
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

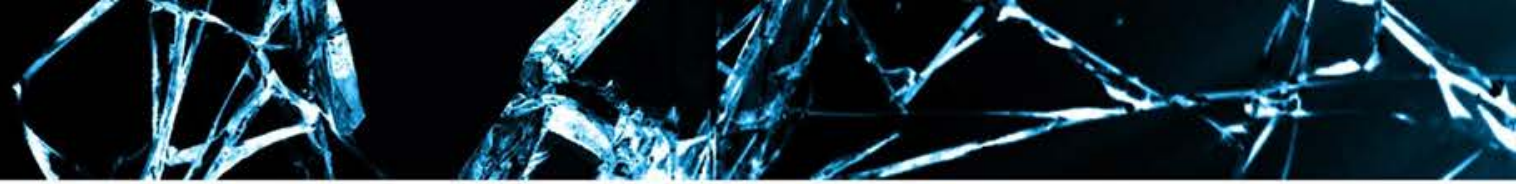
- ❑ Ability to decrypt SChannel TLS connections that use ephemeral key exchanges
- ❑ Ability to decrypt and extract persistent private key and session ticket key directly from memory
- ❑ Map Process IDs to Public certificates and hostnames

# Agenda

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

# Disclaimer

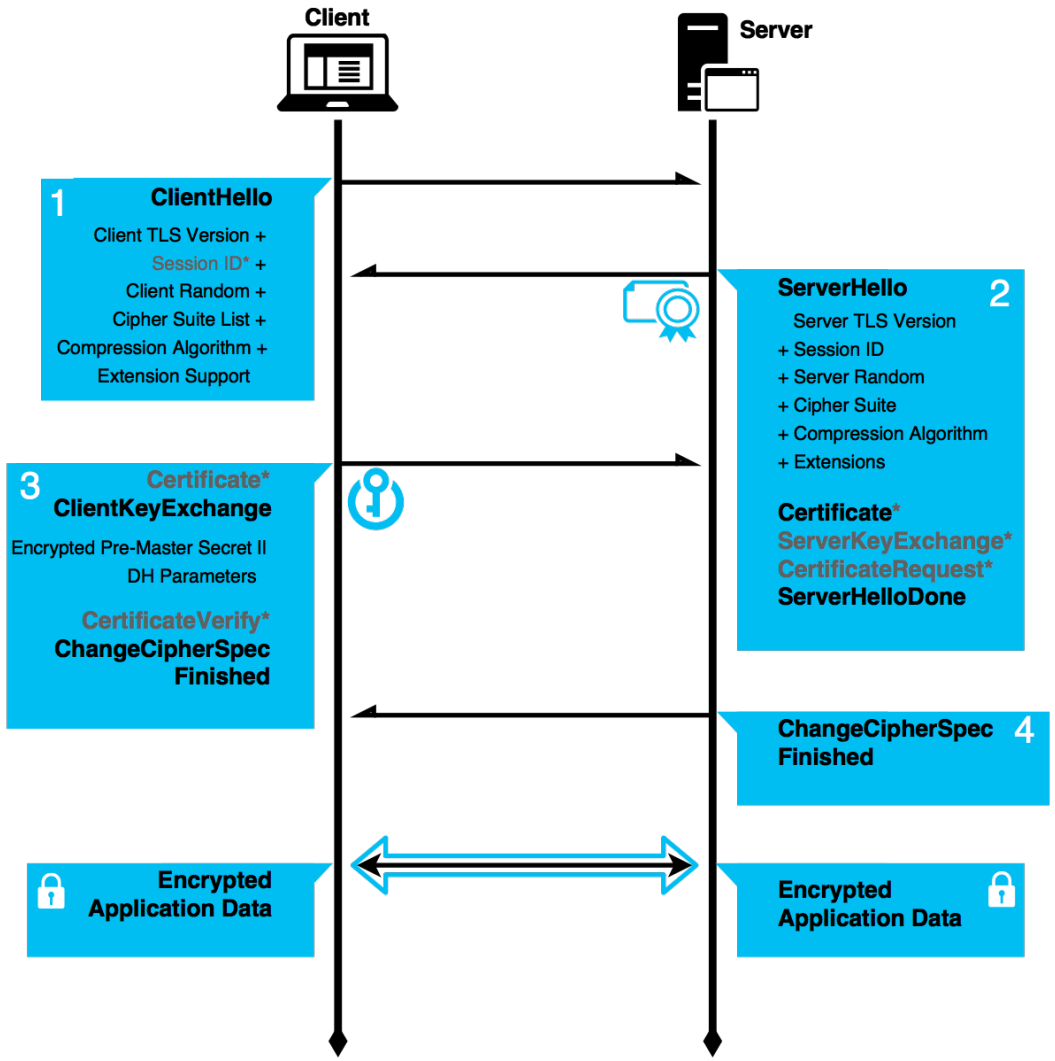
- **This is NOT an exploit**
  - It's just the spec :D
  - …and some implementation specific oddities
- **Microsoft has done nothing [especially] wrong**
  - To the contrary, their documentation was actually pretty great
- **Windows doesn't track sessions for processes that load their own TLS libs**
  - 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...



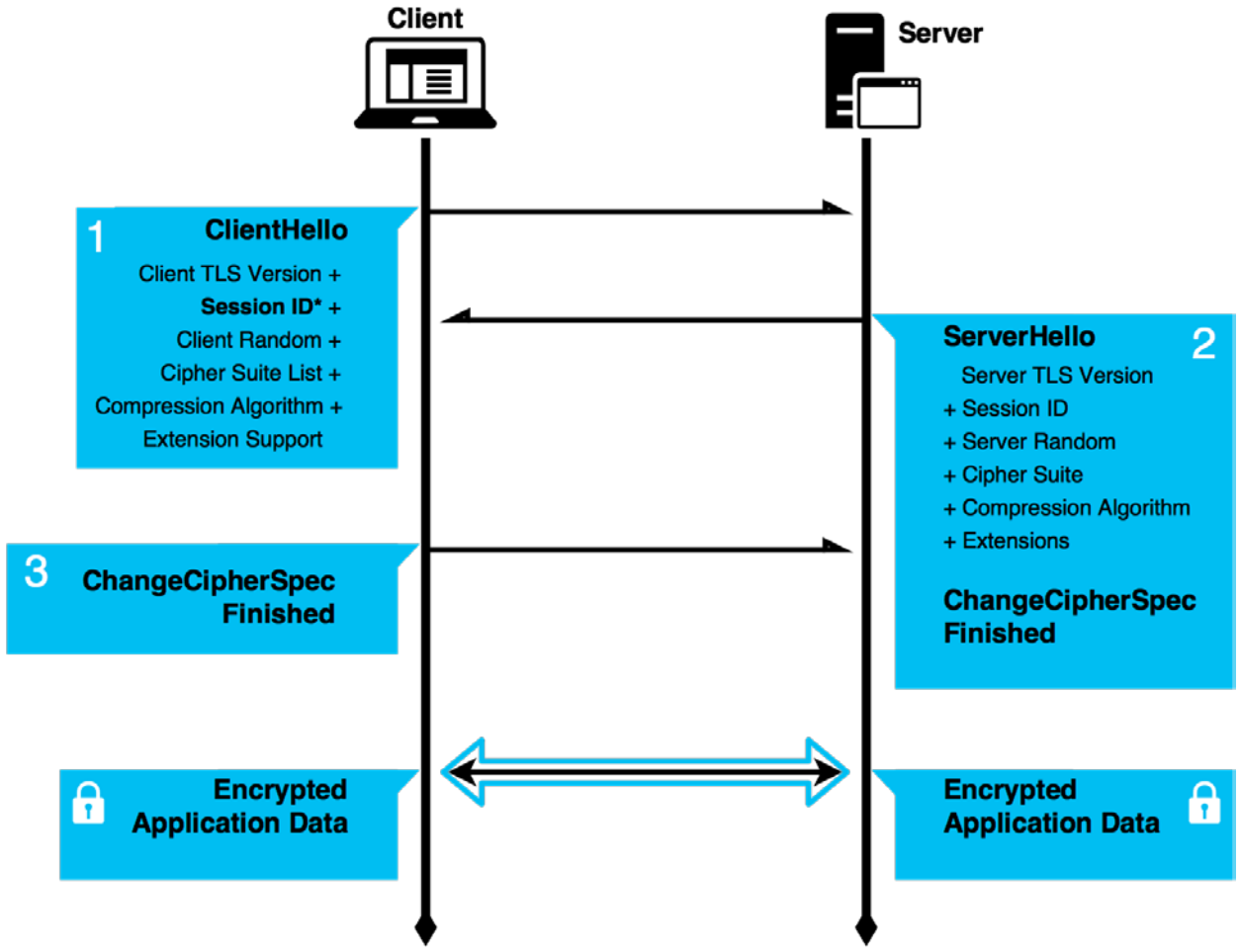
# Background

TLS, Schannel, and CNG

# The infamous TLS Handshake



# 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

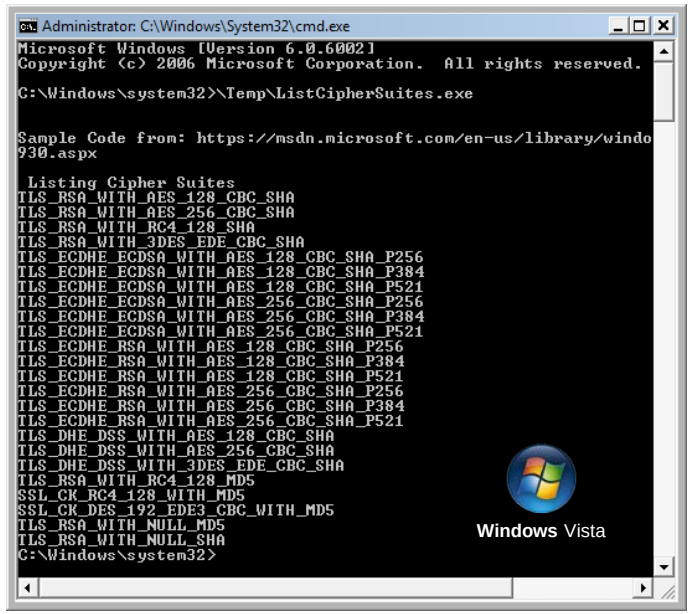
**Secure Channel**

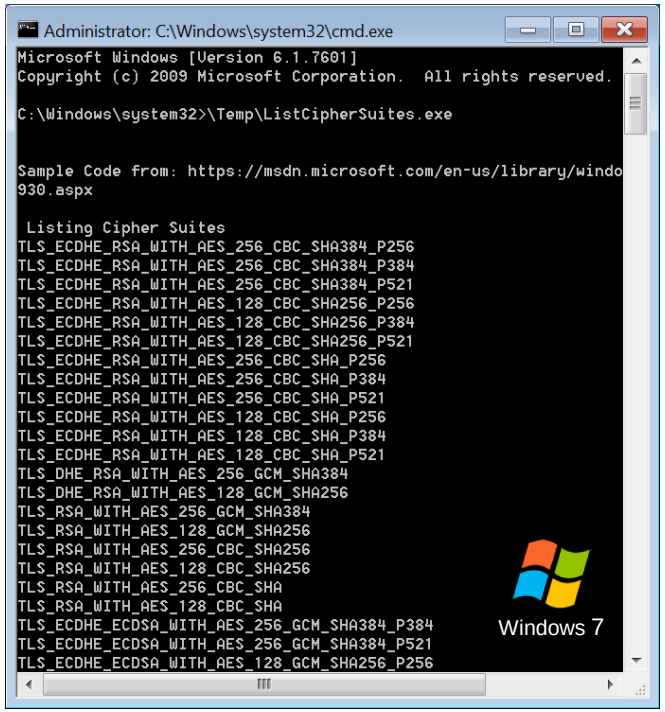
- It's TLS -> the Secure Channel for Windows!
- A library that gets loaded into the "key isolation process" **and** the "client" process
  - Technically a Security Support Provider (SSP)
- Spoiler: the Key Isolation process is LSASS

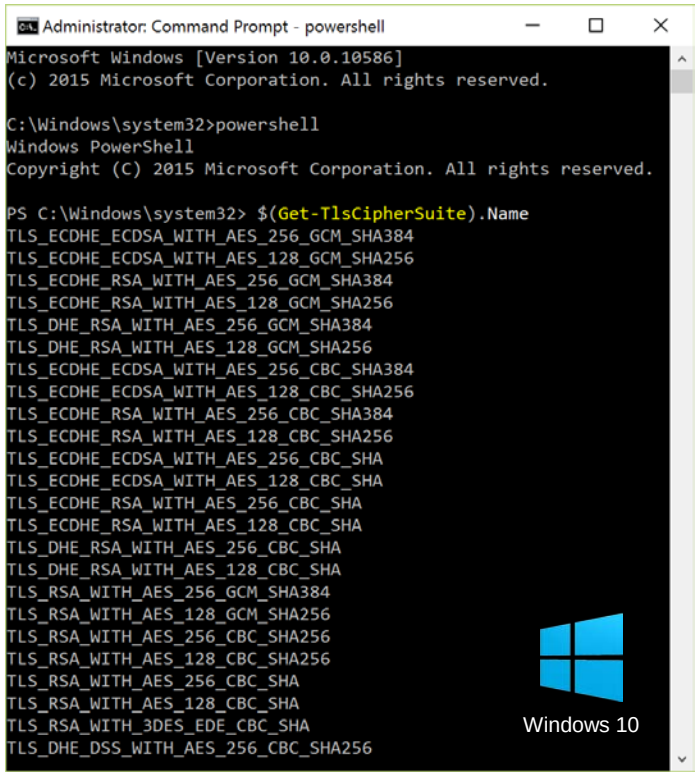
**The CryptoAPI-Next Generation (CNG)**

- 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

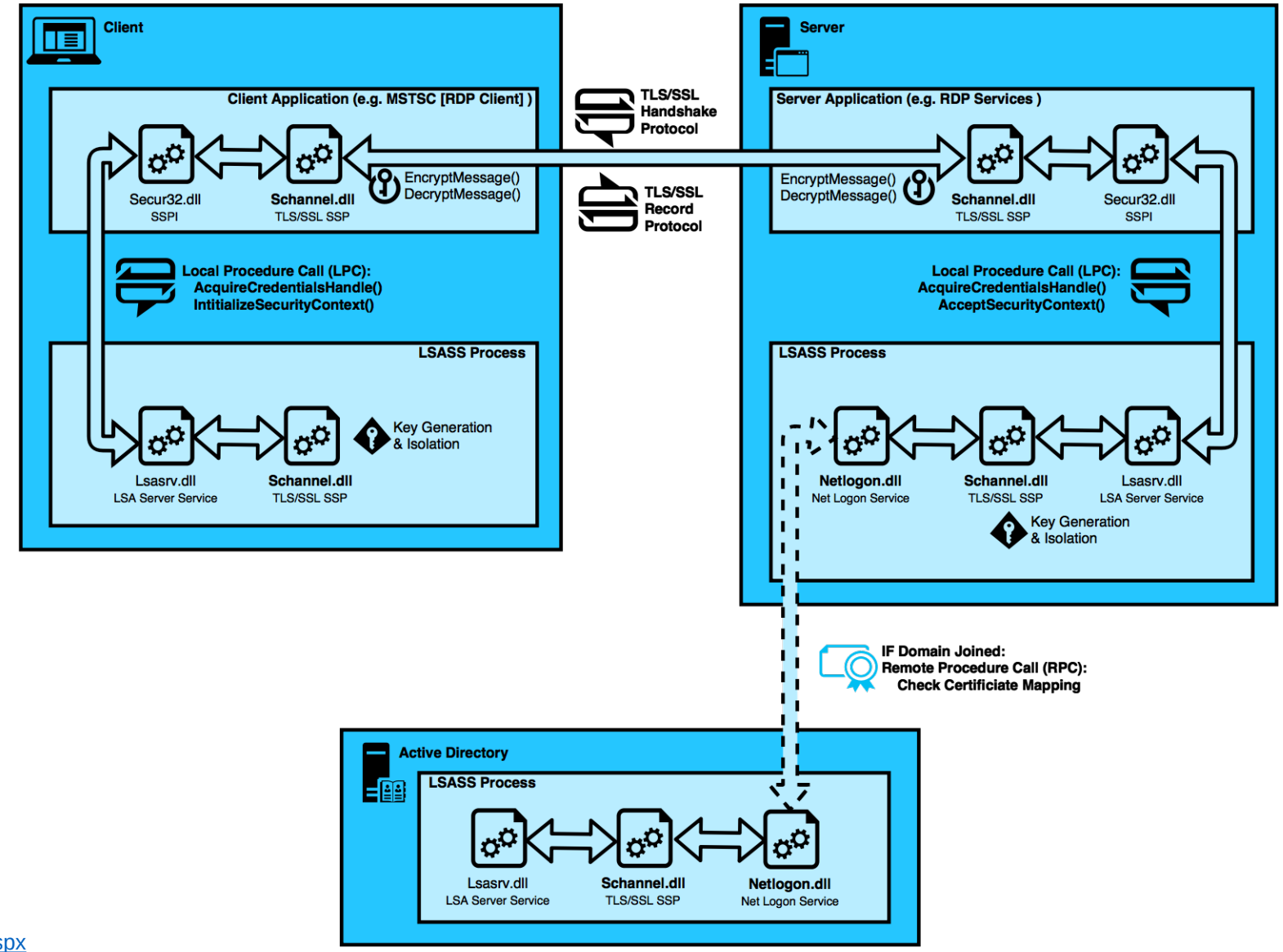



Diagram based on: https://technet.microsoft.com/en-us/library/dn786429.aspx

# CNG Key Isolation

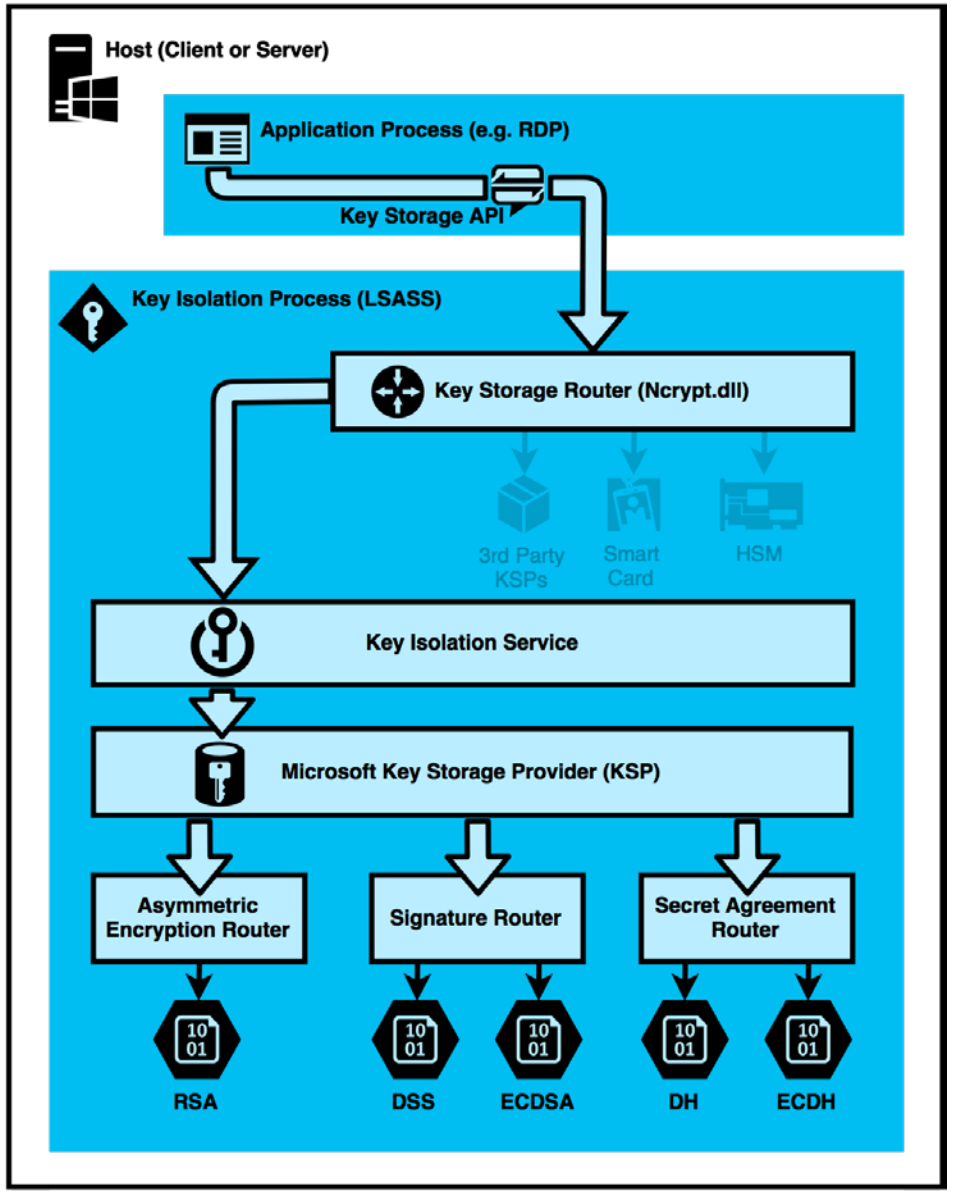
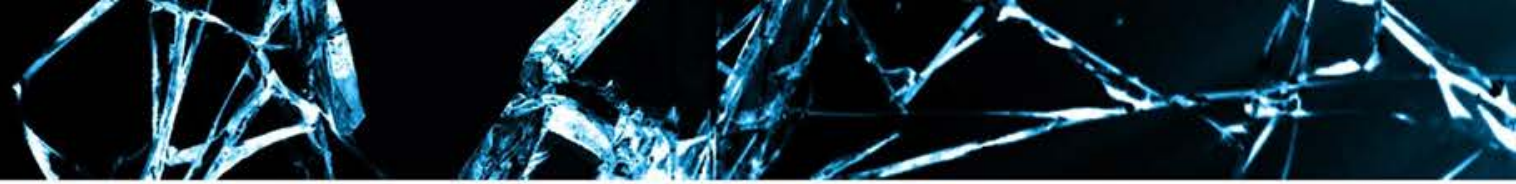



Diagram based on: https://msdn.microsoft.com/en-us/library/windows/desktop/bb204778.aspx



# Secrets

# The Keys

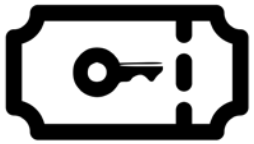
Session Keys

Master Secret

Pre-Master Secret

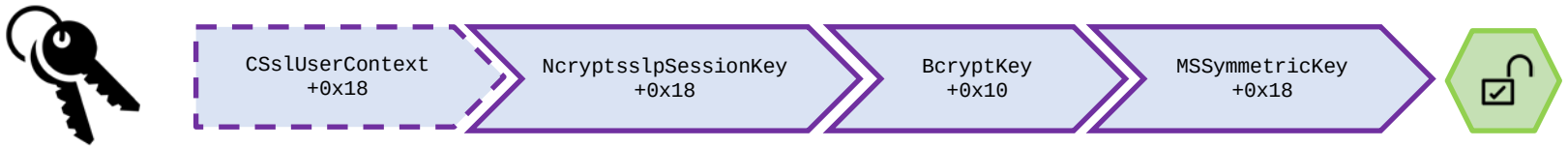
Ephemeral Private Key*

Persistent Private Key (Signing)

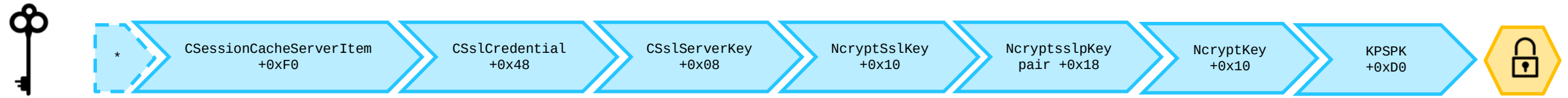


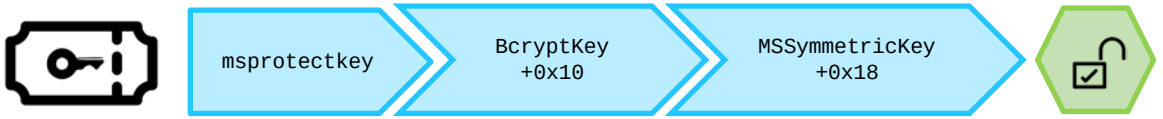
Session Ticket Key*

# The Keys? We got 'em.





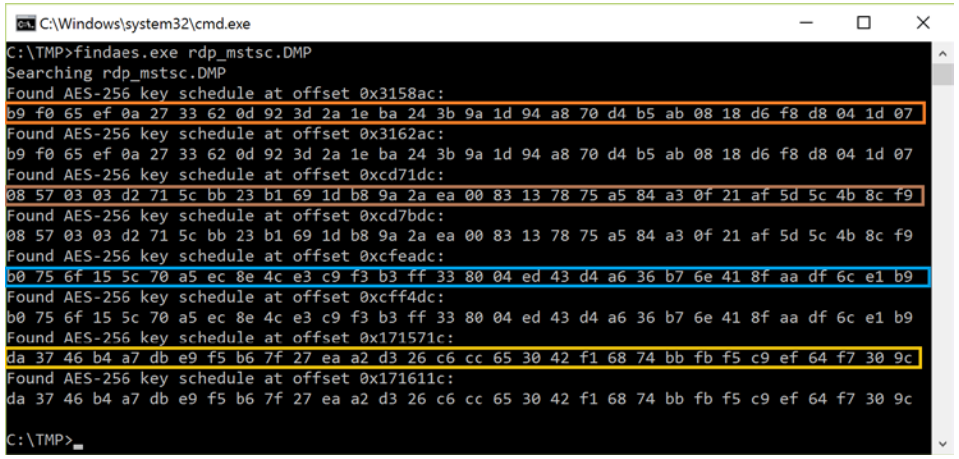
# Session Keys

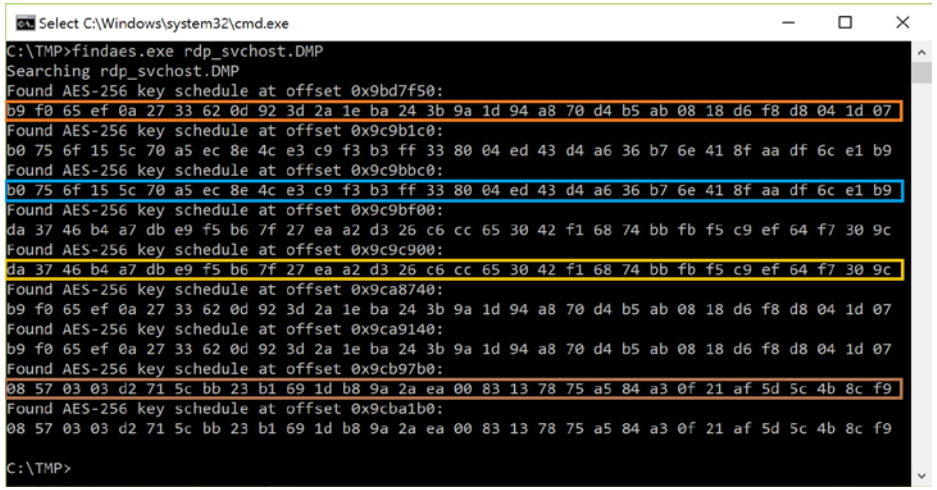
- Smallest scope / most ephemeral
- Required for symmetric encrypted comms
- Not going to be encrypted

**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




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

# Session Keys

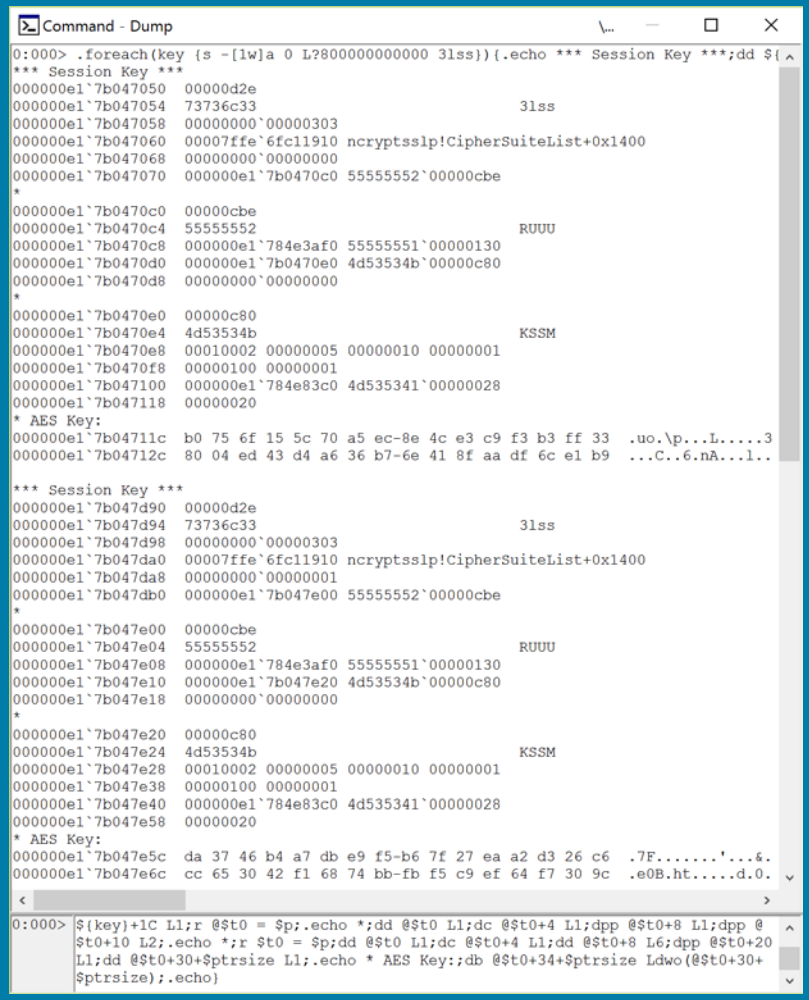
| _SSL_SESSION_KEY | |
|---|---|
| 4 | cbStructLength |
| 4 | dwMagic [“ssl3”] |
| 4 | dwProtocolVersion |
| 4/8 | pvCipherSuiteListEntry |
| 4 | IsWriteKey |
| 4/8 | pvBcryptKeyStruct |

| _BCRYPT_KEY | |
|---|---|
| 4 | cbStructLength |
| 4 | dwMagic [“UUUR”] |
| 4/8 | pvBcryptProvider |
| 4/8 | pvBcryptSymmKey |

| _MS_SYMMETRIC_KEY | |
|---|---|
| 4 | cbStructLength |
| 4 | dwMagic [“MSSK”] |
| 4 | dwKeyType |
| ... | ... |
| 4 | KeyLength |
| ? | SymmetricKey |
| ? | SymmKeySchedule |

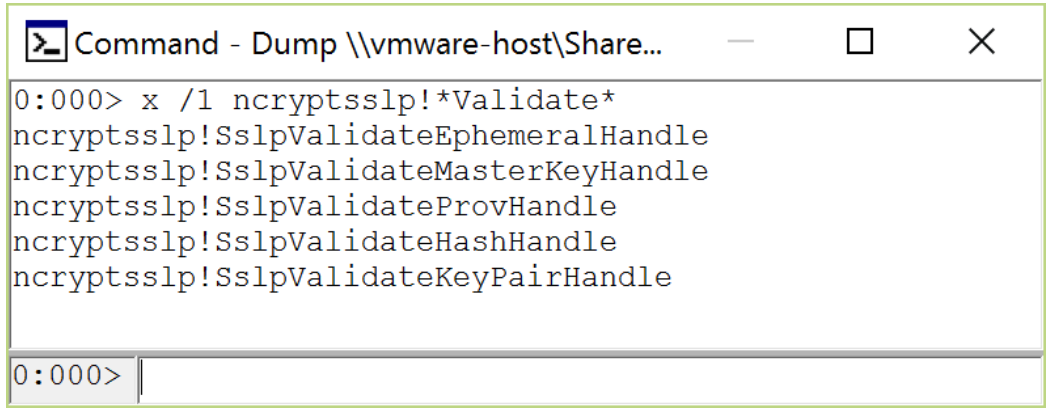
CSslUserContext

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



# The Ncrypt SSL Provider (ncryptsslp.dll)

## Ncryptsslp Validation function Symbols

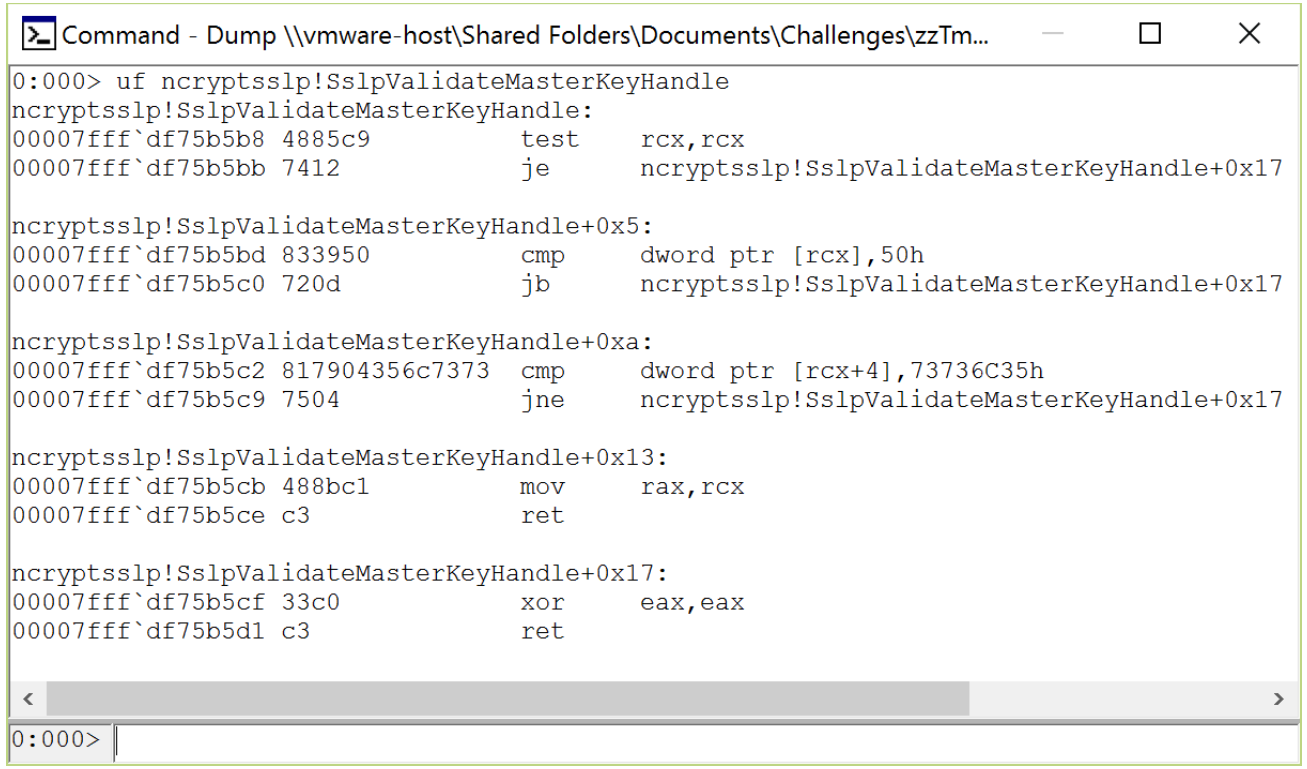



## These functions do three things:

- ❑ 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



# 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

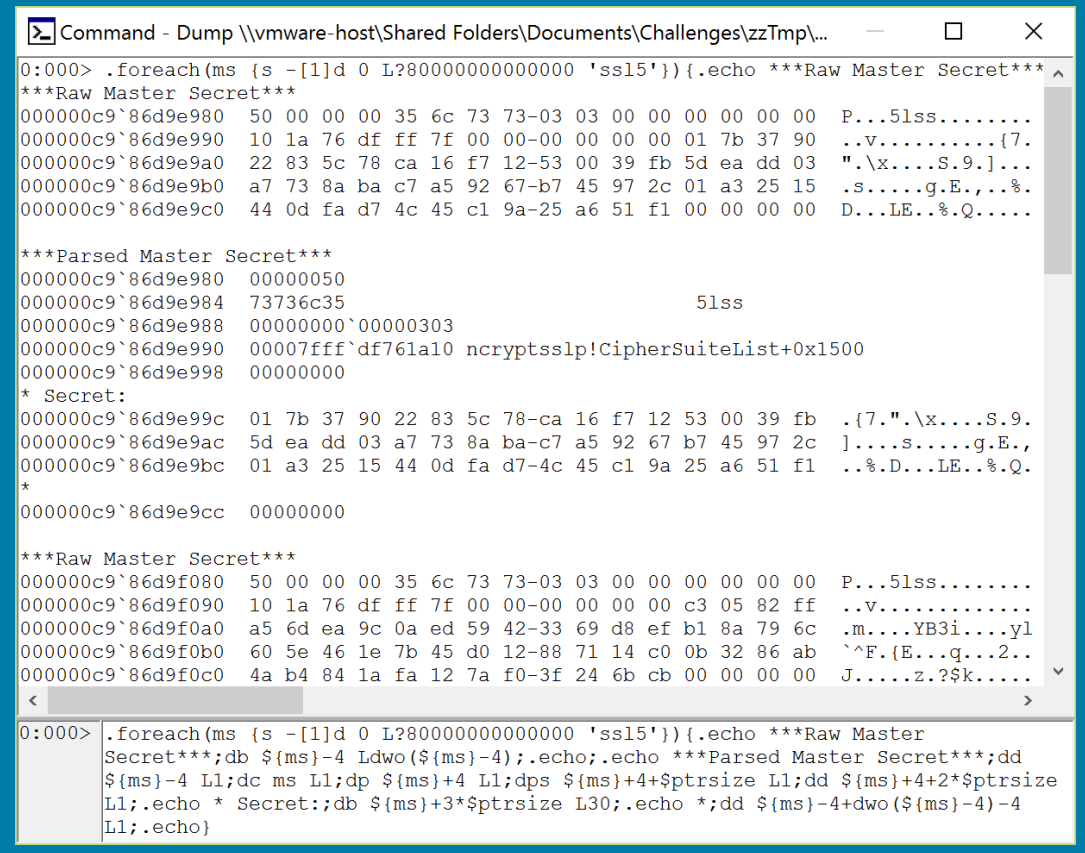
- Basically the Holy Grail for a given connection
  - It always exists
  - It's what gets cached and used to derive the session keys
- Structure for storage is simple - secret is unencrypted (as you'd expect)
- This + **Unique ID** = decryption, natively in tools like wireshark

So...how do we get there?

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

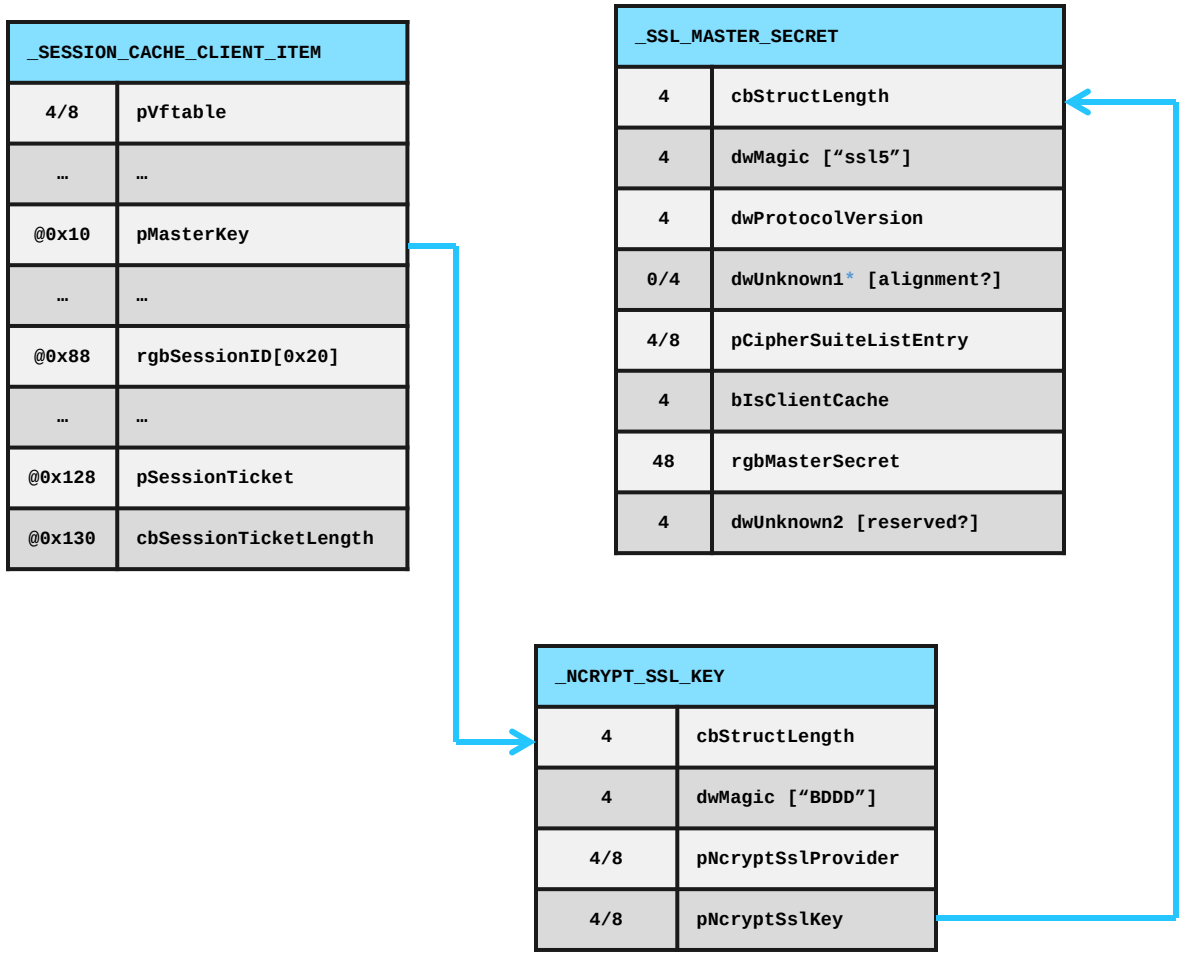


# Master Secret Mapped to Unique Identifier

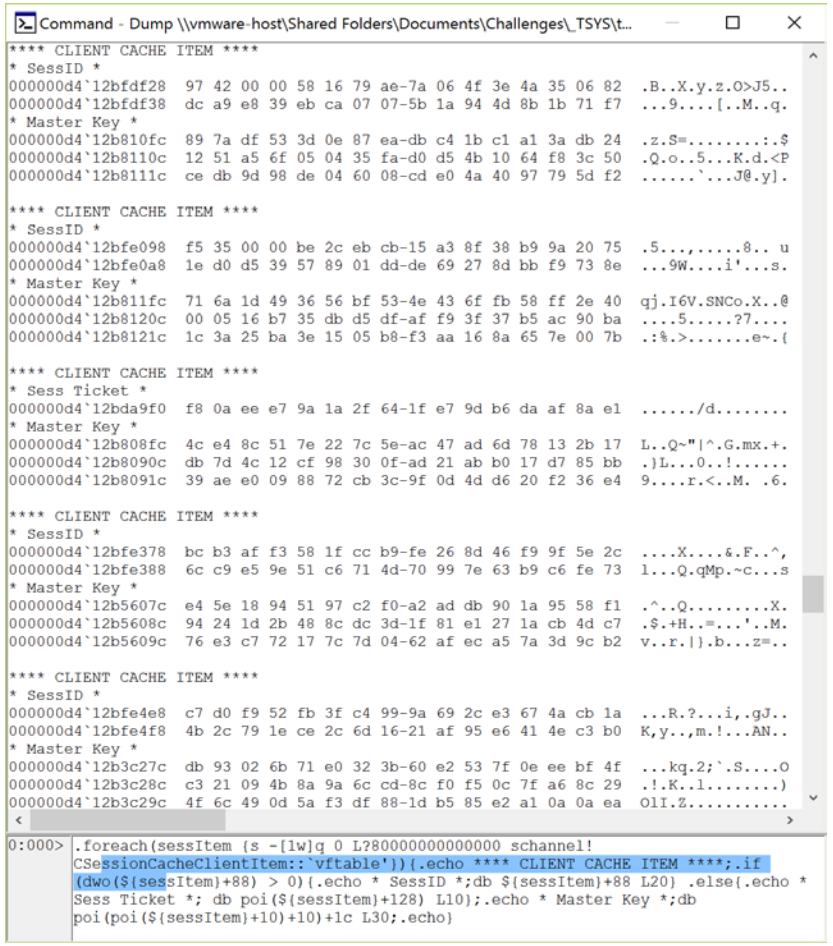
- The Master Key is linked back to a unique ID through an "`NcryptSslKey`"
- The NcryptSslKey is referenced by an "`SessionCacheItem`"
- The SessionCacheItem contains either the SessionID, or a pointer and length value for a SessionTicket
  - Instantiated as either client or server item

**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



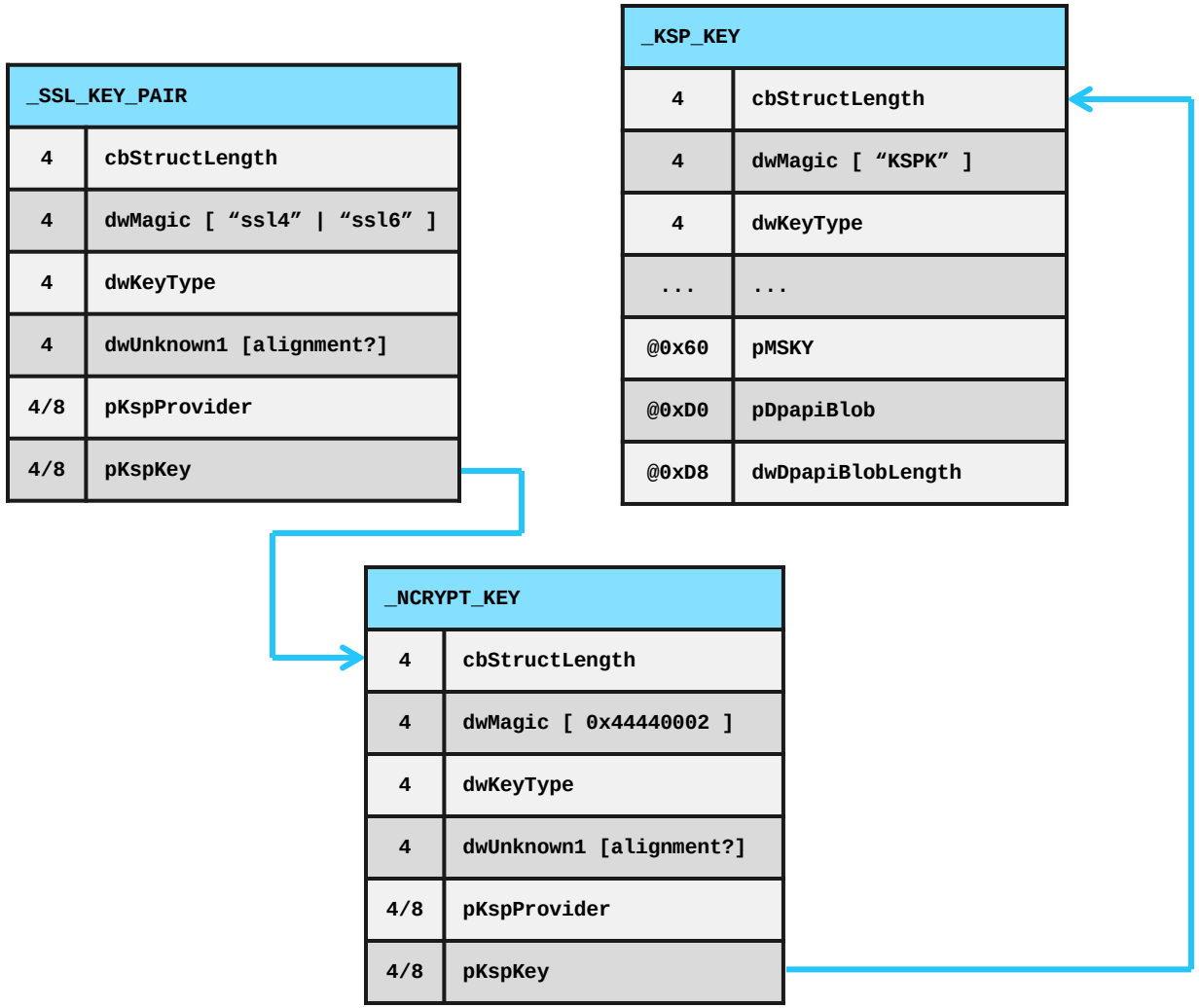

## Wireshark SSL Log Format

```
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*

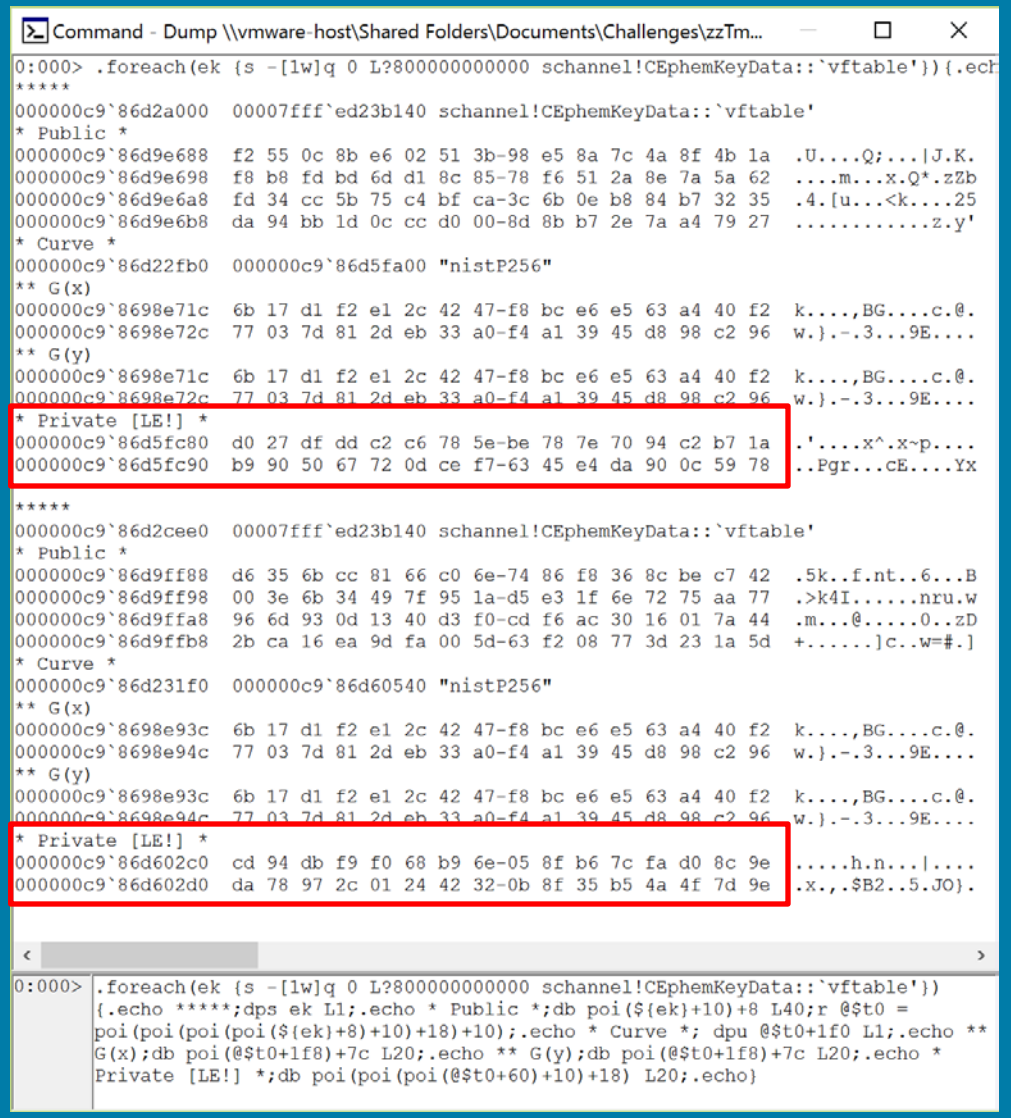



*NcryptKey not to be confused with NcryptSslKey

# Ephemeral Private Key

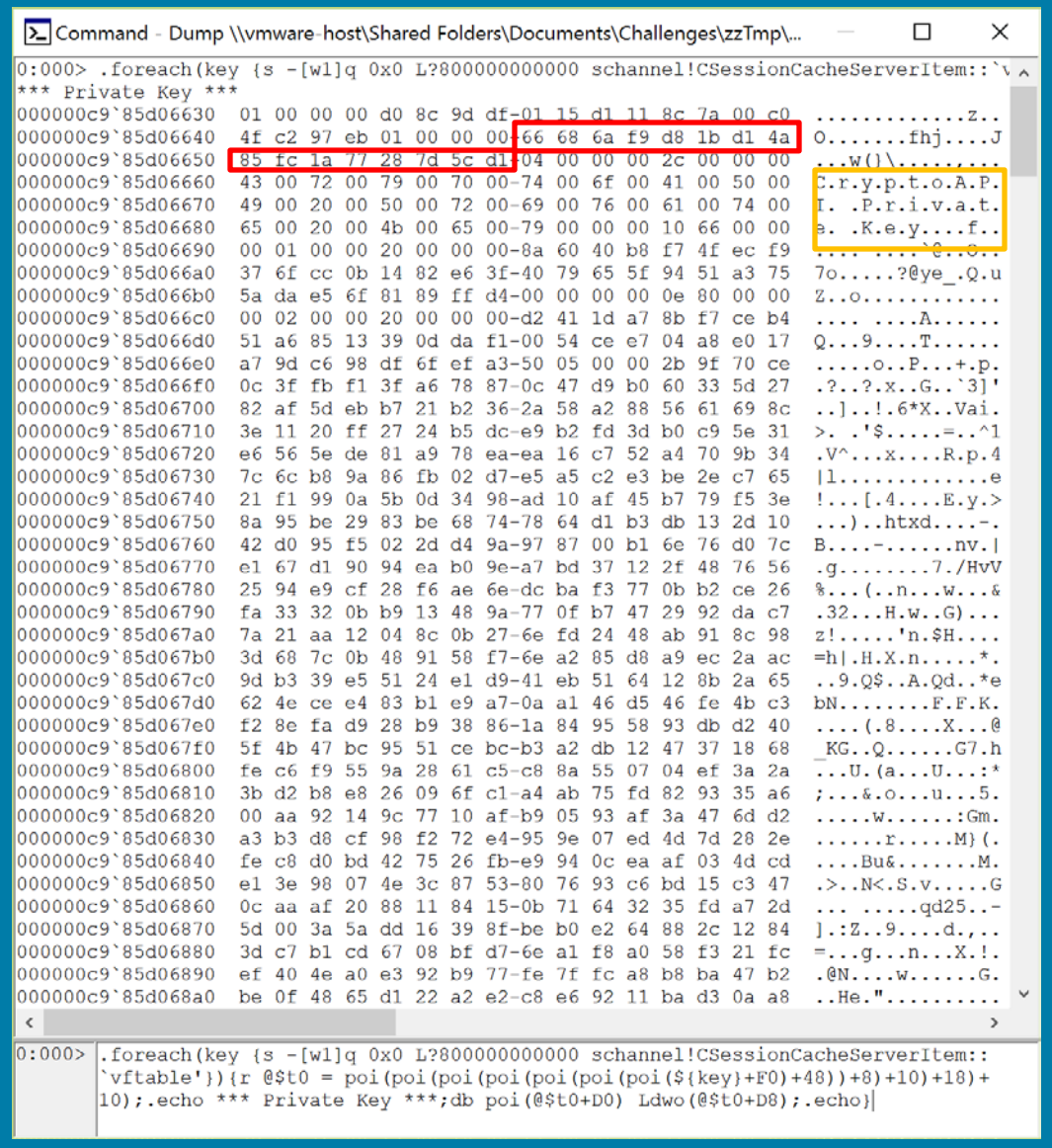
- **For performance, reused across connections**
  - Given the public connection params, we can derive the PMS and subsequently MS
- **Stored unencrypted in a LE byte array**
  - Inside of MSKY struct
- The curve parameters are stored in the KPSK
  - Other parameters (A&B, etc) are stored in MSKY w/ 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`**"

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

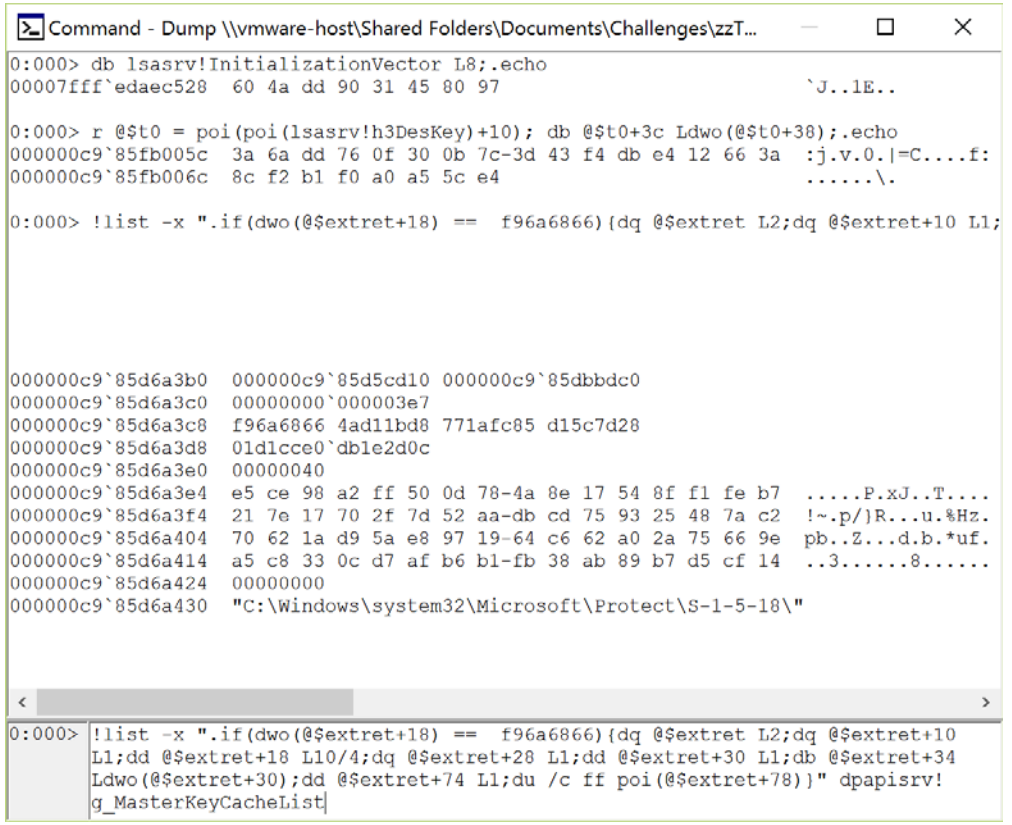


# Decrypting Persistent Key - DPAPI

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

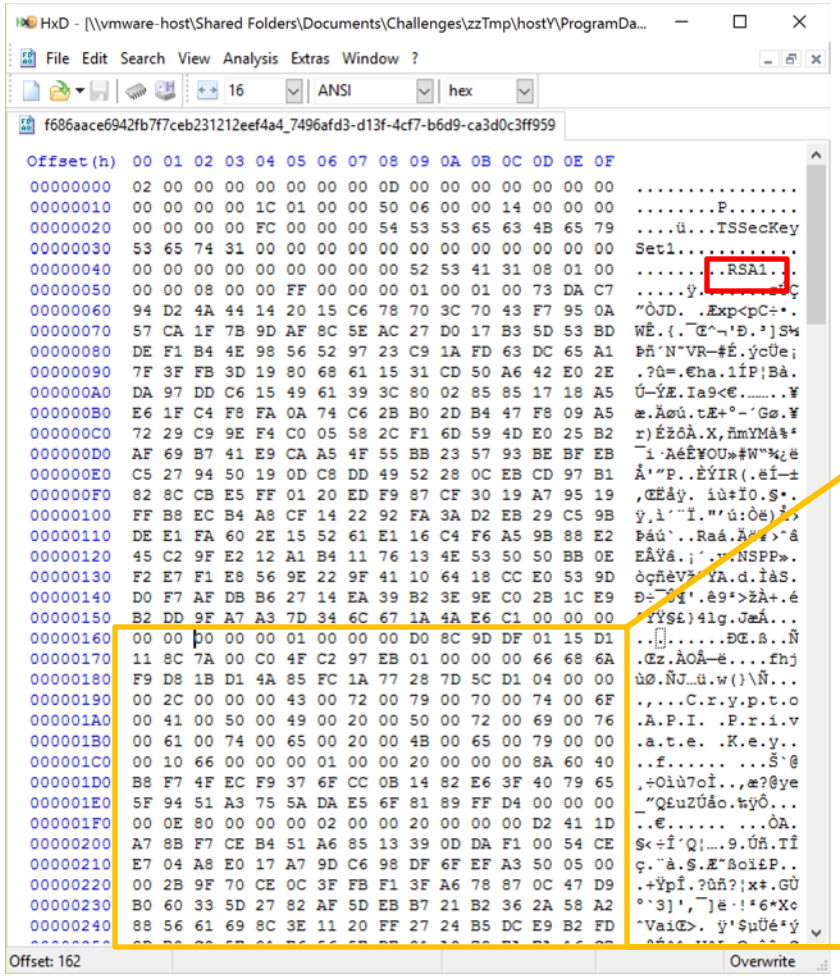
**OR**

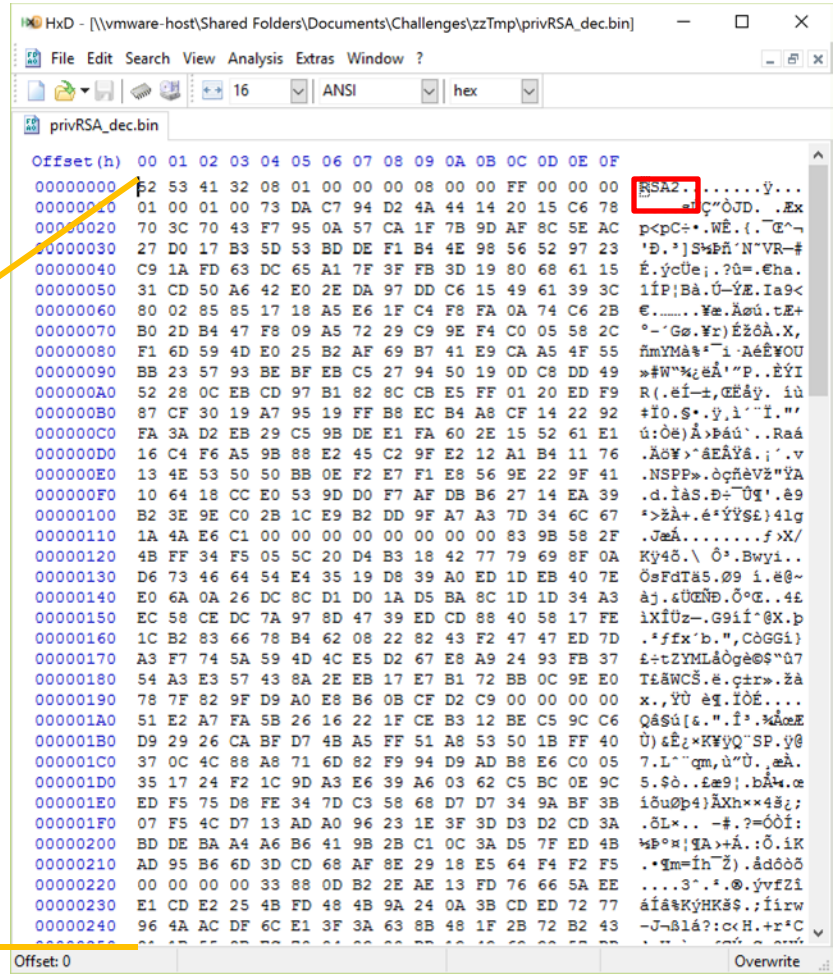
- Can decrypt directly from memory :D
  - MasterKeys get cached in Memory
    - On Win10 in: dpapisrv!g_MasterKeyCacheList
    - See Mimilib for further details
    - Even though symbols are sort of required, we could likely do without them
      - There are only two Bcrypt key pointers in lsasrv's .rdata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




Cached DPAPI MK + Params to Decrypt

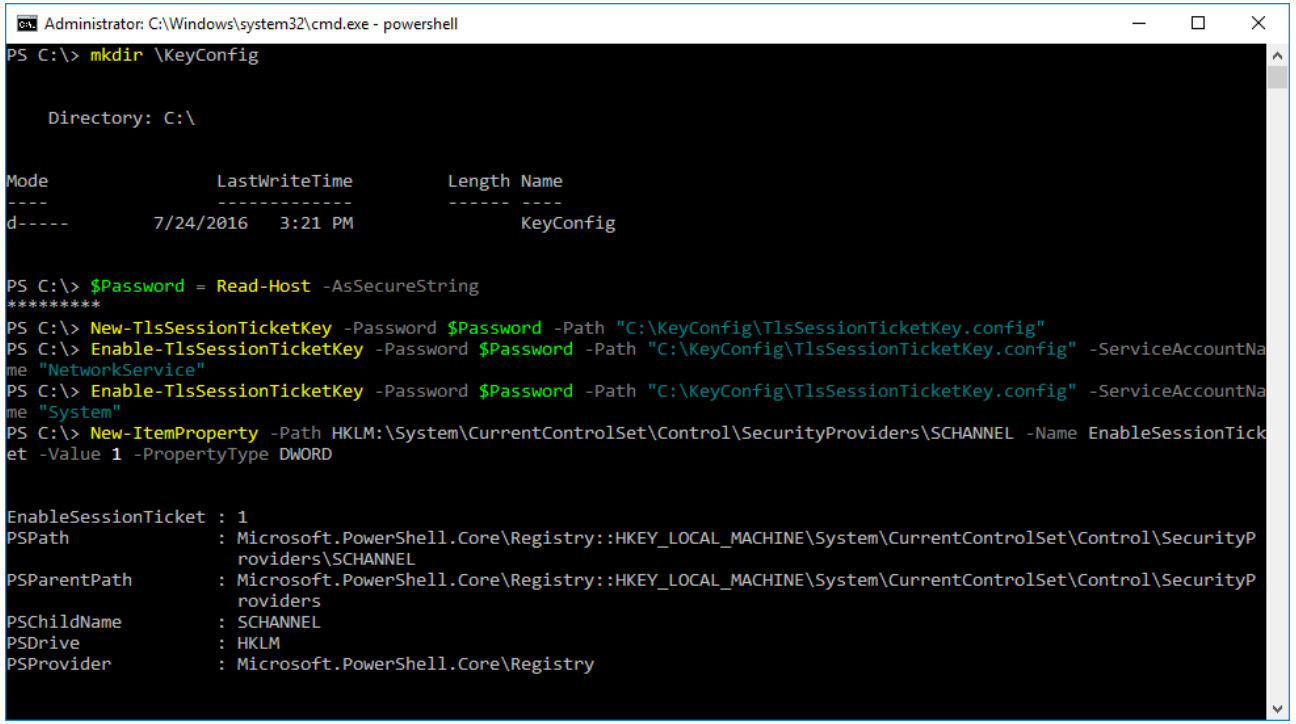
# Decrypting Persistent Key - DPAPI



# Session Tickets

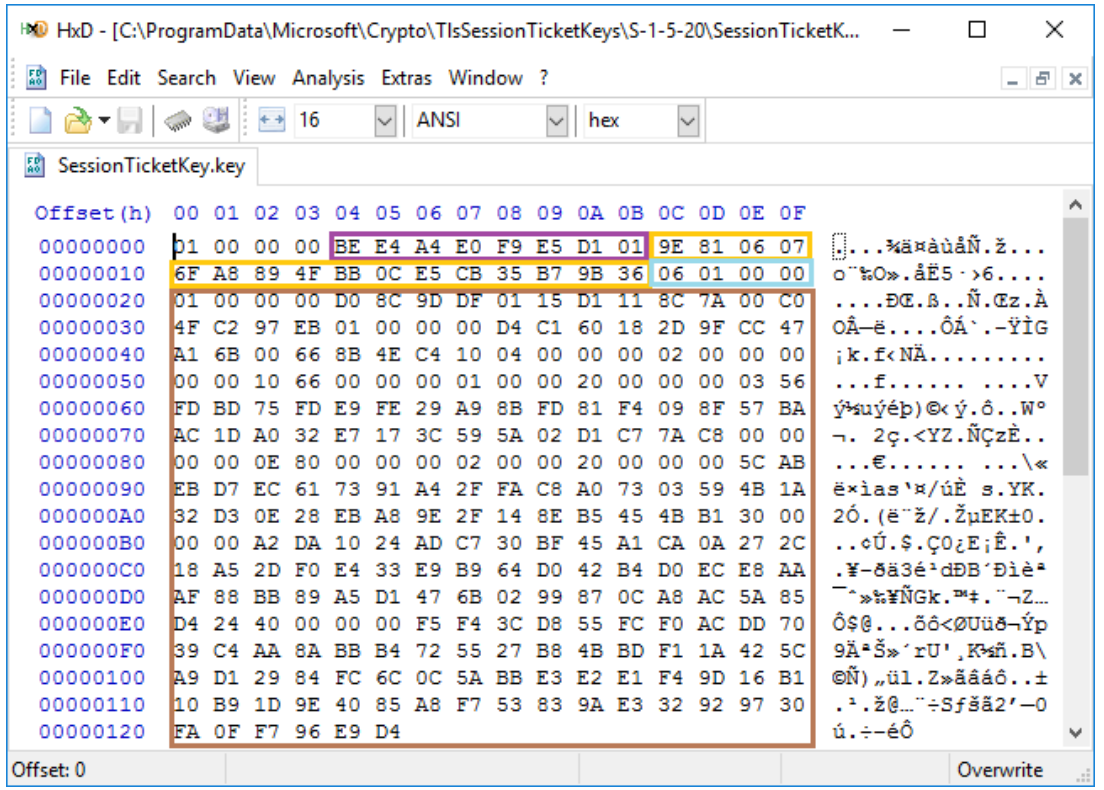
- Not seemingly in widespread use with IIS?
  - Comes around w/ Server 2012 R2
  - Documentation is lacking.
- Enabled via reg key + powershell cmdlets?
  - Creates an “Administrator managed” session ticket key
- Schannel functions related to Session Tickets load the keyfile from disk
- Export-TlsSessionTicketKey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




Reference to DISABLING session tickets in Win8.1 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
  - 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?

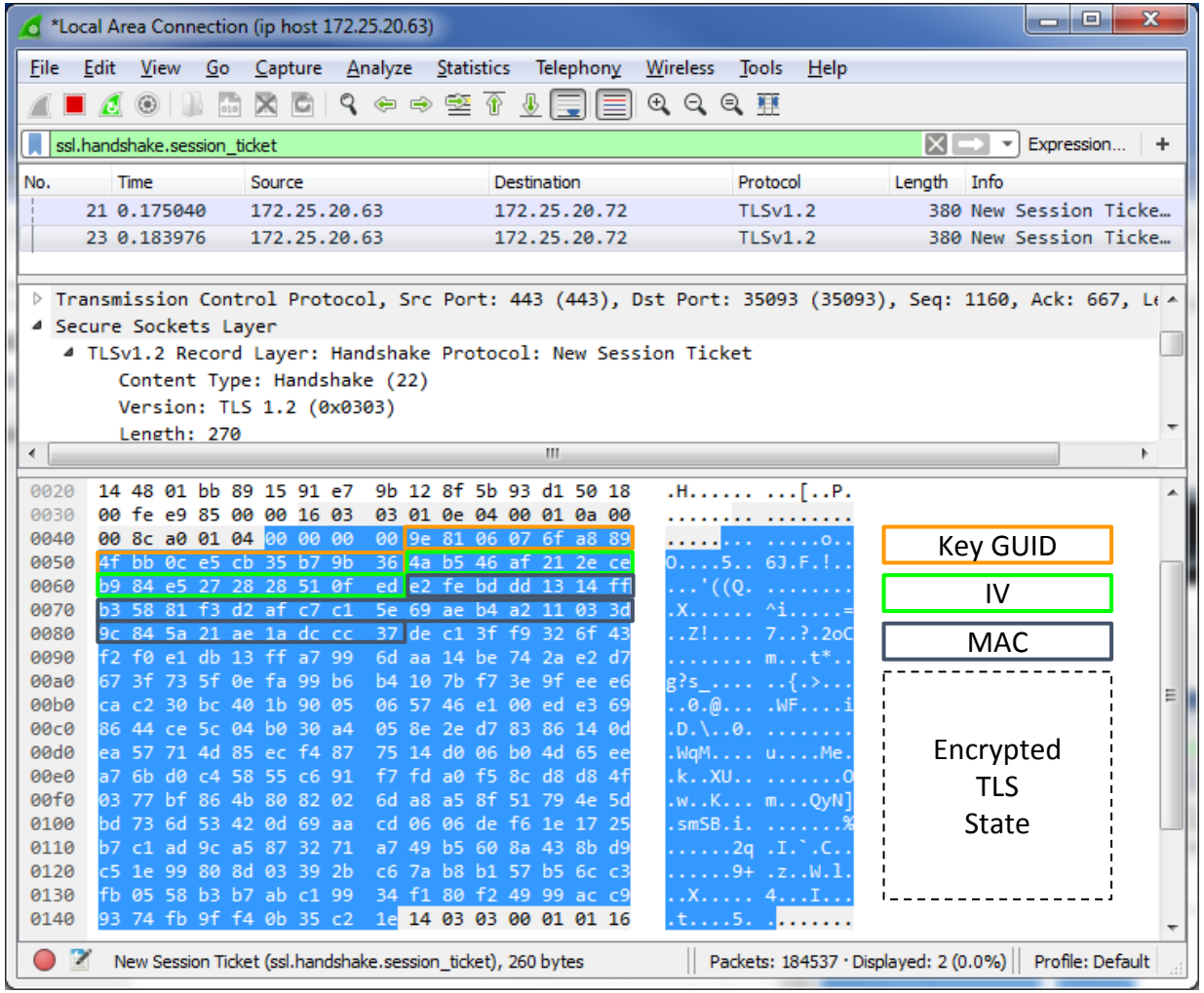
Session Ticket Key GUID

Size of ensuing DPAPI Blob

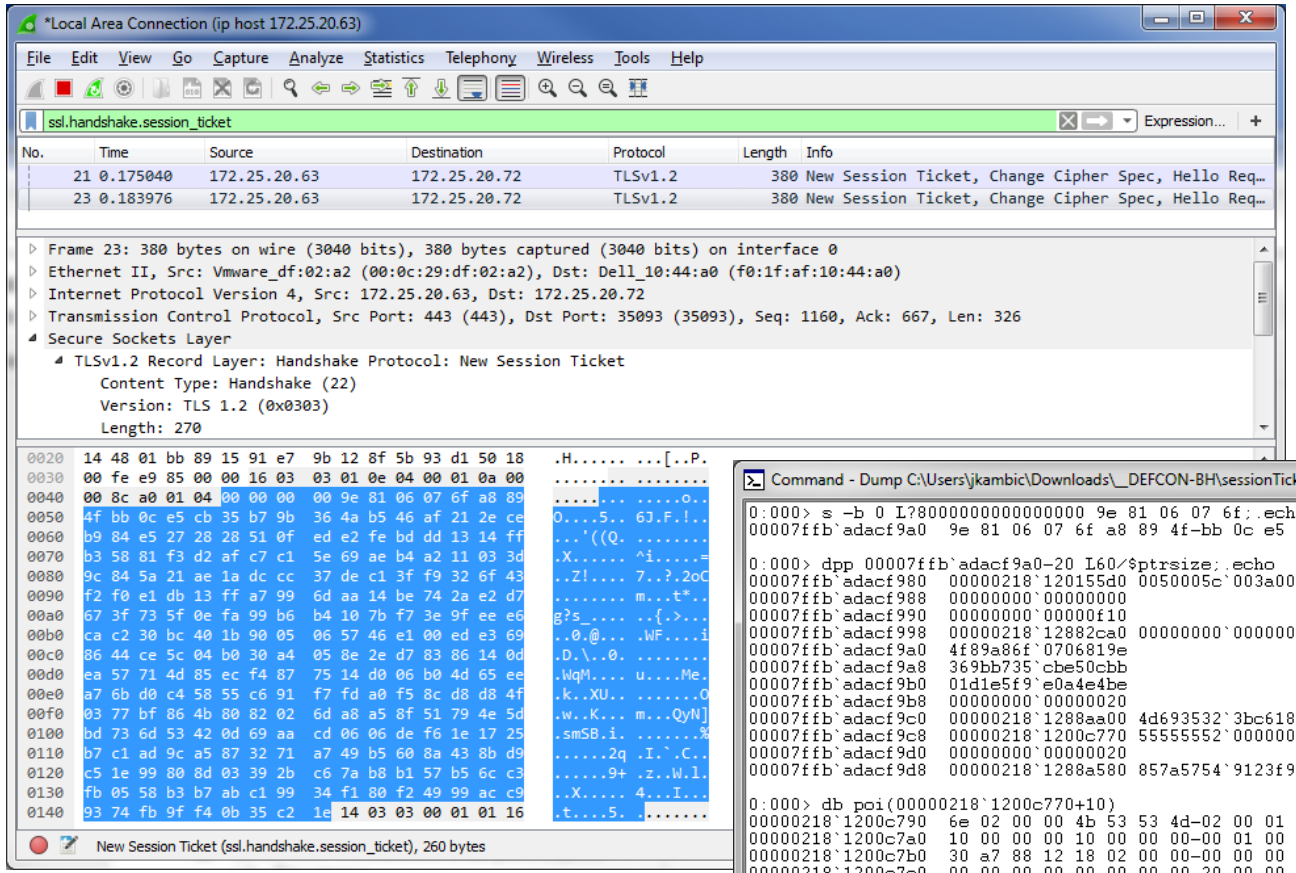
DPAPI Blob (contains it's own fields)

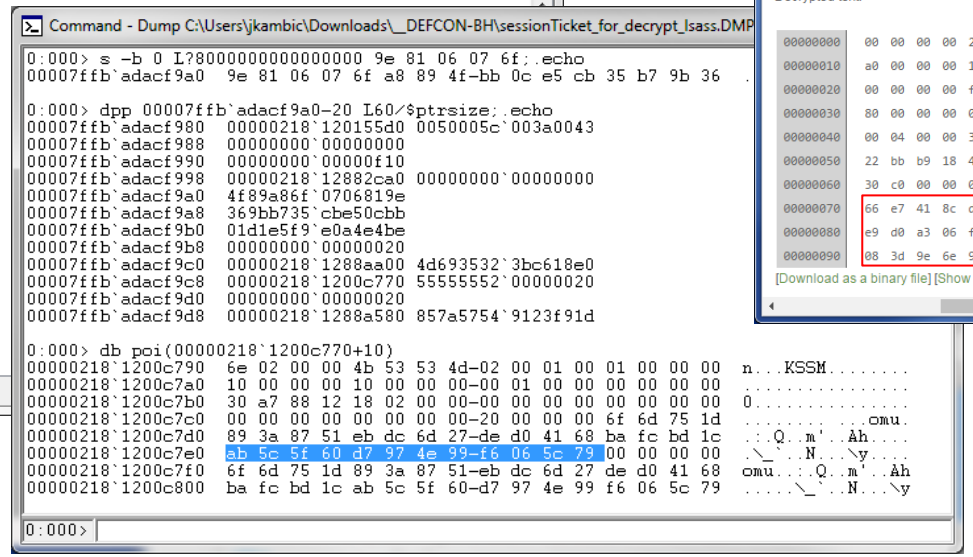
# Decrypting Session Tickets

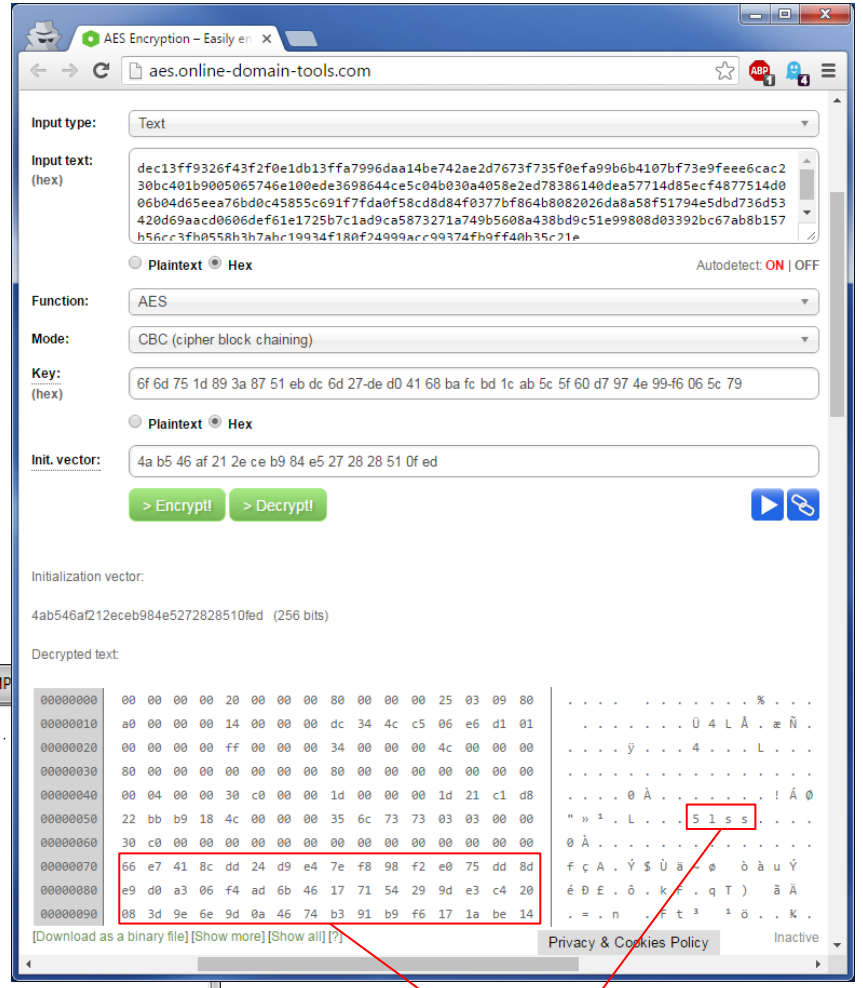
- Session Ticket structure pretty much follows the RFC (5077), except:
  - MAC & Encrypted State are flipped (makes a lot of sense)
- After extracting/deriving the Symm key, it's just straight AES 256
- Contents of the State are what you'd expect:
  - Timestamp
  - Protocol/Ciphersuite info
  - MS struct



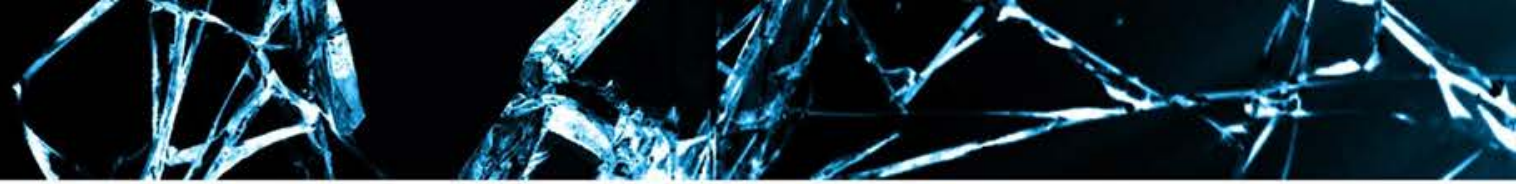
# Decrypting Session Tickets



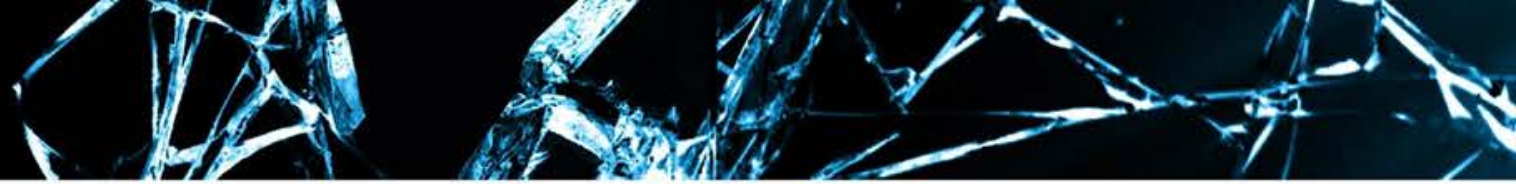




Master Secret



# 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)

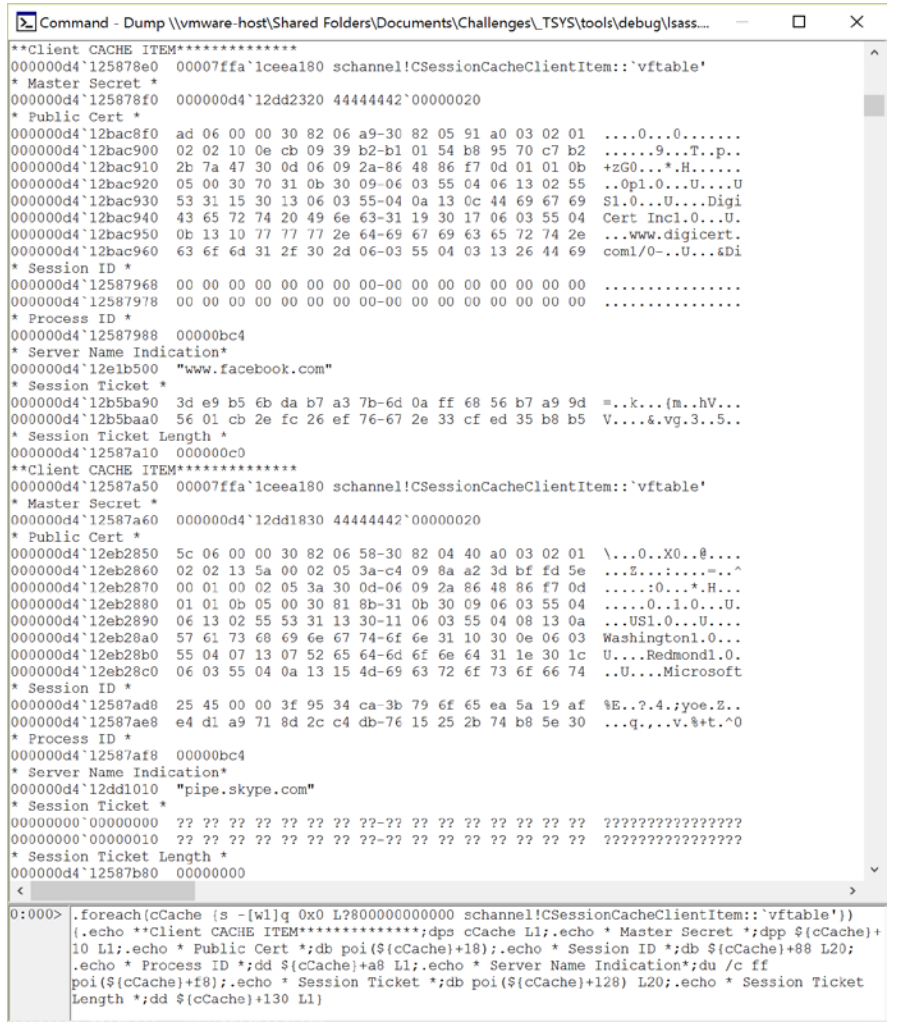
  `m_dwEnableSessionTicket` controls use of session tickets (e.g. `0, 1, 2`)

## 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

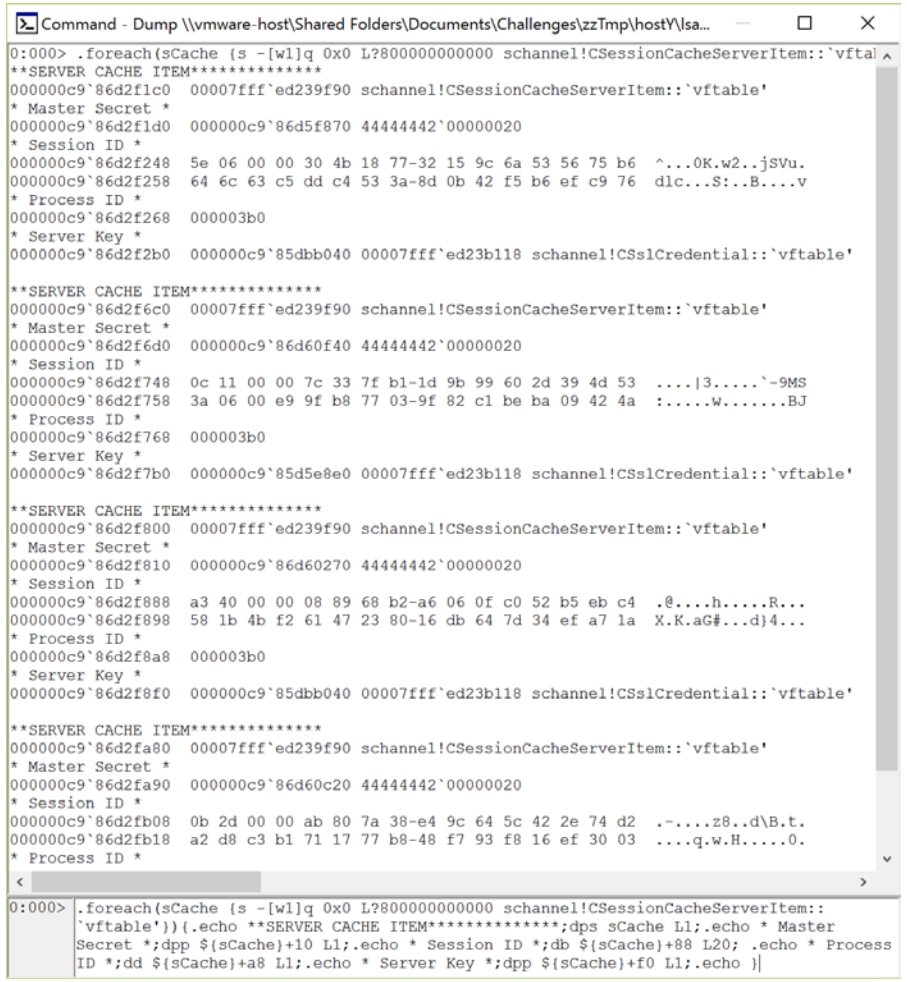
# This is your Schannel Cache (x64)

```
'_SSL_SESSION_CACHE_CLIENT_ITEM': [ 0x140, {
    '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': [0xF8, ['pointer64', ['void']]],
    'CSessCacheManager': [0x110, ['pointer64', ['void']]],
    'SessionTicket': [0x128, ['pointer64', ['void']]],
    'SessionTicketLen': [0x130, ['int']],
    }],
```



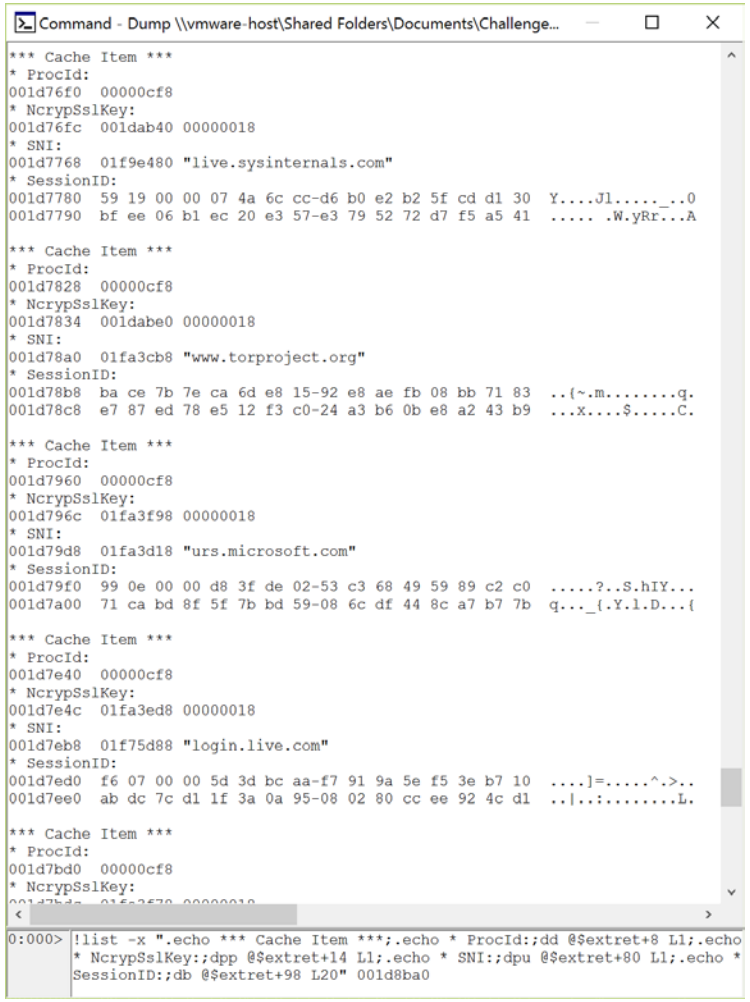
# This is your Schannel Cache (x64)

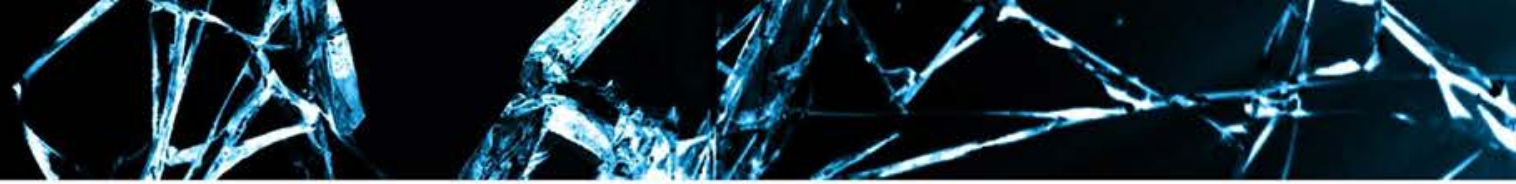
```
'_SSL_SESSION_CACHE_SERVER_ITEM': [ 0x110, {
    'Vftable': [0x0, ['pointer64', ['void']]],
    'NcryptKey': [0x10, ['pointer64', ['void']]],
    'NcryptSslProv': [0x60, ['pointer64', ['void']]],
    'SessionId': [0x88, ['array', 0x20, ['unsigned char']]],
    'ProcessId': [0xa8, ['unsigned long']],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



# 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']]],
    }],
```





# Automating it

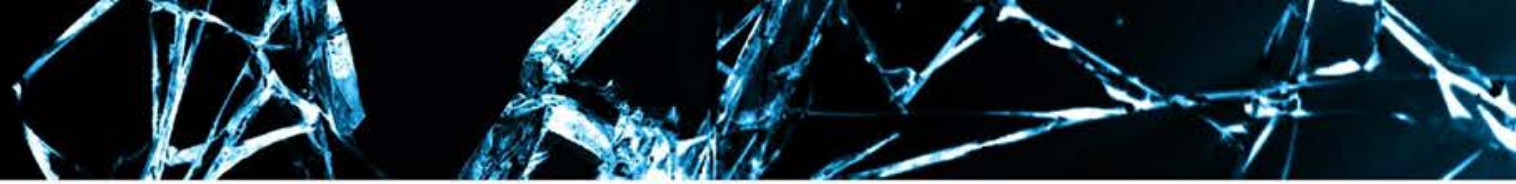
# Volatility / Rekall

- 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)

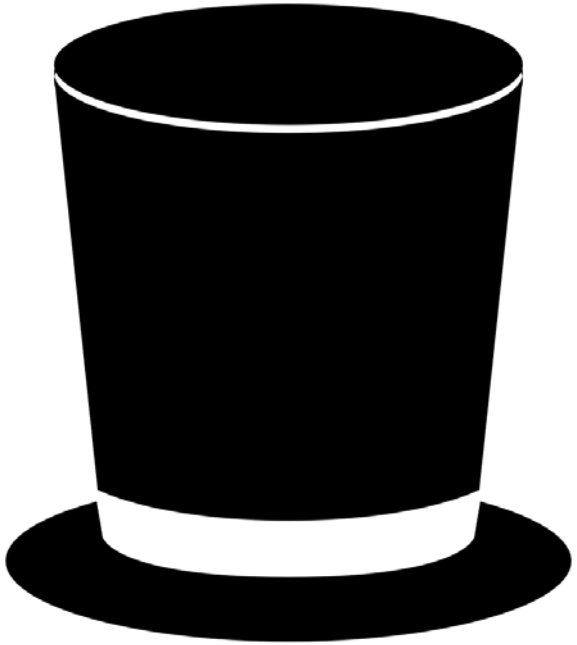
```
      % vol.py --plugins=./plugins --profile=Win10x64 -f ./Win10-Test-c2a4a77d.vm
em lsasslkey
Volatility Foundation Volatility Framework 2.5
RSA Session-ID:b93c0000a110690b4ae9111bce5725c6c47a037b3c39c49c75ce51e1c2eb79ee M
aster-Key:bc28467999b99fd3fdf3a24642c5d93b9ab43e51627f6e0145ef120ba98a1c3223f3dbe
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

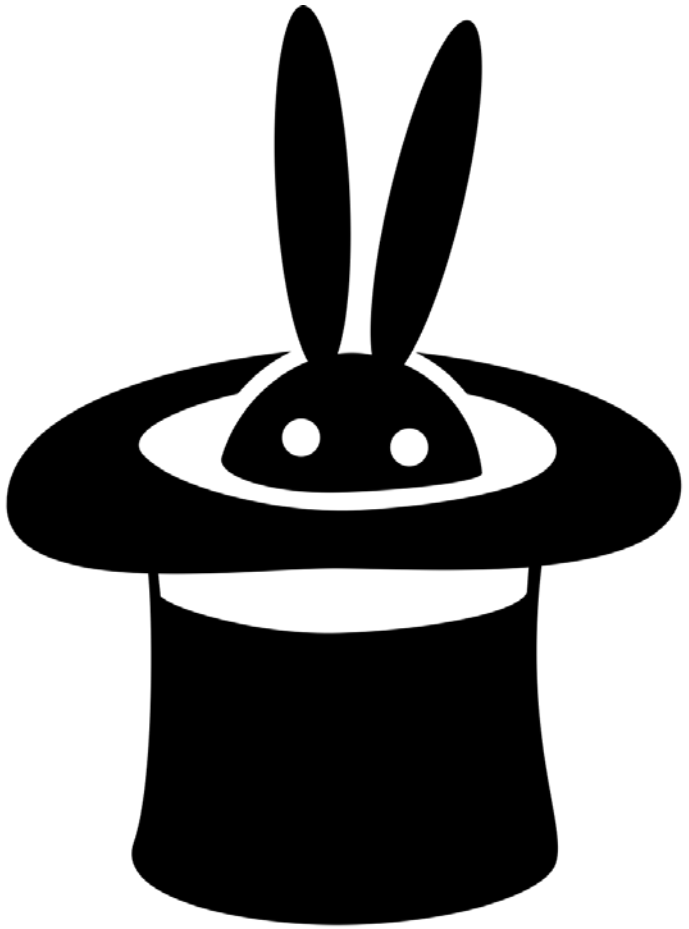
- **We're working with internal, undocumented structures**
  - They change over time -- sometime around April 2016, an element appears to have been inserted in cache after the SessionID and before the SNI
    - Not a huge deal, except when differences amongst instances of same OS (e.g. ones that have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

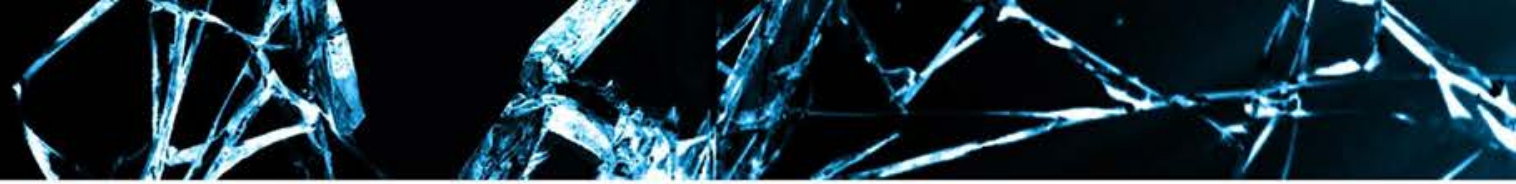


Fin.



# Questions?

@TinRabbit_



# 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