom line:

> **It’s vestigial for our purposes - it doesn’t do anything another secret can’t**

Master Secret

- Basically the Holy Grail for a given connection
  - It always exists
  - It's what gets cached and used to derive the session keys
- Structure for storage is simple - secret is unencrypted (as you'd expect)
- This + **Unique ID** = decryption, natively in tools like wireshark

So...how do we get there?

_SSL_MASTER_SECRET	
4	cbStructLength
4	dwMagic [“ssl5”]
4	dwProtocolVersion
0/4	dwUnknown1* [alignment?]
4/8	pCipherSuiteListEntry
4	bIsClientCache
48	rgbMasterSecret
4	dwUnknown2 [reserved?]

Master Secret

_SSL_MASTER_SECRET	
4	cbStructLength
4	dwMagic [“ssl5”]
4	dwProtocolVersion
0/4	dwUnknown1* [alignment?]
4/8	pCipherSuiteListEntry
4	bIsClientCache
48	rgbMasterSecret
4	dwUnknown2 [reserved?]

Command - Dump \\vmware-host\Shared Folders\Documents\Challenges\zzTmp\...

```
0:000> .foreach(ms {s -[1]d 0 L?80000000000000 'ssl5'}){.echo ***Raw Master Secret***
***Raw Master Secret***
000000c9`86d9e980  50 00 00 00 35 6c 73 73-03 03 00 00 00 00 00 00  P...5lss........
000000c9`86d9e990  10 1a 76 df ff 7f 00 00-00 00 00 00 01 7b 37 90  ..v..........{7.
000000c9`86d9e9a0  22 83 5c 78 ca 16 f7 12-53 00 39 fb 5d ea dd 03  ".\x....S.9.]...
000000c9`86d9e9b0  a7 73 8a ba c7 a5 92 67-b7 45 97 2c 01 a3 25 15  .s.....g.E.,..%.
000000c9`86d9e9c0  44 0d fa d7 4c 45 c1 9a-25 a6 51 f1 00 00 00 00  D...LE..%.Q.....

***Parsed Master Secret***
000000c9`86d9e980  00000050
000000c9`86d9e984  73736c35                                        5lss
000000c9`86d9e988  00000000`00000303
000000c9`86d9e990  00007fff`df761a10 ncryptsslp!CipherSuiteList+0x1500
000000c9`86d9e998  00000000
* Secret:
000000c9`86d9e99c  01 7b 37 90 22 83 5c 78-ca 16 f7 12 53 00 39 fb  .{7.".\x....S.9.
000000c9`86d9e9ac  5d ea dd 03 a7 73 8a ba-c7 a5 92 67 b7 45 97 2c  ]....s.....g.E.,
000000c9`86d9e9bc  01 a3 25 15 44 0d fa d7-4c 45 c1 9a 25 a6 51 f1  ..%.D...LE..%.Q.
*
000000c9`86d9e9cc  00000000

***Raw Master Secret***
000000c9`86d9f080  50 00 00 00 35 6c 73 73-03 03 00 00 00 00 00 00  P...5lss........
000000c9`86d9f090  10 1a 76 df ff 7f 00 00-00 00 00 c3 05 82 ff  ..v.............
000000c9`86d9f0a0  a5 6d ea 9c 0a ed 59 42-33 69 d8 ef b1 8a 79 6c  .m....YB3i....yl
000000c9`86d9f0b0  60 5e 46 1e 7b 45 d0 12-88 71 14 c0 0b 32 86 ab  `^F.{E...q...2..
000000c9`86d9f0c0  4a b4 84 1a fa 12 7a f0-3f 24 6b cb 00 00 00 00  J.....z.?$k.....

0:000> .foreach(ms {s -[1]d 0 L?80000000000000 'ssl5'}){.echo ***Raw Master
Secret***;db ${ms}-4 Ldwo(${ms}-4);.echo;.echo ***Parsed Master Secret***;dd
${ms}-4 L1;dc ms L1;dp ${ms}+4 L1;dps ${ms}+4+$ptrsize L1;dd ${ms}+4+2*$ptrsize
L1;.echo * Secret:;db ${ms}+3*$ptrsize L30;.echo *;dd ${ms}-4+dwo(${ms}-4)-4
L1;.echo}
```

Command - Dump \\vmware-host\Shared Folders\Documents\Challenges\zzTmp...

```
0:000> r @$t0 = 000000c9`86d9e980;dc @$t0 L2;dp @$t0+10 L1;.echo *;dpu poi(@$t0+10) L2
000000c9`86d9e980  00000050 73736c35                       P...5lss
000000c9`86d9e990  00007fff`df761a10
*
00007fff`df761a10  0000c030`00000c00
00007fff`df761a18  00007fff`df762010 "TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384"

0:000>
```

Master Secret Mapped to Unique Identifier

- The Master Key is linked back to a unique ID through an "**NcryptSslKey**"
- The NcryptSslKey is referenced by an "**SessionCacheItem**"
- The SessionCacheItem contains either the SessionID, or a pointer and length value for a SessionTicket
  - Instantiated as either client or server item

_SESSION_CACHE_CLIENT_ITEM	
4/8	pVftable
…	…
@0x10	pMasterKey
…	…
@0x88	rgbSessionID[0x20]
…	…
@0x128	pSessionTicket
@0x130	cbSessionTicketLength

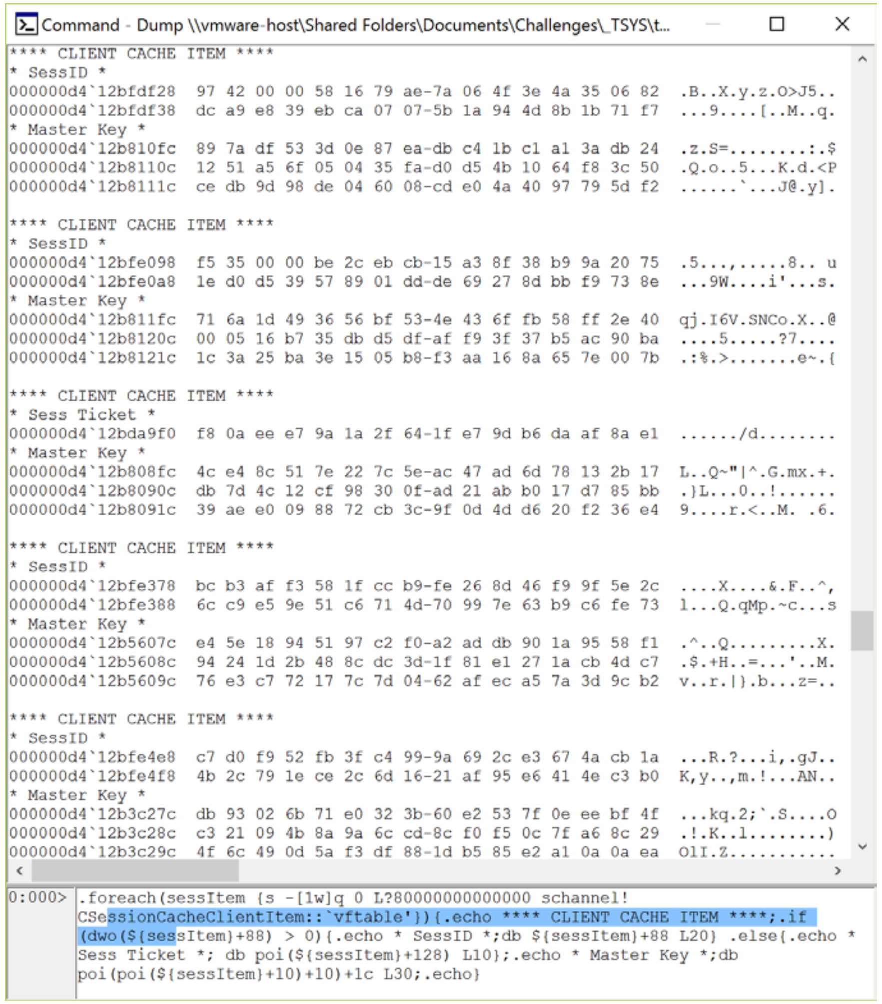
_SSL_MASTER_SECRET	
4	cbStructLength
4	dwMagic ["ssl5"]
4	dwProtocolVersion
0/4	dwUnknown1* [alignment?]
4/8	pCipherSuiteListEntry
4	bIsClientCache
48	rgbMasterSecret
4	dwUnknown2 [reserved?]

_NCRYPT_SSL_KEY_HANDLE	
4	cbStructLength
4	dwMagic ["BDDD"]
4/8	pNcryptSslProvider
4/8	pNcryptSslKey

At this point, we can find cache items, and extract the Master Secret + Unique ID

… Houston, we has plaintext.

Master Secret Mapped to Unique Identifier



Wireshark SSL Log Format

RSA Session-ID:97420000581679ae7a064f3e4a350682dca9e839ebca07075b1a944d8b1b71f7 **Master-Key:**897adf533d0e87eadbc41bc1a13adb241251a56f050435fad0d54b1064f83c50cedb9d98de046008cde04a4097795df2

RSA Session-ID:f5350000be2cebcb15a38f38b99a20751ed0d539578901ddde69278dbbf9738e **Master-Key:**716a1d493656bf534e436ffb58ff2e40000516b735dbd5dfaff93f37b5ac90ba1c3a25ba3e1505b8f3aa168a657e007b

RSA Session-ID:bcb3aff3581fccb9fe268d46f99f5e2c6cc9e59e51c6714d70997e63b9c6fe73 **Master-Key:**e45e18945197c2f0a2addb901a9558f194241d2b488cdc3d1f81e1271acb4dc776e3c772177c7d0462afeca57a3d9cb2

RSA Session-ID:c7d0f952fb3fc4999a692ce3674acb1a4b2c791ece2c6d1621af95e6414ec3b0 **Master-Key:**db93026b71e0323b60e2537f0eeebf4fc321094b8a9a6ccd8cf0f50c7fa68c294f6c490d5af3df881db585e2a10a0aea

Wireshark SSL input formats found here: https://github.com/boundary/wireshark/blob/master/epan/dissectors/packet-ssl.c

Ephemeral & Persistent Private Keys

- ❑ Both share the same structure
- ❑ Both store secrets in a Key Storage Provider Key struct (KPSK)
- ❑ The "Key Type" is compared with different values
  - ❑ ssl6 gets compared with a list stored in bcryptprimitives
  - ❑ ssl4 gets compared with a list stored in NCRYPTPROV
- ❑ The Key Storage Provider Key (KPSK) is referenced indirectly through an "Ncrypt Key" struct*

_SSL_KEY_PAIR	
4	cbStructLength
4	dwMagic ["ssl4" | "ssl6"]
4	dwKeyType
4	dwUnknown1 [alignment?]
4/8	pKspProvider
4/8	pKspKey

_NCRYPT_KEY_HANDLE	
4	cbStructLength
4	dwMagic [0x44440002]
4	dwKeyType
4	dwUnknown1 [alignment?]
4/8	pKspProvider
4/8	pKspKey

_KSP_KEY	
4	cbStructLength
4	dwMagic ["KSPK"]
4	dwKeyType
...	...
@0x60	pMSKY
@0xD0	pDpapiBlob
@0xD8	dwDpapiBlobLength

*NcryptKey not to be confused with NcryptSslKey

Ephemeral Private Key

- **For performance, reused across connections**
  - Given the public connection params, we can derive the PMS and subsequently MS
- **Stored unencrypted in a LE byte array**
  - Inside of MSKY struct
- The curve parameters are stored in the KPSK
  - Other parameters (A&B, etc) are stored in MSKY w/ the key
- Verified by generating the Public & comparing
  - The Public Key is also stored in the first pointer of the **`CEphemData`** struct that points to “**`ssl6`**”

Command - Dump \\vmware-host\Shared Folders\Documents\Challenges\zzTm...

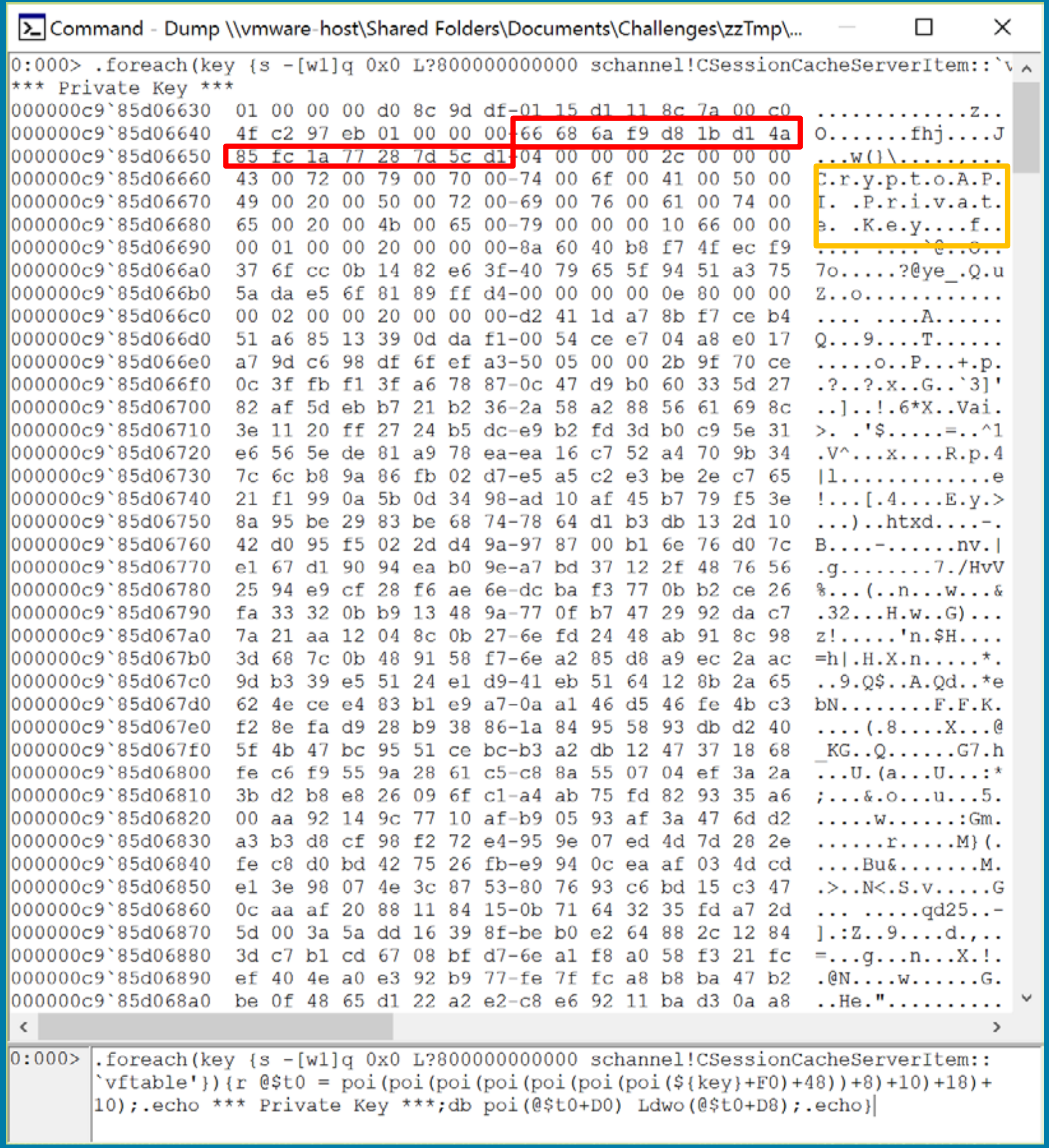
```
0:000> .foreach(ek {s -[1w]q 0 L?800000000000 schannel!CEphemKeyData::`vftable'}){.ech
*****
000000c9`86d2a000  00007fff`ed23b140 schannel!CEphemKeyData::`vftable'
* Public *
000000c9`86d9e688  f2 55 0c 8b e6 02 51 3b-98 e5 8a 7c 4a 8f 4b 1a  .U....Q;...|J.K.
000000c9`86d9e698  f8 b8 fd bd 6d d1 8c 85-78 f6 51 2a 8e 7a 5a 62  ....m...x.Q*.zZb
000000c9`86d9e6a8  fd 34 cc 5b 75 c4 bf ca-3c 6b 0e b8 84 b7 32 35  .4.[u...<k....25
000000c9`86d9e6b8  da 94 bb 1d 0c cc d0 00-8d 8b b7 2e 7a a4 79 27  ............z.y'
* Curve *
000000c9`86d22fb0  000000c9`86d5fa00 "nistP256"
** G(x)
000000c9`8698e71c  6b 17 d1 f2 e1 2c 42 47-f8 bc e6 e5 63 a4 40 f2  k....,BG....c.@.
000000c9`8698e72c  77 03 7d 81 2d eb 33 a0-f4 a1 39 45 d8 98 c2 96  w.}.-.3...9E....
** G(y)
000000c9`8698e71c  6b 17 d1 f2 e1 2c 42 47-f8 bc e6 e5 63 a4 40 f2  k....,BG....c.@.
000000c9`8698e72c  77 03 7d 81 2d eb 33 a0-f4 a1 39 45 d8 98 c2 96  w.}.-.3...9E....
* Private [LE!] *
000000c9`86d5fc80  d0 27 df dd c2 c6 78 5e-be 78 7e 70 94 c2 b7 1a  .'....x^.x~p....
000000c9`86d5fc90  b9 90 50 67 72 0d ce f7-63 45 e4 da 90 0c 59 78  ..Pgr...cE....Yx

*****
000000c9`86d2cee0  00007fff`ed23b140 schannel!CEphemKeyData::`vftable'
* Public *
000000c9`86d9ff88  d6 35 6b cc 81 66 c0 6e-74 86 f8 36 8c be c7 42  .5k..f.nt..6...B
000000c9`86d9ff98  00 3e 6b 34 49 7f 95 1a-d5 e3 1f 6e 72 75 aa 77  .>k4I......nru.w
000000c9`86d9ffa8  96 6d 93 0d 13 40 d3 f0-cd f6 ac 30 16 01 7a 44  .m...@.....0..zD
000000c9`86d9ffb8  2b ca 16 ea 9d fa 00 5d-63 f2 08 77 3d 23 1a 5d  +......]c..w=#.]
* Curve *
000000c9`86d231f0  000000c9`86d60540 "nistP256"
** G(x)
000000c9`8698e93c  6b 17 d1 f2 e1 2c 42 47-f8 bc e6 e5 63 a4 40 f2  k....,BG....c.@.
000000c9`8698e94c  77 03 7d 81 2d eb 33 a0-f4 a1 39 45 d8 98 c2 96  w.}.-.3...9E....
** G(y)
000000c9`8698e93c  6b 17 d1 f2 e1 2c 42 47-f8 bc e6 e5 63 a4 40 f2  k....,BG....c.@.
000000c9`8698e94c  77 03 7d 81 2d eb 33 a0-f4 a1 39 45 d8 98 c2 96  w.}.-.3...9E....
* Private [LE!] *
000000c9`86d602c0  cd 94 db f9 f0 68 b9 6e-05 8f b6 7c fa d0 8c 9e  .....h.n...|....
000000c9`86d602d0  da 78 97 2c 01 24 42 32-0b 8f 35 b5 4a 4f 7d 9e  .x.,.$B2..5.JO}.
```

```
0:000> .foreach(ek {s -[1w]q 0 L?800000000000 schannel!CEphemKeyData::`vftable'})
{.echo *****;dps ek L1;.echo * Public *;db poi(${ek}+10)+8 L40;r @$t0 =
poi(poi(poi(poi(${ek}+8)+10)+18)+10);.echo * Curve *; dpu @$t0+1f0 L1;.echo **
G(x);db poi(@$t0+1f8)+7c L20;.echo ** G(y);db poi(@$t0+1f8)+7c L20;.echo *
Private [LE!] *;db poi(poi(poi(@$t0+60)+10)+18) L20;.echo}
```

In-line with suggestion of this paper: http://dualec.org/DualECTLS.pdf

"Persistent" Private Key

- The RSA Key that is stored on disk
  - Unique instance for each private RSA Key – by default, the system has several
  - E.g. one for Terminal Services
- RSA Keys are DPAPI protected
  - Lots of research about protection / exporting
  - Note the MK GUID highlighted from the Blob
- The Key is linked to a given Server Cache Item
- Verified by comparing the DPAPI blob in memory to protected certificate on disk
  - Also verified through decryption



Decrypting Persistent Key - DPAPI

- Can extract the blob from memory and decrypt w/ keys from disk
  - DPAPIck / Mimikatz

OR

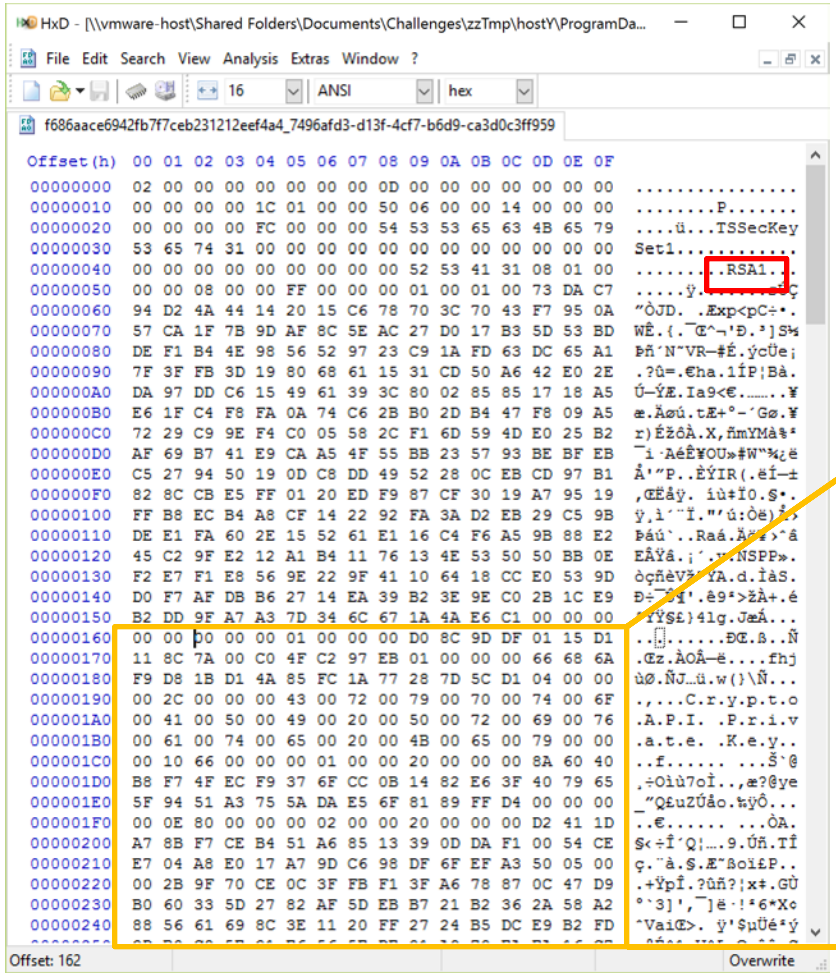
- Can decrypt directly from memory :D
  - MasterKeys get cached in Memory
    - On Win10 in: dpapisrv!g_MasterKeyCacheList
    - See Mimilib for further details
    - Even though symbols are sort of required, we could likely do without them
      - There are only two Bcrypt key pointers in lsasrv's .rdata section (plus one lock)
      - Identifying the IV is more challenging

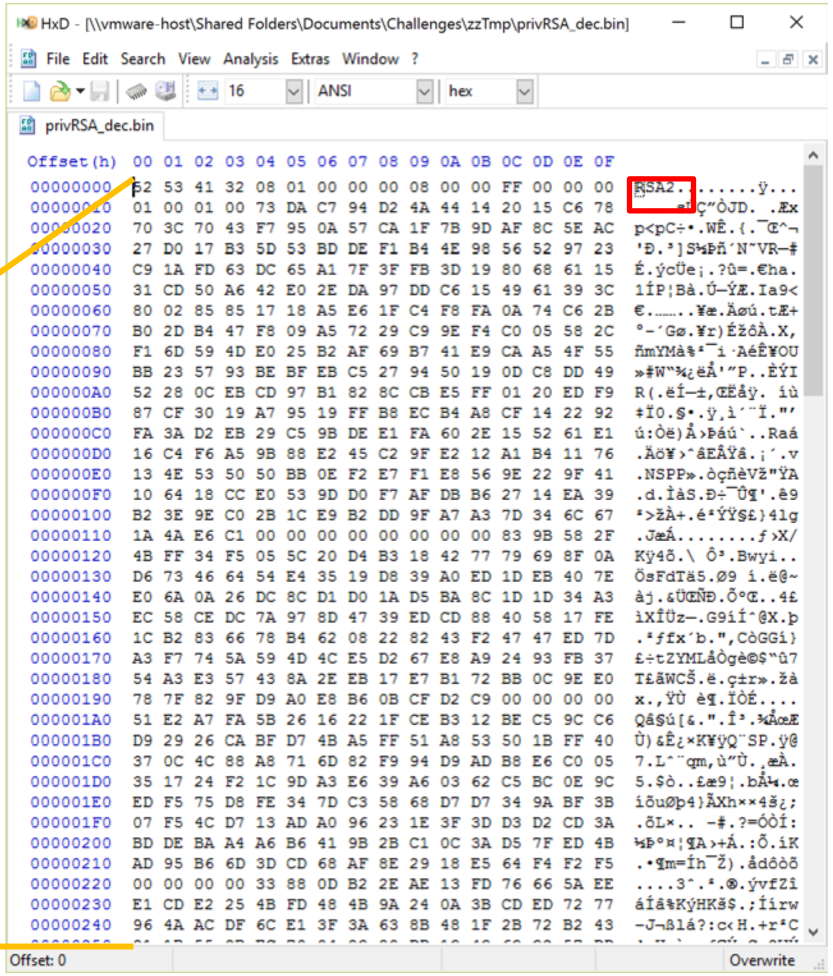
Command - Dump \\vmware-host\Shared Folders\Documents\Challenges\zzT...

```
0:000> db lsasrv!InitializationVector L8;.echo
00007fff`edaec528  60 4a dd 90 31 45 80 97                          `J..1E..

0:000> r @$t0 = poi(poi(lsasrv!h3DesKey)+10); db @$t0+3c Ldwo(@$t0+38);.echo
000000c9`85fb005c  3a 6a dd 76 0f 30 0b 7c-3d 43 f4 db e4 12 66 3a  :j.v.0.|=C....f:
000000c9`85fb006c  8c f2 b1 f0 a0 a5 5c e4                          ......\.

0:000> !list -x ".if(dwo(@$extret+18) ==  f96a6866){dq @$extret L2;dq @$extret+10 L1;



000000c9`85d6a3b0  000000c9`85d5cd10 000000c9`85dbbdc0
000000c9`85d6a3c0  00000000`000003e7
000000c9`85d6a3c8  f96a6866 4ad11bd8 771afc85 d15c7d28
000000c9`85d6a3d8  01d1cce0`db1e2d0c
000000c9`85d6a3e0  00000040
000000c9`85d6a3e4  e5 ce 98 a2 ff 50 0d 78-4a 8e 17 54 8f f1 fe b7  .....P.xJ..T....
000000c9`85d6a3f4  21 7e 17 70 2f 7d 52 aa-db cd 75 93 25 48 7a c2  !~.p/}R...u.%Hz.
000000c9`85d6a404  70 62 1a d9 5a e8 97 19-64 c6 62 a0 2a 75 66 9e  pb..Z...d.b.*uf.
000000c9`85d6a414  a5 c8 33 0c d7 af b6 b1-fb 38 ab 89 b7 d5 cf 14  ..3......8......
000000c9`85d6a424  00000000
000000c9`85d6a430  "C:\Windows\system32\Microsoft\Protect\S-1-5-18\"

0:000> !list -x ".if(dwo(@$extret+18) ==  f96a6866){dq @$extret L2;dq @$extret+10
L1;dd @$extret+18 L10/4;dq @$extret+28 L1;dd @$extret+30 L1;db @$extret+34
Ldwo(@$extret+30);dd @$extret+74 L1;du /c ff poi(@$extret+78)}" dpapisrv!
g_MasterKeyCacheList
```

Cached DPAPI MK + Params to Decrypt

Decrypting Persistent Key - DPAPI





Session Tickets

- Not seemingly in widespread use with IIS?
  - Comes around w/ Server 2012 R2
  - Documentation is lacking.
- Enabled via reg key + powershell cmdlets?
  - Creates an “Administrator managed” session ticket key
- Schannel functions related to Session Tickets load the keyfile from disk
- Export-TlsSessionTicketKey :D

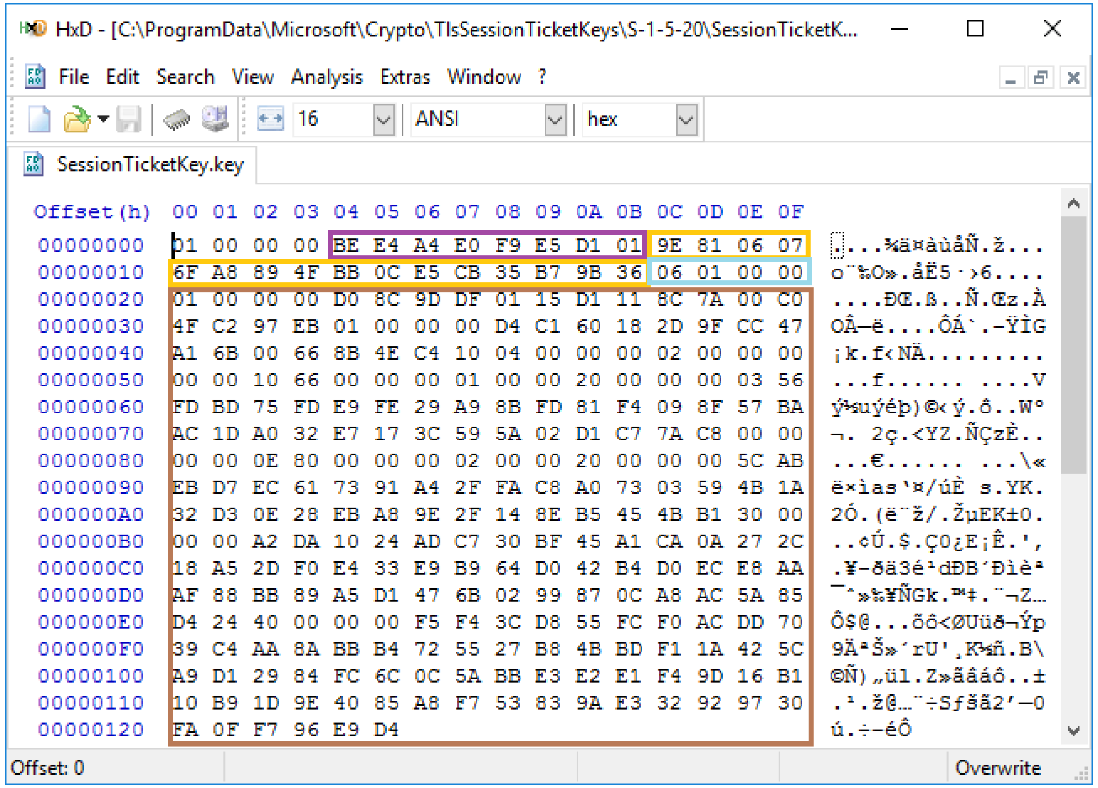
Administrator: C:\Windows\system32\cmd.exe - powershell

```
PS C:\> mkdir \KeyConfig


    Directory: C:\


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
d-----        7/24/2016   3:21 PM                KeyConfig


PS C:\> $Password = Read-Host -AsSecureString
*********
PS C:\> New-TlsSessionTicketKey -Password $Password -Path "C:\KeyConfig\TlsSessionTicketKey.config"
PS C:\> Enable-TlsSessionTicketKey -Password $Password -Path "C:\KeyConfig\TlsSessionTicketKey.config" -ServiceAccountNa
me "NetworkService"
PS C:\> Enable-TlsSessionTicketKey -Password $Password -Path "C:\KeyConfig\TlsSessionTicketKey.config" -ServiceAccountNa
me "System"
PS C:\> New-ItemProperty -Path HKLM:\System\CurrentControlSet\Control\SecurityProviders\SCHANNEL -Name EnableSessionTick
et -Value 1 -PropertyType DWORD


EnableSessionTicket : 1
PSPath              : Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\SecurityP
                      roviders\SCHANNEL
PSParentPath        : Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\SecurityP
                      roviders
PSChildName         : SCHANNEL
PSDrive             : HKLM
PSProvider          : Microsoft.PowerShell.Core\Registry
```

Reference to DISABLING session tickets in Win8.1 Preview release notes: https://technet.microsoft.com/en-us/library/dn303404.aspx

Session Ticket Key

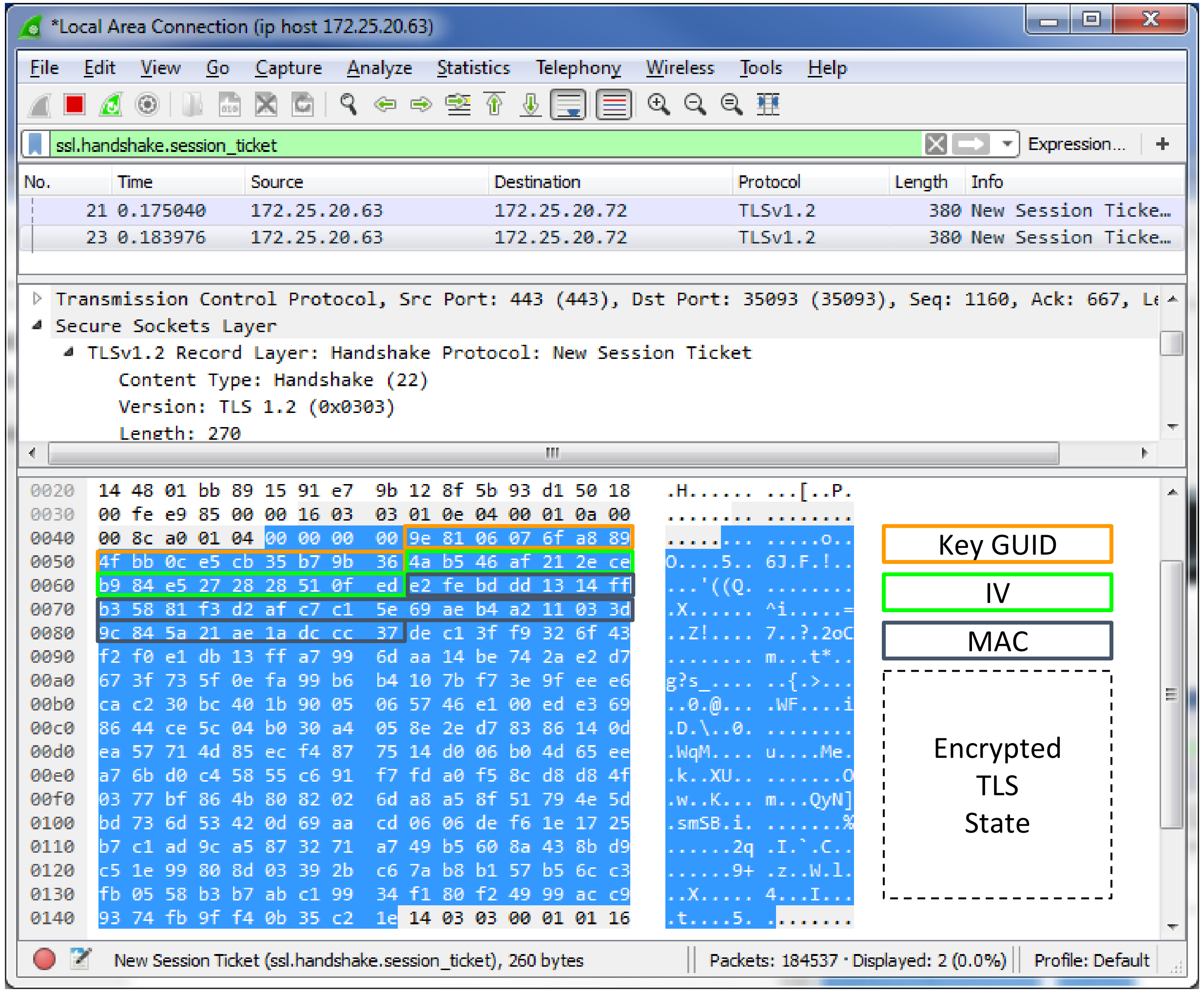
- Keyfile contains a DPAPI blob, preceded by a SessionTicketKey GUID + 8 byte value
- Key gets loaded via schannel
  - The heavy lifting (at least in Win10) is done via mskeyprotect
- AES key derived from decrypted blob via BCryptKeyDerivation()
- Key gets cached inside mskeyprotect!
  - No symbols for cache : /
  - No bother, we can just find the Key GUID that's cached with it :D



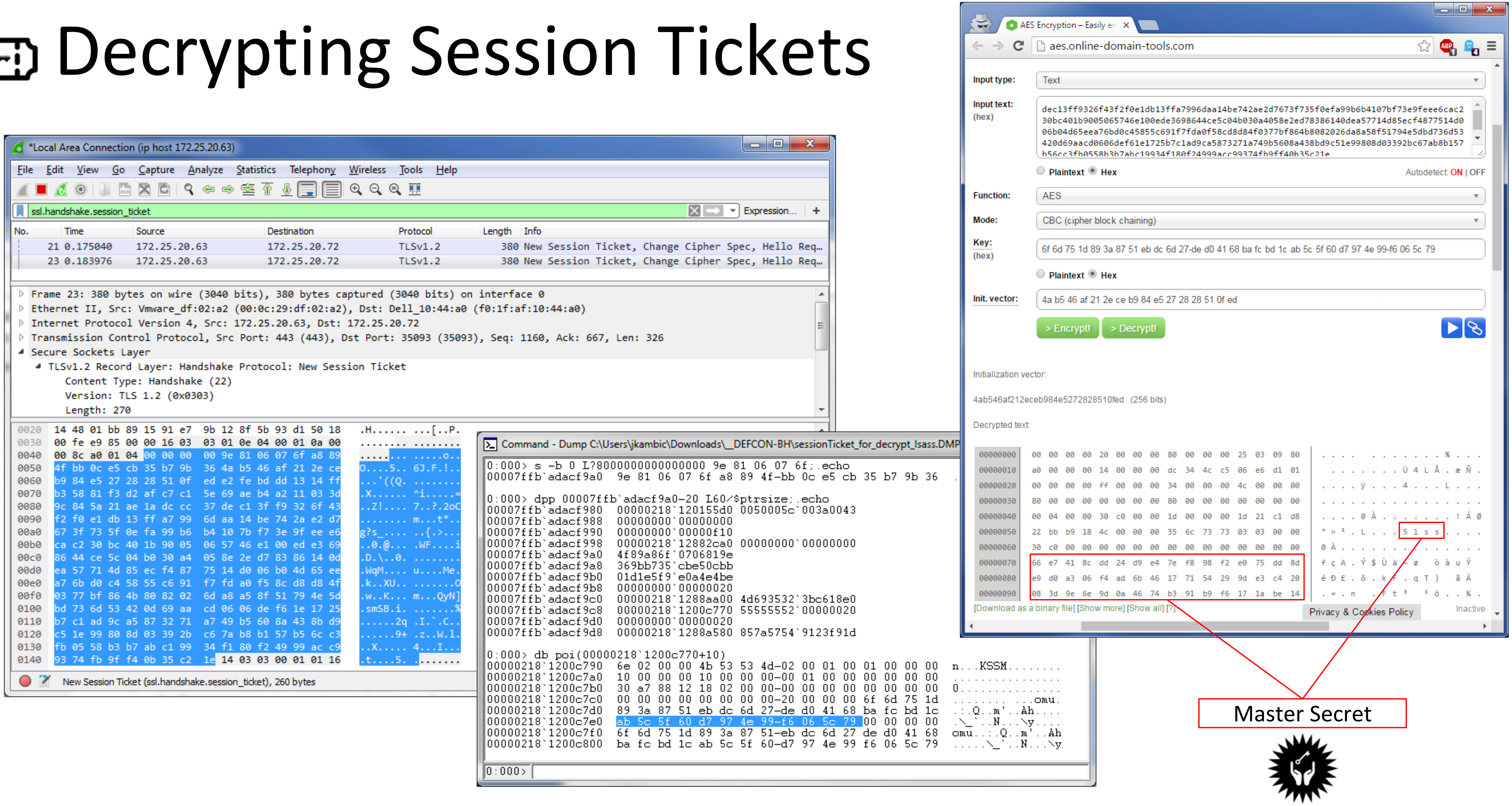
- Possibly Salt or MAC?
- Session Ticket Key GUID
- Size of ensuing DPAPI Blob
- DPAPI Blob (contains it's own fields)

Decrypting Session Tickets

- Session Ticket structure pretty much follows the RFC (5077), except:
  - MAC & Encrypted State are flipped (makes a lot of sense)
- After extracting/deriving the Symm key, it's just straight AES 256
- Contents of the State are what you'd expect:
  - Timestamp
  - Protocol/Ciphersuite info
  - MS struct



Decrypting Session Tickets



Secrets are cool and all...

But Jake, what if I don't have a packet capture?
(And I don't care about future connections?)

The Context

Inherent Metadata TLS Provides

Core SSL/TLS functionality

- Timestamps
  - The random values *typically* start with a 4-byte timestamp (if you play by the RFCs)
- Identity / fingerprinting
  - Public Key
  - Session ID*
  - Offered Cipher Suites / Extensions
- Session ID's are arbitrary, but are not always random -> Schannel is a perfect example
  - uses `MaximumCacheEntries` parameter when creating the first dword of the random, leading to a(n imperfect) fingerprint of two zero bytes in 3/4th byte*

TLS Extensions

- Server Name Indication (SNI)
  - Virtual hosts
- Application-Layer Protocol Negotiation (ALPN)
  - Limited, but what protocol comes next
    - fingerprinting?
- Session Tickets
  - Key GUID

*Referenced in this paper: http://dualec.org/DualECTLS.pdf

Schannel Caching Parameters

Parameters:

- The following control upper-limit of cache time:
  `m_dwClientLifespan`
  `m_dwServerLifespan`
  `m_dwSessionTicketLifespan`
- All of which:
  are set to **`0x02255100`** (**10hrs** in ms)
- Also of Interest:
  `m_dwMaximumEntries` (set to **`0x4e20`** or **20,000 entries** by default)

  `m_dwEnableSessionTicket` controls use of session tickets (e.g. **`0, 1, 2`**)

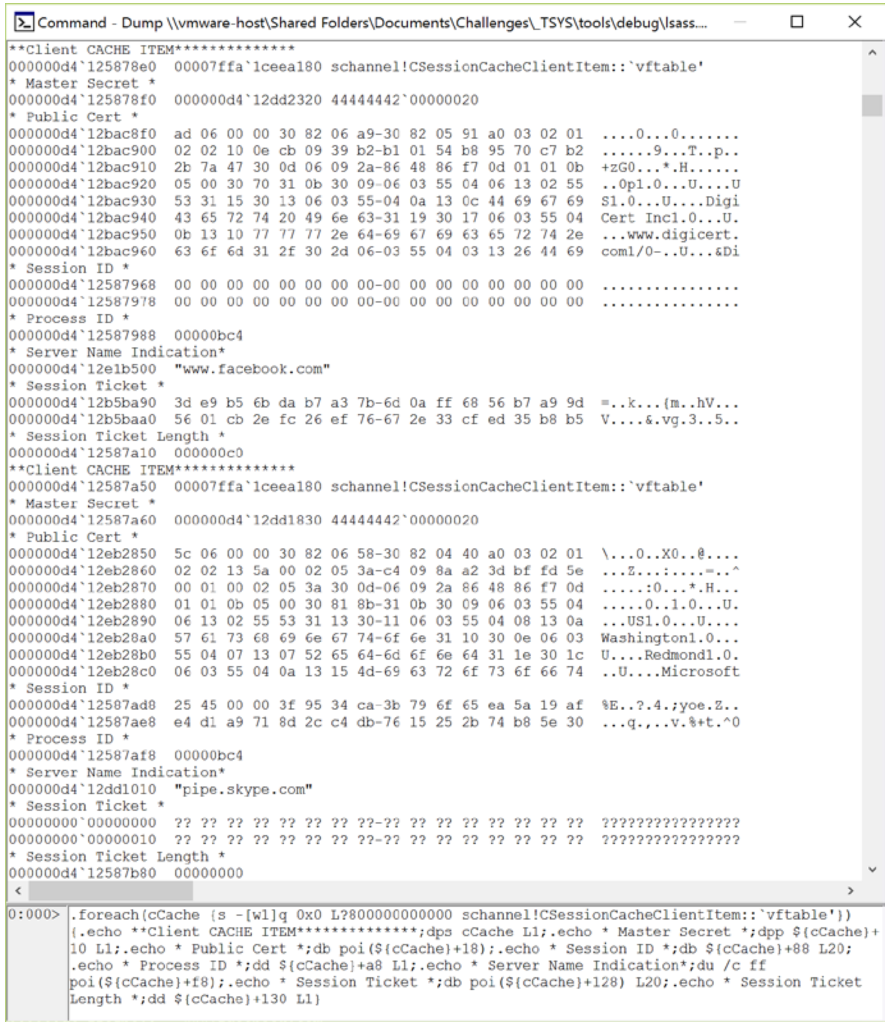
  `m_dwSessionCleanupIntervalInSeconds` (set to **`0x012c`** or **300 seconds** by default)

HOWEVER:

- Schannel is the library, the ***process*** has control
- Proc can purge its own cache at will
  - For example, IIS reportedly* purges after around two hours
- Schannel maintains track of process, frees cache items after client proc terminates : <
  - Haven't looked at the exact mechanism
  - As you'll see, the upside is that the Process ID is stored in the Cache

This is your Schannel Cache (x64)

```
'_SSL_SESSION_CACHE_CLIENT_ITEM': [ 0x148, {
     'Vftable': [0x0, ['pointer64', ['void']]],
     ‘MasterKey': [0x10, ['pointer64', ['void']]],
     'PublicCertificate': [0x18, ['pointer64', ['void']]],
     'PublicKey': [0x28, ['pointer64', ['void']]],
     'NcryptSslProv': [0x60, ['pointer64', ['void']]],
     'SessionIdLen': [0x86, ['short short']],
     'SessionId': [0x88, ['array', 0x20, ['unsigned char']]],
     'ProcessId': [0xa8, ['unsigned long']],
     'MaxLifeTime': [0xB0, ['unsigned long']],
     'CertSerializedCertificateChain': [0xB0, ['pointer64', ['void']]],
     'UnkList1Flink': [0xB8, ['pointer64', ['void']]],
     'UnkList1Blink': [0xC0, ['pointer64', ['void']]],
     'UnkCacheList2Flink': [0xC8, ['pointer64', ['void']]],
     'UnkCacheList2Blink': [0xD0, ['pointer64', ['void']]],
     'ServerName': [0x108, ['pointer64', ['void']]],
     ‘LogonSessionUID': [0x110, ['pointer64', ['void']]],
     'CSessCacheManager': [0x120, ['pointer64', ['void']]],
     'SessionTicket': [0x138, ['pointer64', ['void']]],
     'SessionTicketLen': [0x140, ['int']],
     }],
```



This is your Schannel Cache (x64)

```
'_SSL_SESSION_CACHE_SERVER_ITEM': [ 0x110, {
    'Vftable': [0x0, ['pointer64', ['void']]],
    'NcryptKey': [0x10, ['pointer64', ['void']]],
    'NcryptSslProv': [0x60, ['pointer64', ['void']]],
    'SessionId': [0x88, ['array', 0x20, ['unsigned char']]],
    'ProcessId': [0xa8, ['unsigned long']],
    'MaxLifeTime': [0xB0, ['unsigned long']],
    'LastError?': [0xE8, ['unsigned long']],
    'CSslCredential': [0xF0, ['pointer64', ['void']]],
    }],
```

Command - Dump \\vmware-host\Shared Folders\Documents\Challenges\zzTmp\hostY\lsa...

```
0:000> .foreach(sCache {s -[wl]q 0x0 L?800000000000 schannel!CSessionCacheServerItem::`vftal
**SERVER CACHE ITEM**************
000000c9`86d2f1c0  00007fff`ed239f90 schannel!CSessionCacheServerItem::`vftable'
* Master Secret *
000000c9`86d2f1d0  000000c9`86d5f870 44444442`00000020
* Session ID *
000000c9`86d2f248  5e 06 00 00 30 4b 18 77-32 15 9c 6a 53 56 75 b6  ^...0K.w2..jSVu.
000000c9`86d2f258  64 6c 63 c5 dd c4 53 3a-8d 0b 42 f5 b6 ef c9 76  dlc...S:..B....v
* Process ID *
000000c9`86d2f268  000003b0
* Server Key *
000000c9`86d2f2b0  000000c9`85dbb040 00007fff`ed23b118 schannel!CSslCredential::`vftable'

**SERVER CACHE ITEM**************
000000c9`86d2f6c0  00007fff`ed239f90 schannel!CSessionCacheServerItem::`vftable'
* Master Secret *
000000c9`86d2f6d0  000000c9`86d60f40 44444442`00000020
* Session ID *
000000c9`86d2f748  0c 11 00 00 7c 33 7f b1-1d 9b 99 60 2d 39 4d 53  ....|3.....`-9MS
000000c9`86d2f758  3a 06 00 e9 9f b8 77 03-9f 82 c1 be ba 09 42 4a  :.....w.......BJ
* Process ID *
000000c9`86d2f768  000003b0
* Server Key *
000000c9`86d2f7b0  000000c9`85d5e8e0 00007fff`ed23b118 schannel!CSslCredential::`vftable'

**SERVER CACHE ITEM**************
000000c9`86d2f800  00007fff`ed239f90 schannel!CSessionCacheServerItem::`vftable'
* Master Secret *
000000c9`86d2f810  000000c9`86d60270 44444442`00000020
* Session ID *
000000c9`86d2f888  a3 40 00 00 08 89 68 b2-a6 06 0f c0 52 b5 eb c4  .@....h.....R...
000000c9`86d2f898  58 1b 4b f2 61 47 23 80-16 db 64 7d 34 ef a7 1a  X.K.aG#...d}4...
* Process ID *
000000c9`86d2f8a8  000003b0
* Server Key *
000000c9`86d2f8f0  000000c9`85dbb040 00007fff`ed23b118 schannel!CSslCredential::`vftable'

**SERVER CACHE ITEM**************
000000c9`86d2fa80  00007fff`ed239f90 schannel!CSessionCacheServerItem::`vftable'
* Master Secret *
000000c9`86d2fa90  000000c9`86d60c20 44444442`00000020
* Session ID *
000000c9`86d2fb08  0b 2d 00 00 ab 80 7a 38-e4 9c 64 5c 42 2e 74 d2  .-....z8..d\B.t.
000000c9`86d2fb18  a2 d8 c3 b1 71 17 77 b8-48 f7 93 f8 16 ef 30 03  ....q.w.H.....0.
* Process ID *
```

```
0:000> .foreach(sCache {s -[wl]q 0x0 L?800000000000 schannel!CSessionCacheServerItem::
`vftable'}){.echo **SERVER CACHE ITEM**************;dps sCache L1;.echo * Master
Secret *;dpp ${sCache}+10 L1;.echo * Session ID *;db ${sCache}+88 L20; .echo * Process
ID *;dd ${sCache}+a8 L1;.echo * Server Key *;dpp ${sCache}+f0 L1;.echo }
```

This is your Schannel Cache on ~~Drugs~~ Vista

```
'_SSL_SESSION_CACHE_CLIENT_ITEM': [ 0xf0, {
    'Flink': [0x0, ['pointer', ['void']]],
    'Blink': [0x4, ['pointer', ['void']]],
    'ProcessId': [0x8, [['unsigned long']],
    'MasterKey': [0x14, ['pointer', ['NcryptSslKey']]],
    'CipherSuiteId': [0x1C, ['pointer', ['void']]],
    'ECCurveParam': [0x20, ['pointer', ['void']]],
    'NcryptSslProv': [0x28, ['pointer', ['void']]],
    'PublicCertificate': [0x2C, ['pointer', ['void']]],
    'PublicCert2': [0x34, ['pointer', ['void']]],
    'PublicKeyStruct': [0x3C, ['pointer', ['void']]],
    'PublicCertStruct3': [0x44, ['pointer', ['void']]],
    'ServerName': [0x80, ['pointer', ['void']]],
    'SessionIdSize': [0x94, ['short short']],
    'SessionId': [0x98, ['array', 0x20, ['unsigned char']]],
    'ErrorCode': [0xEC, ['pointer64', ['void']]],
    }],
```

Command - Dump \\vmware-host\Shared Folders\Documents\Challenge...

```
*** Cache Item ***
* ProcId:
001d76f0  00000cf8
* NcrypSslKey:
001d76fc  001dab40 00000018
* SNI:
001d7768  01f9e480 "live.sysinternals.com"
* SessionID:
001d7780  59 19 00 00 07 4a 6c cc-d6 b0 e2 b2 5f cd d1 30  Y....Jl....._..0
001d7790  bf ee 06 b1 ec 20 e3 57-e3 79 52 72 d7 f5 a5 41  ..... .W.yRr...A

*** Cache Item ***
* ProcId:
001d7828  00000cf8
* NcrypSslKey:
001d7834  001dabe0 00000018
* SNI:
001d78a0  01fa3cb8 "www.torproject.org"
* SessionID:
001d78b8  ba ce 7b 7e ca 6d e8 15-92 e8 ae fb 08 bb 71 83  ..{~.m........q.
001d78c8  e7 87 ed 78 e5 12 f3 c0-24 a3 b6 0b e8 a2 43 b9  ...x....$.....C.

*** Cache Item ***
* ProcId:
001d7960  00000cf8
* NcrypSslKey:
001d796c  01fa3f98 00000018
* SNI:
001d79d8  01fa3d18 "urs.microsoft.com"
* SessionID:
001d79f0  99 0e 00 00 d8 3f de 02-53 c3 68 49 59 89 c2 c0  .....?..S.hIY...
001d7a00  71 ca bd 8f 5f 7b bd 59-08 6c df 44 8c a7 b7 7b  q..._{.Y.l.D...{

*** Cache Item ***
* ProcId:
001d7e40  00000cf8
* NcrypSslKey:
001d7e4c  01fa3ed8 00000018
* SNI:
001d7eb8  01f75d88 "login.live.com"
* SessionID:
001d7ed0  f6 07 00 00 5d 3d bc aa-f7 91 9a 5e f5 3e b7 10  ....]=.....^.>..
001d7ee0  ab dc 7c d1 1f 3a 0a 95-08 02 80 cc ee 92 4c d1  ..|..:........L.

*** Cache Item ***
* ProcId:
001d7bd0  00000cf8
* NcrypSslKey:

0:000> !list -x ".echo *** Cache Item ***;.echo * ProcId:;dd @$extret+8 L1;.echo
* NcrypSslKey:;dpp @$extret+14 L1;.echo * SNI:;dpu @$extret+80 L1;.echo *
SessionID:;db @$extret+98 L20" 001d8ba0
```

Automating it

Volatility / Rekall

- Plugins for both – by default (no args) they:
  - Find LSASS
  - Scan Writeable VADs / Heap for Master Key signature (Volatility) or directly for SessionCacheItems (Rekall)
  - Dump out the wireshark format shown earlier

- Hoping to have functional powershell module or maybe incorporation into mimikatz? (Benjamin Delphy is kinda the man for LSASS)

```
     % vol.py --plugins=./plugins --profile=Win10x64 -f ./Win10-Test-c2a4a77d.vm
em lsasslkey
Volatility Foundation Volatility Framework 2.5
RSA Session-ID:b93c0000a110690b4ae9111bce5725c6c47a037b3c39c49c75ce51e1c2eb79ee M
aster-Key:bc28467999b99fd3fdf3a24642c5d93b9ab43e51627f6e0145ef120ba98a1c3223f3dbe
0154e30d7869bdb7ab66f5318
RSA Session-ID:173300000f84a86aebb2c5de0af20e6d5c2cab95ab65043e14c6e19cee54ee17 M
aster-Key:9dd750e12e6e4439b08326d4a1f9eba2d2fe65c2a26c2088e7cec22ce1d91e9f219b704
547a2b2eccb9a81d557d5ae1a
RSA Session-ID:3c2c000024b8f70dd2613d8b13d0c4ac4daaefbe53ab4b7cb9763e80feccb4f1 M
aster-Key:2d119c64695ffc9c143c136471f5625d8cde92d35721f5f2849b92639603799a45e1e60
1786cbf89b00c186969d44983
RSA Session-ID:d4170000da09f8596739215e216c496568fa66e42ac32b974d440949dff33d2b M
aster-Key:44b503bef7842ea9a416fbf8b63b932b23b7b687fbf5297b253eac427877c8e11595e14
c3f00c40bf2a0f4688de0b7aa
RSA Session-ID:432a0000bf4f622f0fc119974a0ef30cd838c3a025b83abbdcdbce7b2325d2d9 M
aster-Key:552699d61e21d1b871af4b05a54003bf03eade60666dd1e54b94c3b5ec98f296db4ae99
baed4e23882175e5ffd88be31
RSA Session-ID:6f230000a021aac48d15544524c1454e4ec01d5adb305d8d9d57ab2b991dd597 M
aster-Key:8bc9e9df653e3cbf533be84c6897787bd453b8cee9d5389e9c3659ebf997d9c8d0666aa
dca5be2258f30b9251215a717
```

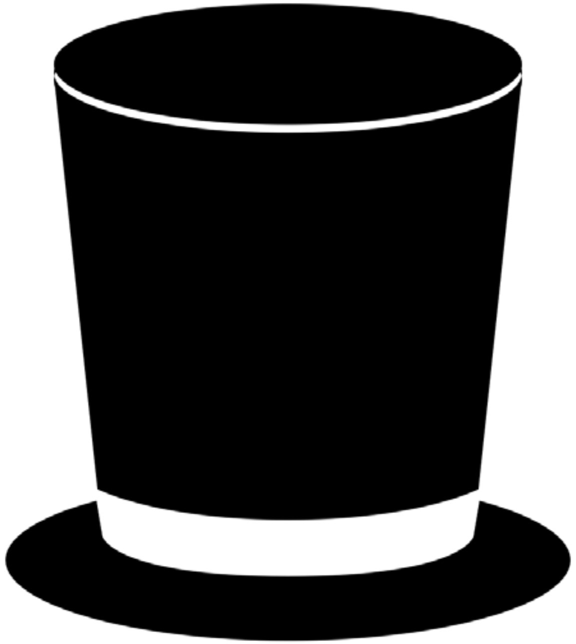
Limitations

- **We're working with internal, undocumented structures**
  - They change over time -- sometime around April 2016, an element appears to have been inserted in cache after the SessionID and before the SNI
    - Not a huge deal, except when differences amongst instances of same OS (e.g. ones that have and have not been updated)

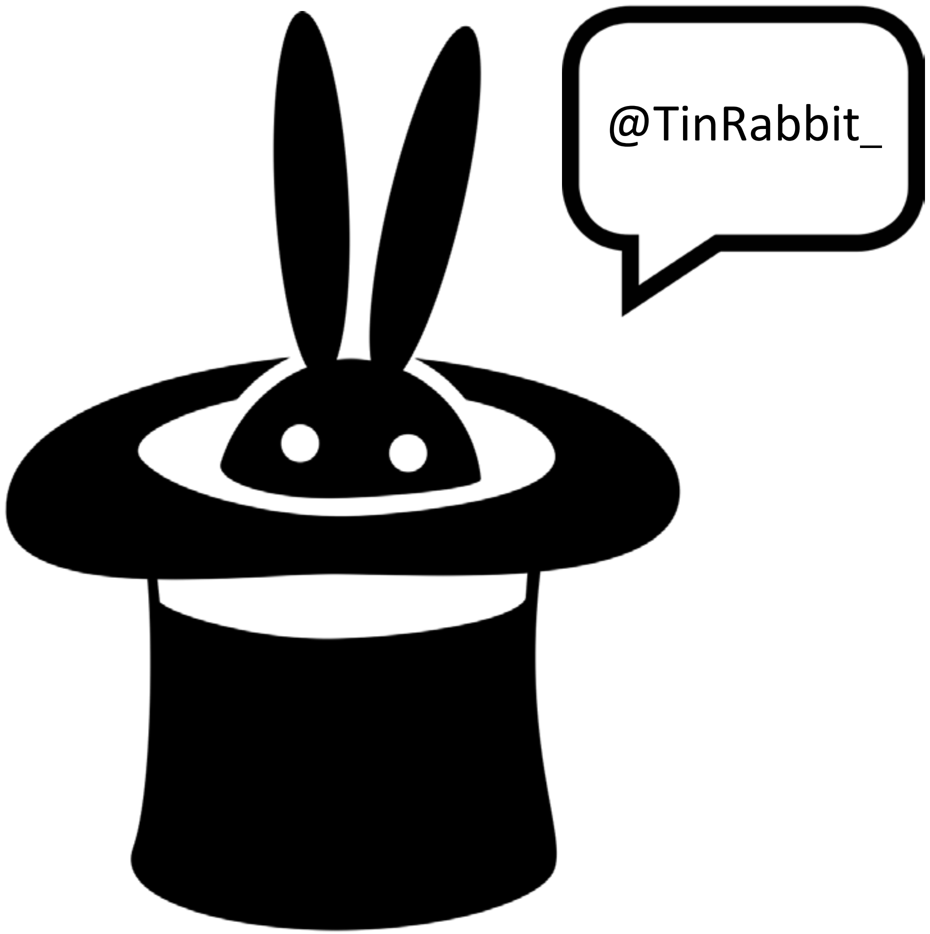
- **Relying on symbols for some of this**
  - MS giveth and can taketh away.
  - Still, can be done without them, just slightly less efficiently.

- **You need to be able to read LSASS memory**
  - Not a huge deal in 2016, but still merits mention -- you need to own the system
  - If you own the system, you can already do bad stuff (keylog / tap net interface)
  - This is why it's probably most useful in a forensic context

Demo



Fin.



Questions?

Special Thanks

For general support, helpful comments, their time, and encouragement.

Áine Doyle - Badass Extraordinaire (OCSC)

Dr. John-Ross Wallrabenstein - Sypris Electronics

Dr. Marcus Rogers - Purdue Cyber Forensics Laboratory

Michael Hale Ligh (MHL) - Volexity

Tatiana Ringenberg - Sypris Electronics