# Cunning with CNG:

## Soliciting Secrets from Schannel

DEFCON 24

# Why you might care

I.E. WHAT YOU GET TODAY

- Extracting TLS / SSL Keys (of various types) from memory
- Ability to decrypt TLS connections that use ephemeral key exchanges
  - For anything that uses Schannel: RDP, IE, Powershell, etc...pretty much anything .NET too
  - Past connections AND Future since the point of the cache is resumption
- Undocumented / partially documented structures elucidated
- TLS session caches mapped to the requesting processes, with SNIs
- A tool that does these things via Volatility/Rekall
- A paper that documents these things

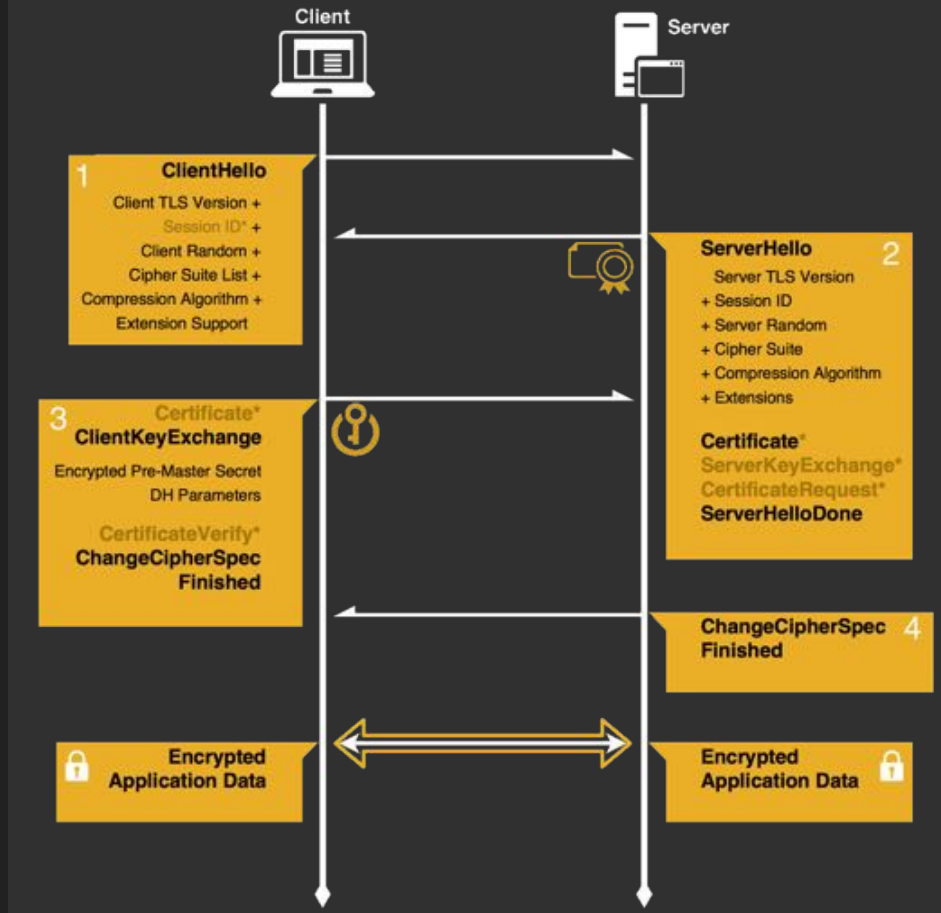
# How we get there


1. Briefest of TLS Refreshers
2. How Schannel Works
3. The Secrets : }
4. The Other Forensic Artifacts!
5. A live demo >.>

# A Disclaimer

- This is NOT an exploit
  - It’s the spec! :D
- Microsoft has done nothing 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...
- This talk has nothing to do with Chanel
  - Sorry Aine.

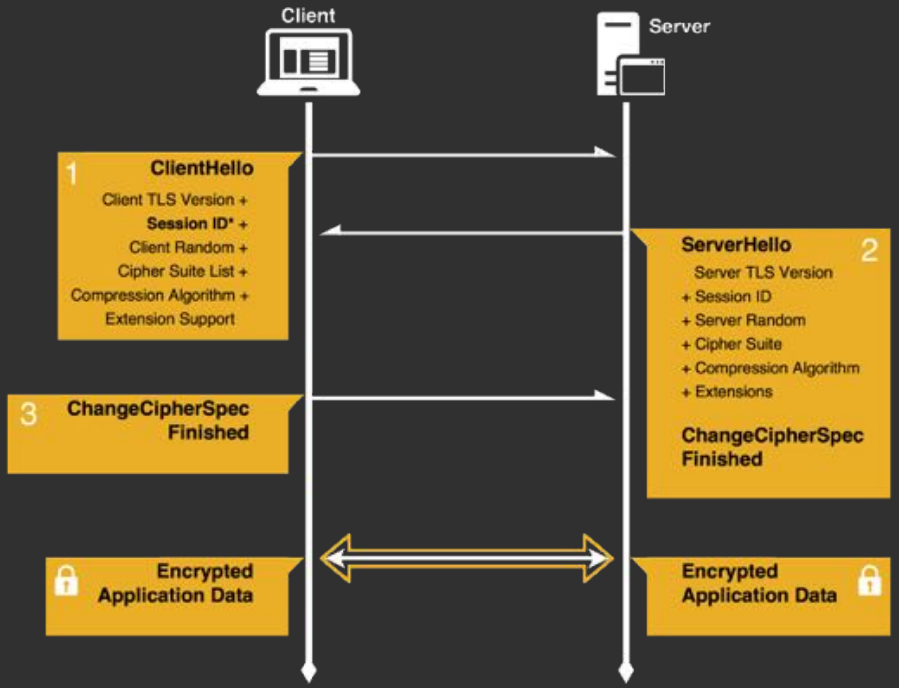
# The now infamous TLS Handshake



# The ~~now~~ ~~infamous~~ TL~~S~~ Handshake

DRi

or, Session Resumption



# Perfect Forward Secrecy < and what it means to TLS

What we *want* to do

- One time use keys, no sending secrets!

What TLS *actually* does

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

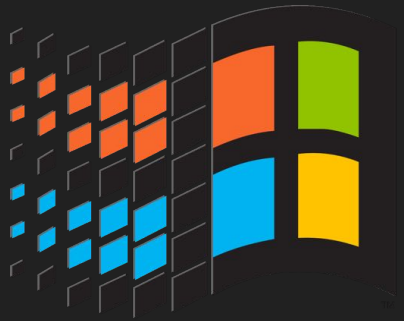
What implementations *also* do

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

See these papers: https://technet.microsoft.com/en-us/library/dn786429.aspx

# What's an Schannel?

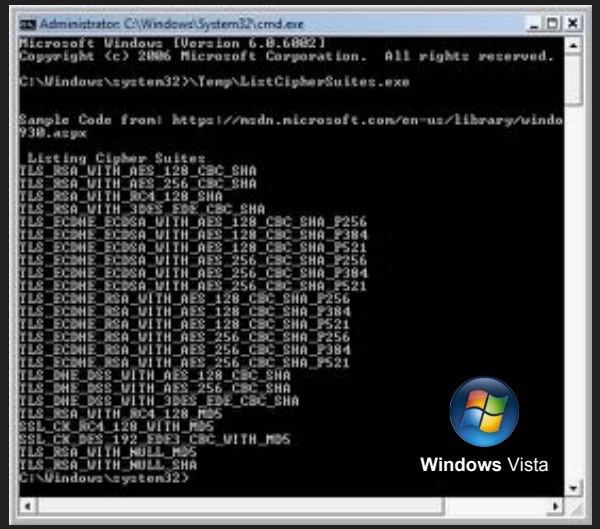
- It's TLS -> the Secure Channel for Windows!
- A library that gets loaded into the "key isolation process" **and** the "client" process
  - Technically a Security Support Provider (SSP)
- Spoiler: the key iso proc is LSASS

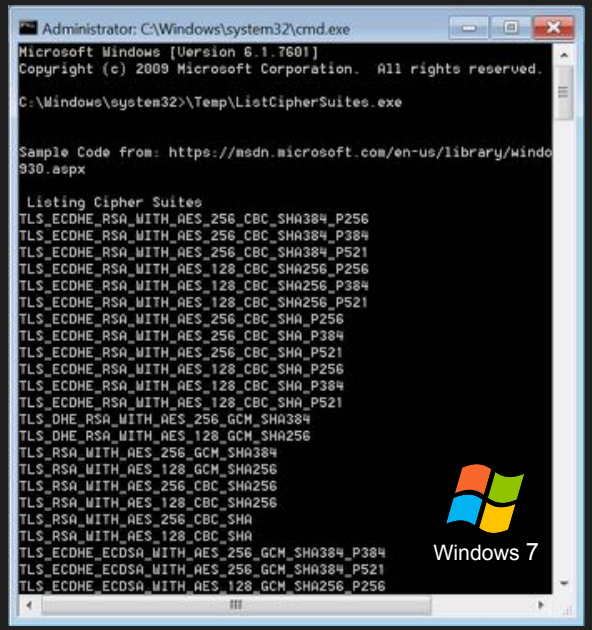


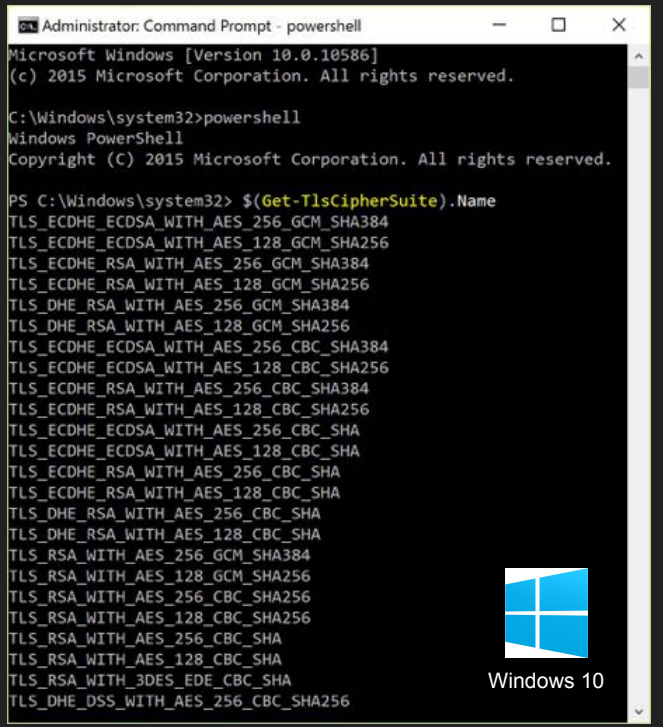
- Microsoft's CryptoAPI-Next Generation
- Introduced in Windows Vista
- Provides Common Criteria compliance
- Used to store secrets, also crypt them
  - The KSP & DPAPI for instance
- Important / reused keys are "isolated" from the less privileged/trusted "client" processes into the "key isolation process"
- Ncrypt is the "key storage router" and gateway to CNG Key Iso service

# What the CNG?!

# Schannel Cipher Suite Preferences




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

# Schannel

by the docs

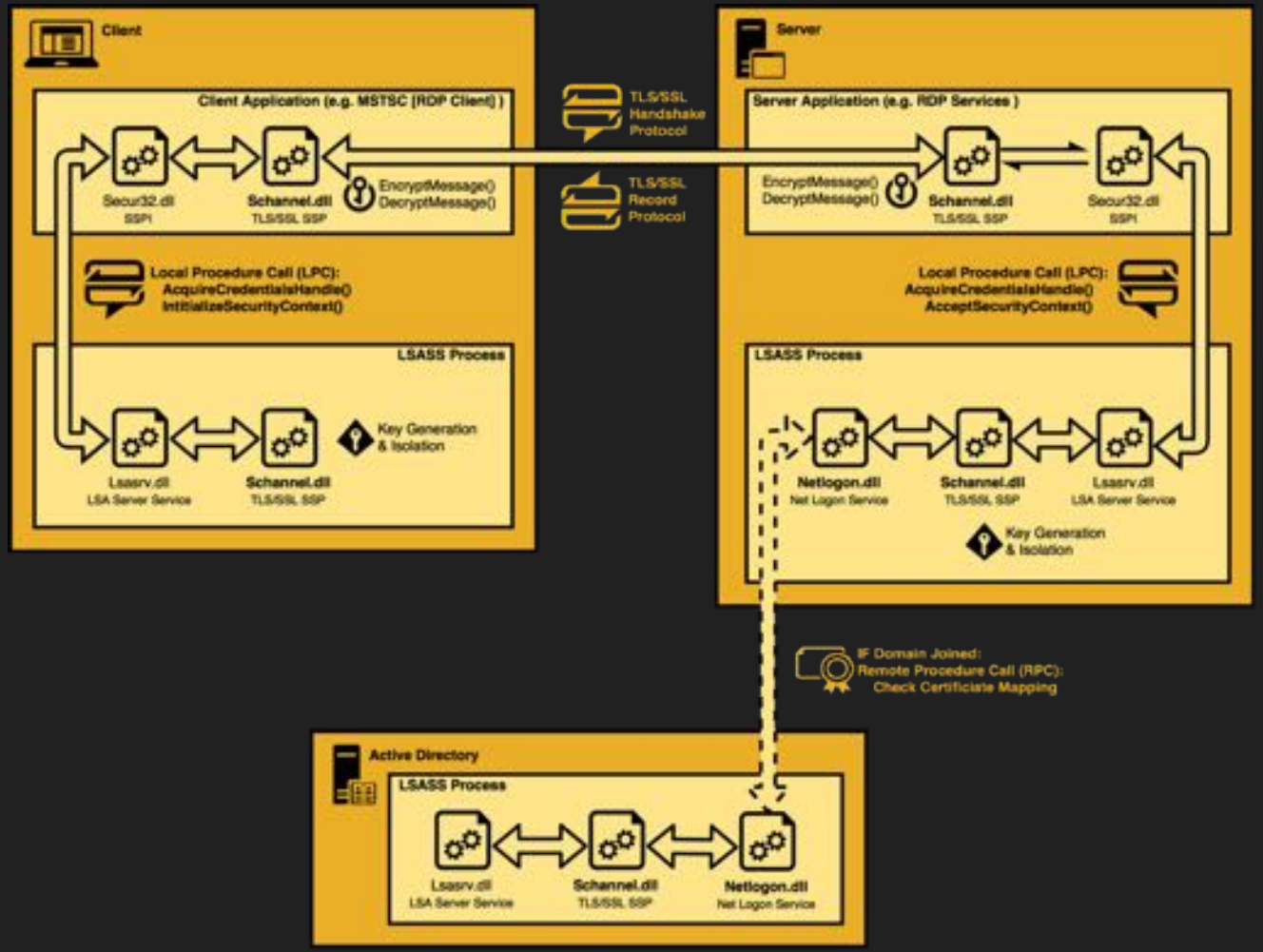



Diagram based on: https://technet.microsoft.com/en-us/library/dn786429.aspx

# CNG Key Isolation

by the docs

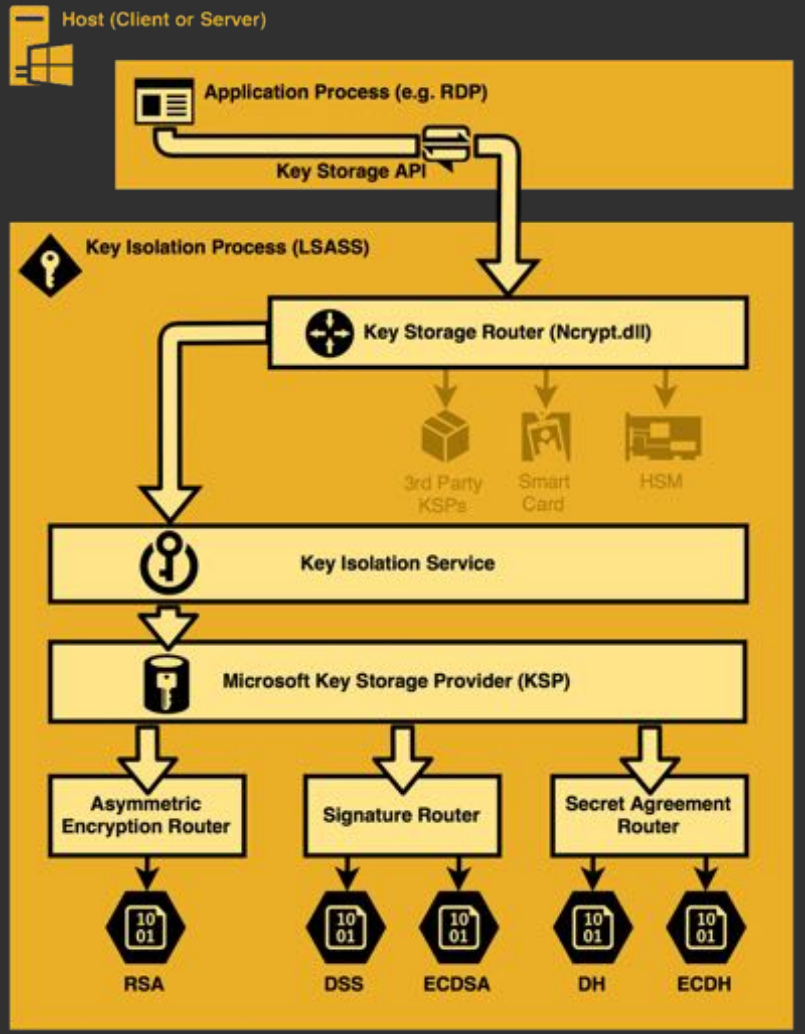



Diagram based on: https://msdn.microsoft.com/en-us/library/windows/desktop/bb204778.aspx

# Matching Session Keys

Basic Premis:

AES Keys are small and random

AES Key Schedules are larger and deterministic by design…they're a schedule.

Most implementations calculate schedule once and store it*

While a connection is active, both side NEED access to the symmetric keys used for encryption/verification

*Based on this paper: http://citp.princeton.edu/pub/coldboot.pdf

# Matching Session Keys

So I scanned LSASS for cross-matched AES key schedules on both hosts…

**And got nothing.**

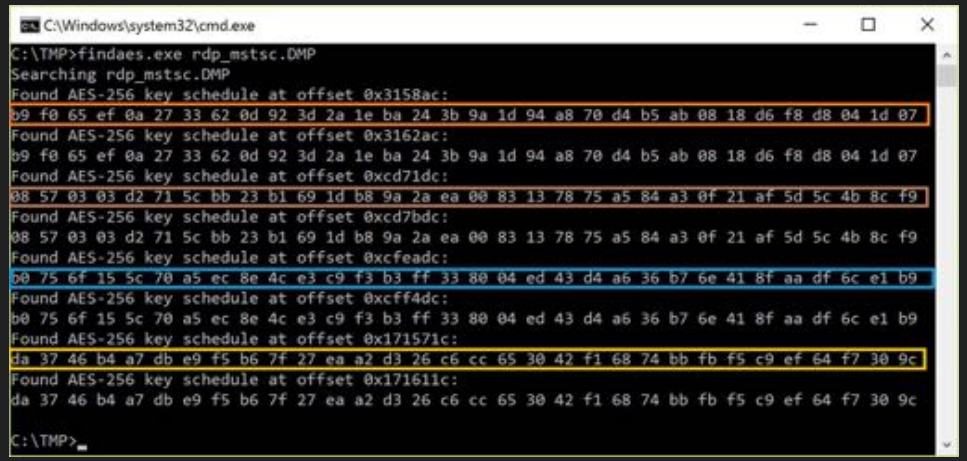
**Well, no matches anyway.**

*A friendly neighborhood P.S.A.*

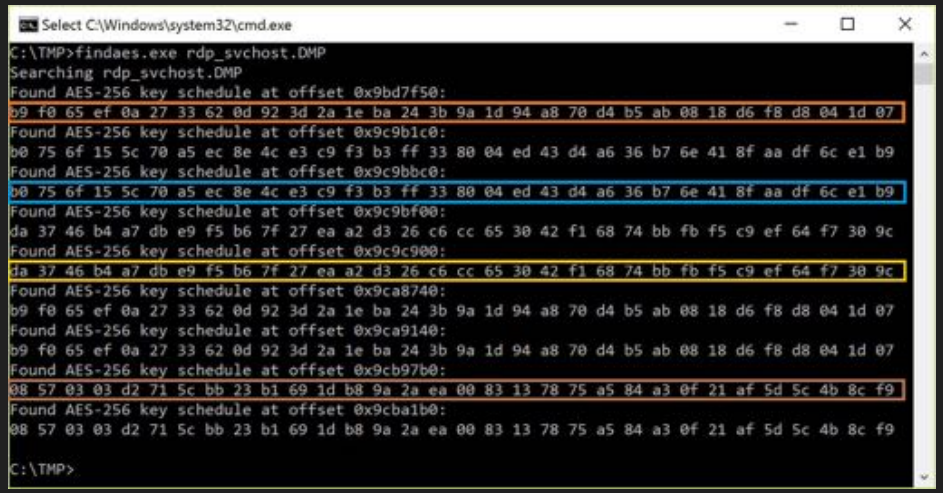

*This announcement brought to you by an hour of wasted time*

# Matching Session Keys

## RDP MSTSC AES Keys [Client]



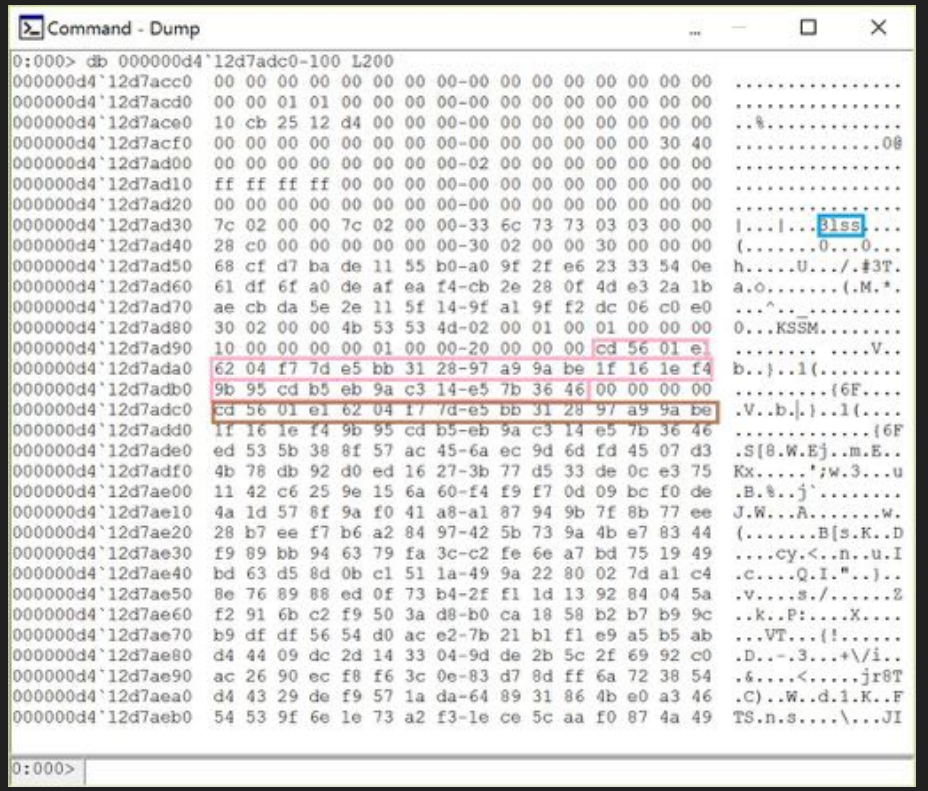

## RDP SVCHost AES Keys [Server]




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

# The Session Key Structure

- Notice the value “`3lss`”
  - “`3lss`” -> “`ssl3`”
  - Initially noticed while checking LSASS structs
- Structure is different in LSASS vs client process
- AES Key & Schedule highlighted
- Key and schedule appear multiple times in the same structure

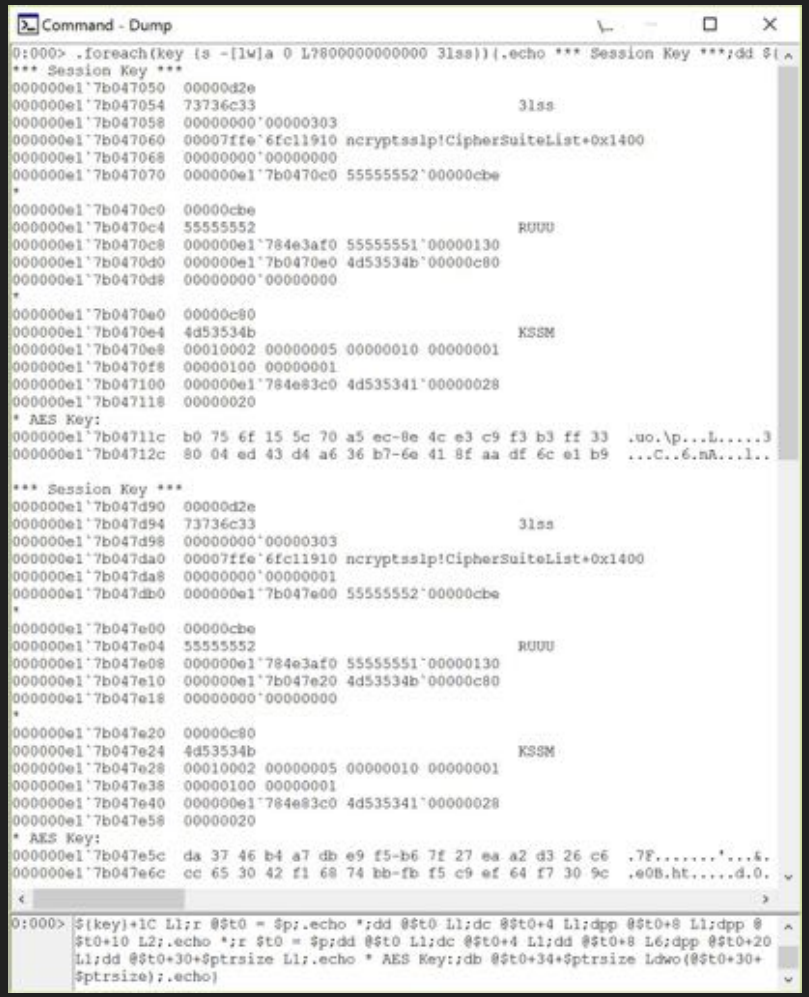


# The Session Key Structure

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

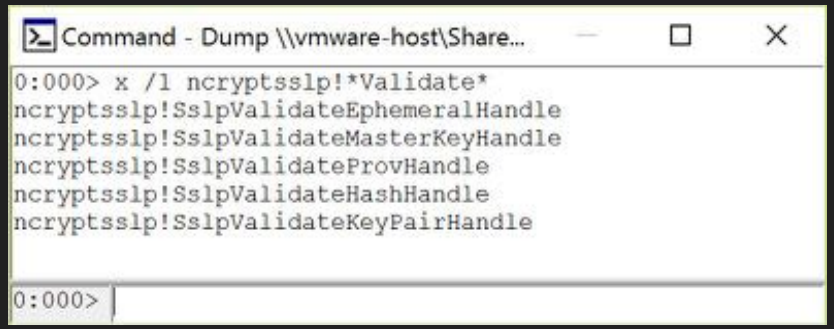
| _BCRYPT_KEY | |
|---|---|
| 4 | cbStructLength |
| 4 | dwMagic ["UUUR"] |
| 4/8 | pvBcryptProvider |
| 4/8 | pvBcryptSymmKey |

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



# The Ncrypt SSL Provider [ ncryptsslp.dll ]

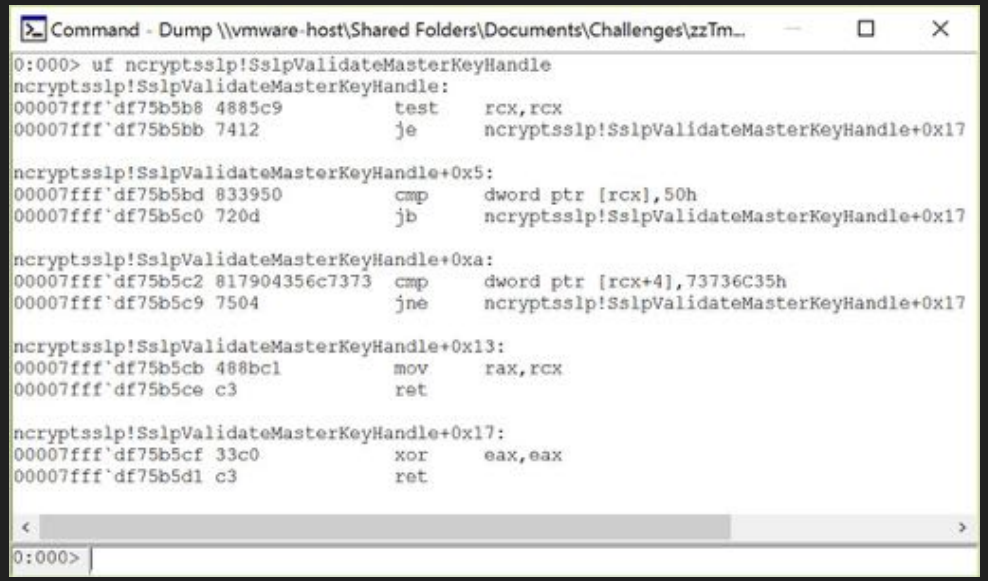
## Ncryptsslp Validation function Symbols




## 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

## Master Key Validation Function Disassembly




*All handles in this case are explicitly pointers

| SSL Magic | Size (x86) | Size (x64) | Validation Functions |
|---|---|---|---|
| ssl1 | 0xE4 | 0x130 | SslpValidateProvHandle |
| ssl2 | 0x24 | 0x30 | SslpValidateHashHandle |
| ssl3 | ? | ? | <none> |
| ssl4 | 0x18 | 0x20 | SslpValidateKeyPairHandle |
| ssl5 | 0x48 | 0x50 | SslpValidateMasterKeyHandle |
| ssl6 | 0x18 | 0x20 | SslpValidateEphemeralHandle |
| ssl7 | ? | ? | <none> |

## Ncryptsslp “ssl3” symbols*

```
>_ Command - Dump C:\Defcon\Exa...      - [] X

0:000> .foreach(magic {s -[1]a
00007fff`df750000 00007fff`df76f000 3lss}){ln
magic}
ncryptsslp!TlsGenerateSessionKeys+0x251
ncryptsslp!SPSslDecryptPacket+0x43
ncryptsslp!SPSslEncryptPacket+0x43
ncryptsslp!SPSslImportKey+0x19a
ncryptsslp!SPSslExportKey+0x76
ncryptsslp!SPSslFreeObject+0x1b
ncryptsslp!Ssl2GenerateSessionKeys+0x22c
ncryptsslp!Ssl2GenerateSessionKeys+0x294
```

`ssl3` = session key struct

## Ncryptsslp “ssl7” symbols*

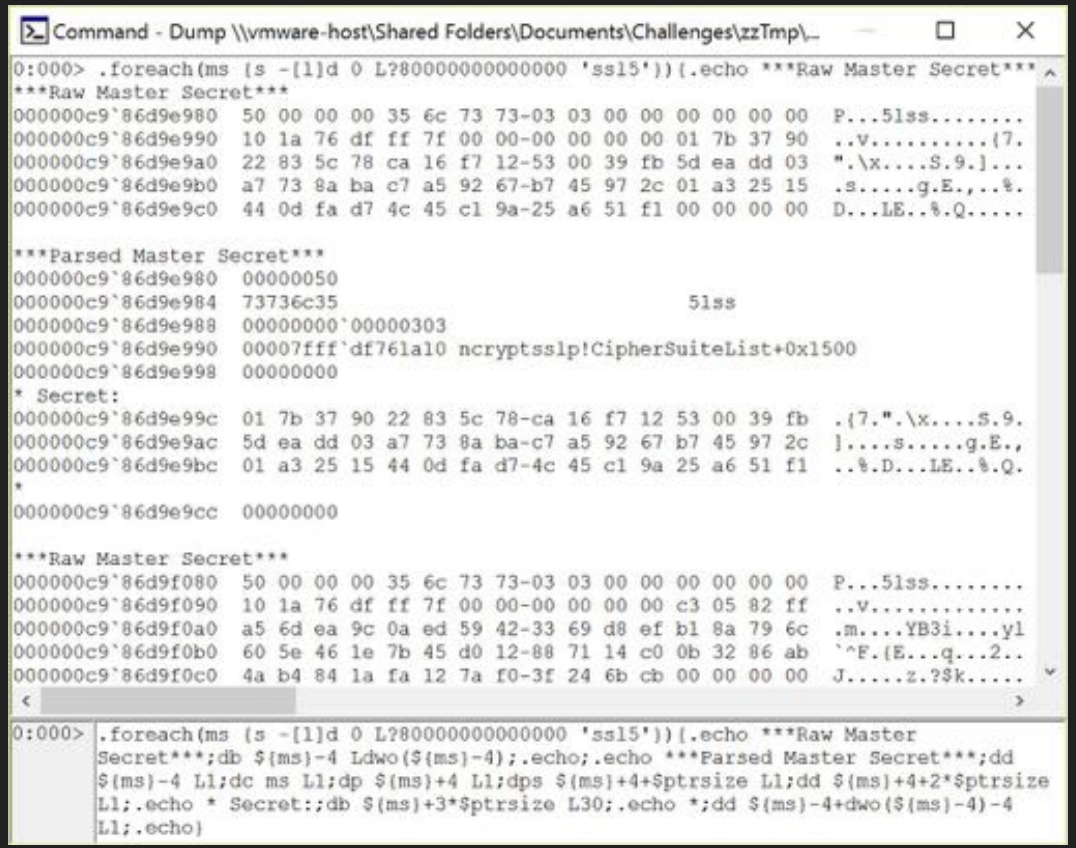
```
>_ Command - Dump C:\Defcon\Exa...      - [] X

0:000> lmm schannel
start             end                 module
name
00007fff`ed1e0000 00007fff`ed254000   schannel

0:000> .foreach(magic {s -[1]a
00007fff`df750000 00007fff`df76f000 7lss}){ln
magic}
ncryptsslp!SPSslGenerateMasterKey+0x75
ncryptsslp!SPSslGenerateMasterKey+0x5595
ncryptsslp!SPSslGeneratePreMasterKey+0x15e
ncryptsslp!TlsDecryptMasterKey+0x 6b
```

`ssl7` = pre-master secret struct?

*command output significantly reduced for brevity & clarity

# The Master Secret

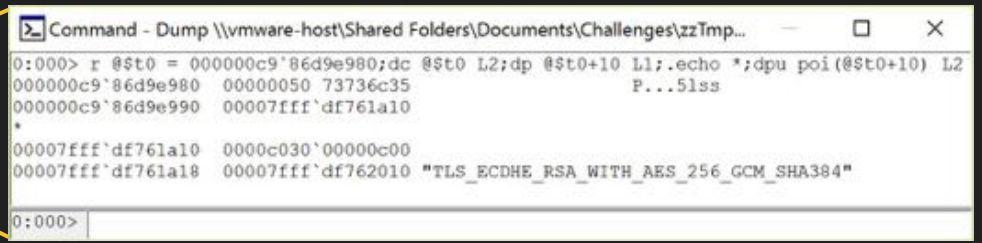
| _SSL_MASTER_SECRET | |
|---|---|
| 4 | cbStructLength |
| 4 | dwMagic [“ssl5”] |
| 4 | dwProtocolVersion |
| 0/4 | dwUnknown1* [alignment?] |
| 4/8 | pvCipherSuiteListEntry |
| 4 | bIsClientCache |
| 48 | rgbMasterSecret |
| 4 | dwUnknown2 [reserved?] |




*Not present in x86 - either padding or part of previous member

# The 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?] |




*Not present in x86 - either padding or part of previous member

# 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

At this point, we can identify and decrypt sessions robustly.

# Schannel $

Under the covers


```
'_SSL_SESSION_CACHE_CLIENT_ITEM': [ 0x140, {
    'Vftable': [0x0, ['pointer64', ['void']]],
    'NcryptKey': [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']],
    }],
```

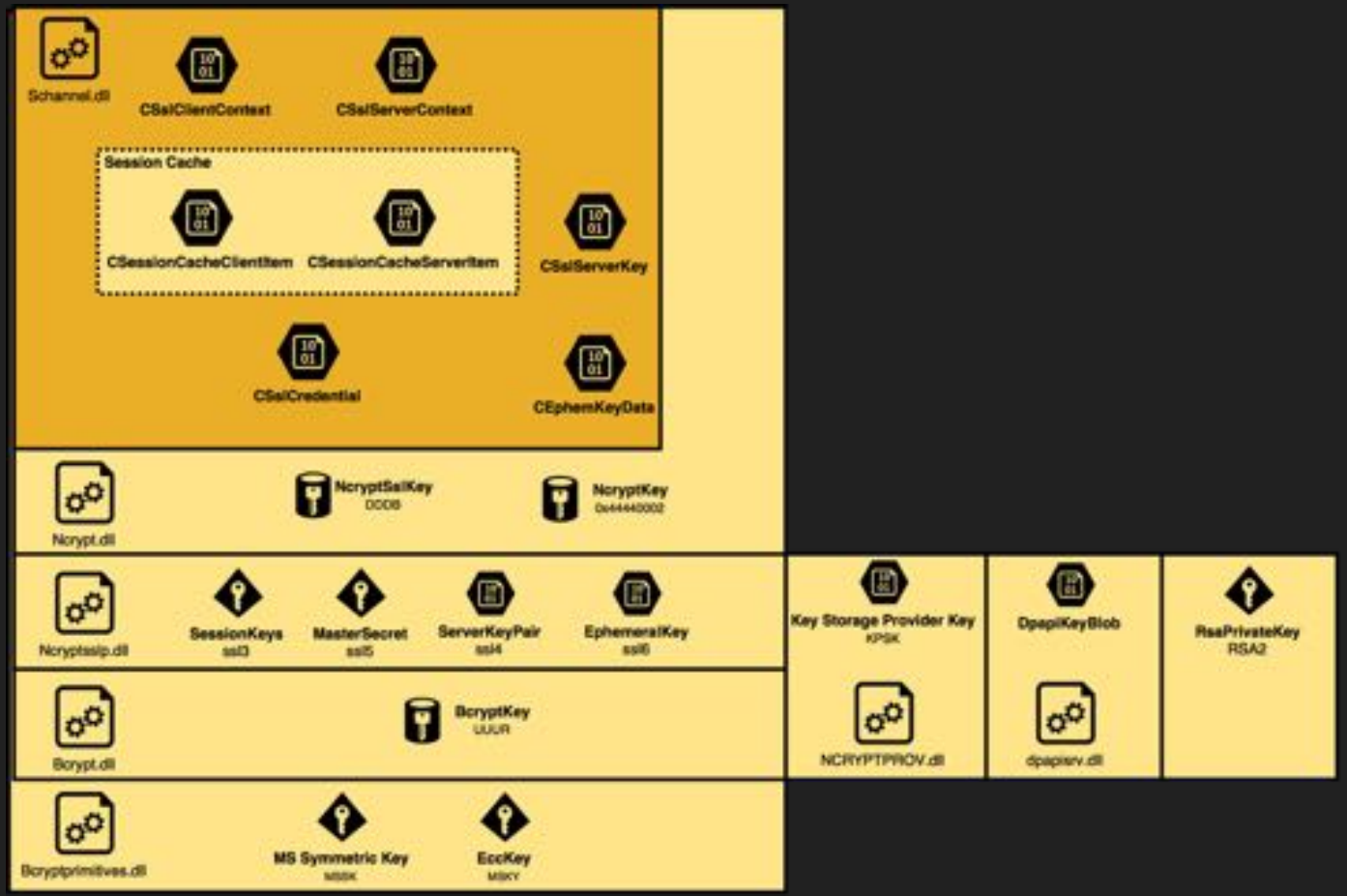
# Schannel $

Under the covers


```
'_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']]],
    }],
```

# Schannel $

Under the covers



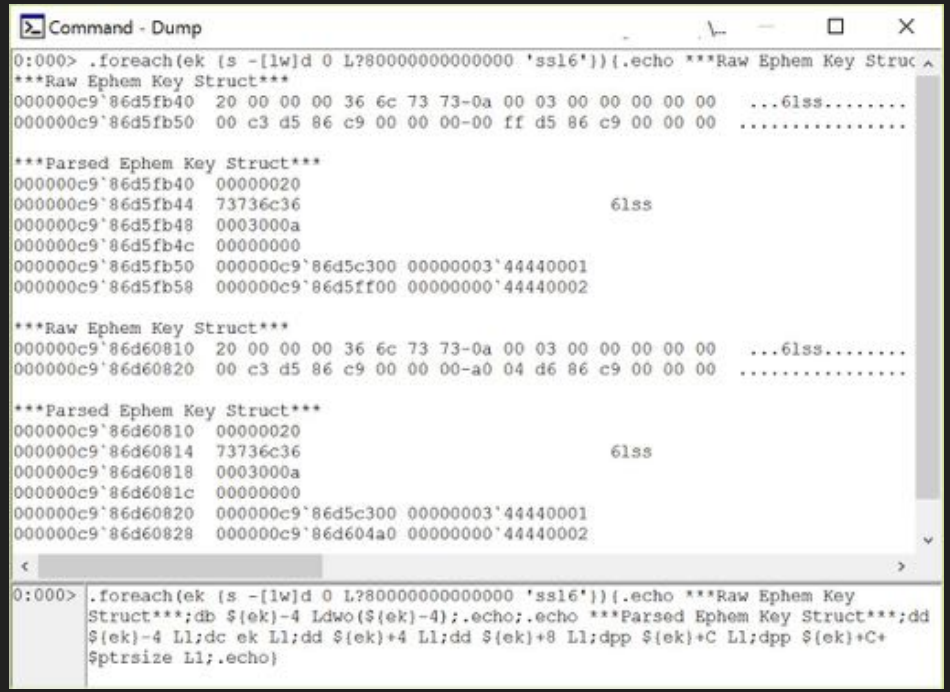
# The Key Pairs

- The Server & Ephemeral Key Pairs use an identical structure
- 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*


*Not to be confused with an `NcryptSslKey`struct

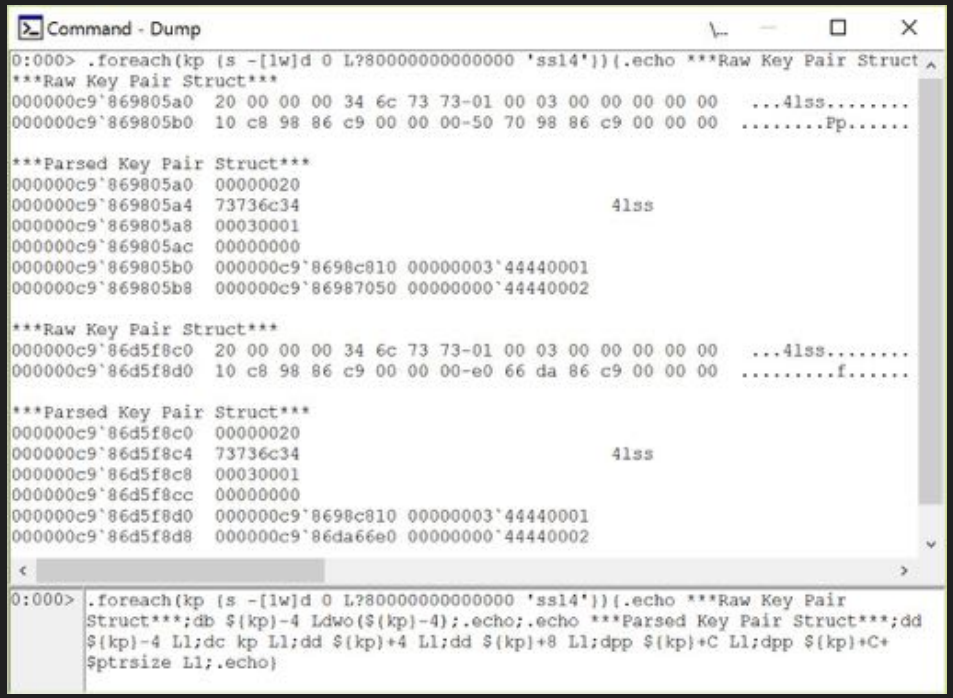
# The Ephemeral Key Data

- Public Key is referenced by schannel! CEphemKeyData
- Private Key is not stored in natively usable format, but is accessible
- The KPSK structure pointed to references another structure with the magic “MSKY” that appears to be the EccKey structure
- The KPSK structure has details about the curve selection / other valuable info



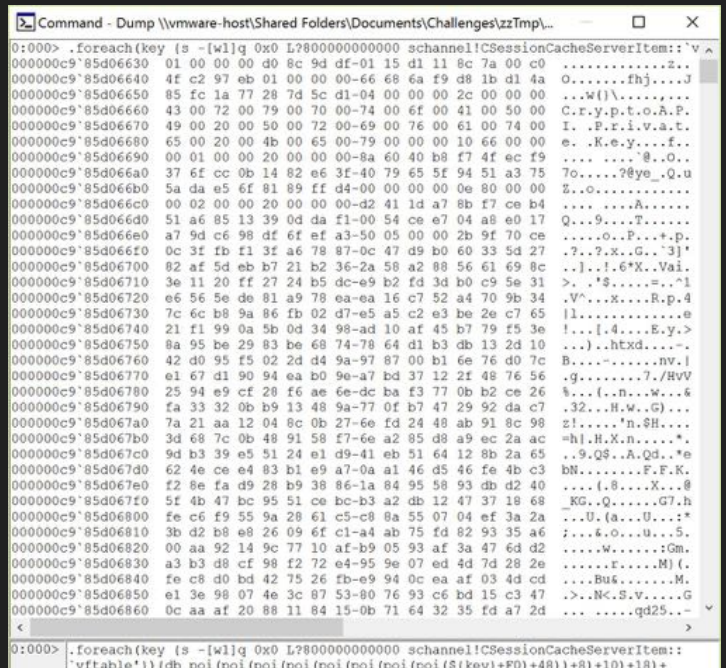
# The Server (RSA) Private Key

- KSPK structure pointed to by the server key mostly resembles the file from disk
- The public information is parsed into fields
- The DPAPI protected private key blob is loaded into memory



# The Server (RSA) Private Key

```
defc0n% cd ./ProgramData/RSA/MachineKeys
defc0n% xxd -g 1 -s 0x165 f686aace6942fb7f7ceb231212eef4a4_7496afd3-d13f-4cf7-b6
d9-ca3d0c3ff959
0000165: 01 00 00 00 d0 8c 9d df 01 15 d1 11 8c 7a 00 c0  .............z..
0000175: 4f c2 97 eb 01 00 00 00 66 68 6a f9 d8 1b d1 4a  O.......fhj....J
0000185: 85 fc 1a 77 28 7d 5c d1 04 00 00 00 2c 00 00 00  ...w(}\.....,...
0000195: 43 00 72 00 79 00 70 00 74 00 6f 00 41 00 50 00  C.r.y.p.t.o.A.P.
00001a5: 49 00 20 00 50 00 72 00 69 00 76 00 61 00 74 00  I. .P.r.i.v.a.t.
00001b5: 65 00 20 00 4b 00 65 00 79 00 00 00 10 66 00 00  e. .K.e.y....f..
00001c5: 00 01 00 00 20 00 00 00 8a 60 40 b8 f7 4f ec f9  .... ....`@..O..
00001d5: 37 6f cc 0b 14 82 e6 3f 40 79 65 5f 94 51 a3 75  7o.....?@ye_.Q.u
00001e5: 5a da e5 6f 81 89 ff d4 00 00 00 00 0e 80 00 00  Z..o............
00001f5: 00 02 00 00 20 00 00 00 d2 41 1d a7 8b f7 ce b4  .... ....A......
0000205: 51 a6 85 13 39 0d da f1 00 54 ce e7 04 a8 e0 17  Q...9....T......
0000215: a7 9d c6 98 df 6f ef a3 50 05 00 00 2b 9f 70 ce  .....o..P...+.p.
0000225: 0c 3f fb f1 3f a6 78 87 0c 47 d9 b0 60 33 5d 27  .?..?.x..G..`3]'
0000235: 82 af 5d eb b7 21 b2 36 2a 58 a2 88 56 61 69 8c  ..]..!.6*X..Vai.
0000245: 3e 11 20 ff 27 24 b5 dc e9 b2 fd 3d b0 c9 5e 31  >. .'$.....=..^1
0000255: e6 56 5e de 81 a9 78 ea ea 16 c7 52 a4 70 9b 34  .V^...x....R.p.4
0000265: 7c 6c b8 9a 86 fb 02 d7 e5 a5 c2 e3 be 2e c7 65  |l.............e
0000275: 21 f1 99 0a 5b 0d 34 98 ad 10 af 45 b7 79 f5 3e  !...[.4....E.y.>
0000285: 8a 95 be 29 83 be 68 74 78 64 d1 b3 db 13 2d 10  ...)..htxd....-.
0000295: 42 d0 95 f5 02 2d d4 9a 97 87 00 b1 6e 76 d0 7c  B....-......nv.|
00002a5: e1 67 d1 90 94 ea b0 9e a7 bd 37 12 2f 48 76 56  .g........7./HvV
00002b5: 25 94 e9 cf 28 f6 ae 6e dc ba f3 77 0b b2 ce 26  %...(..n...w...&
00002c5: fa 33 32 0b b9 13 48 9a 77 0f b7 47 29 92 da c7  .32...H.w..G)...
00002d5: 7a 21 aa 12 04 8c 0b 27 6e fd 24 48 ab 91 8c 98  z!.....'n.$H....
00002e5: 3d 68 7c 0b 48 91 58 f7 6e a2 85 d8 a9 ec 2a ac  =h|.H.X.n.....*.
00002f5: 9d b3 39 e5 51 24 e1 d9 41 eb 51 64 12 8b 2a 65  ..9.Q$..A.Qd..*e
0000305: 62 4e ce e4 83 b1 e9 a7 0a a1 46 d5 46 fe 4b c3  bN........F.F.K.
0000315: f2 8e fa d9 28 b9 38 86 1a 84 95 58 93 db d2 40  ....(.8....X...@
0000325: 5f 4b 47 bc 95 51 ce bc b3 a2 db 12 47 37 18 68  _KG..Q......G7.h
0000335: fe c6 f9 55 9a 28 61 c5 c8 8a 55 07 04 ef 3a 2a  ...U.(a...U...:*
0000345: 3b d2 b8 e8 26 09 6f c1 a4 ab 75 fd 82 93 35 a6  ;...&.o...u...5.
0000355: 00 aa 92 14 9c 77 10 af b9 05 93 af 3a 47 6d d2  .....w......:Gm.
0000365: a3 b3 d8 cf 98 f2 72 e4 95 9e 07 ed 4d 7d 28 2e  ......r.....M}(.
0000375: fe c8 d0 bd 42 75 26 fb e9 94 0c ea af 03 4d cd  ....Bu&.......M.
0000385: e1 3e 98 07 4e 3c 87 53 80 76 93 c6 bd 15 c3 47  .>..N<.S.v.....G
0000395: 0c aa af 20 88 11 84 15 0b 71 64 32 35 fd a7 2d  ... .....qd25..-
```



# Windows Vista

- CNG was introduced in Vista
- The Vista cache is different
- It’s kinda proto-CNG
- Prior to Ncryptsslp (Sslp functions are in Ncrypt)
- Instead of Classes, the cache is just a doubly-linked list
- No RFC5088 support (no tickets)


```
'_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']]],
    }],
```

# Windows Vista

# The Forensic Context

- Active Connection = Security Context
- ProcessID for client process stored
- Server Name Indicator (SNI) stored in the cache as well
- Cache Lifetime of 10 hours
- Session IDs are arbitrary, but not always random
  - Schannel is the perfect example, can be fingerprinted
- If the system is a client, why would it have a server cache?
  - RDP for one, almost guaranteed to live 10 hours (unless there are 20,000 connections afterward)

Global Schannel Variables of Significance:

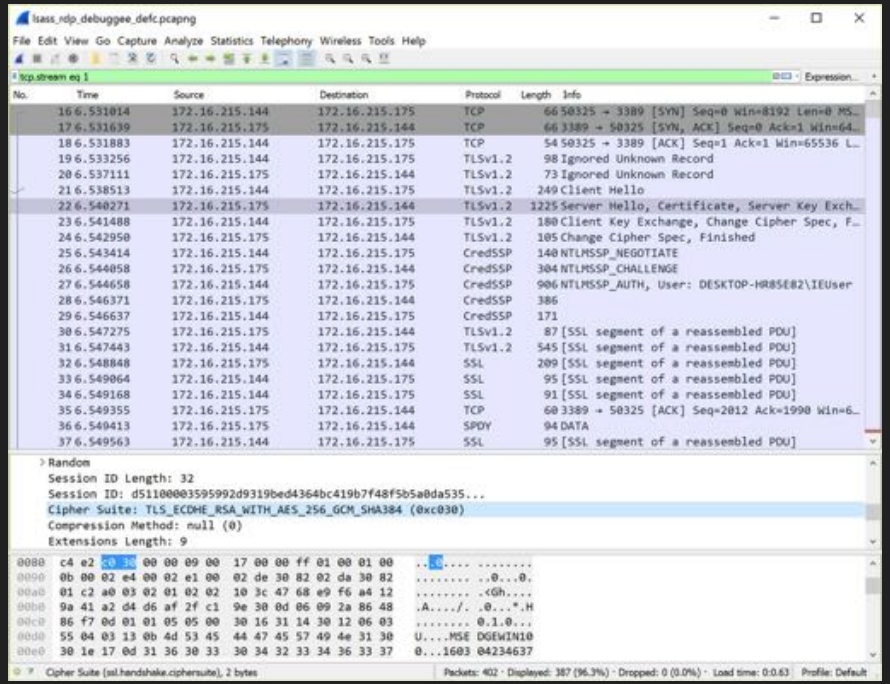
```
schannel!CSslGlobals::m_dwMaximumEntries
schannel!CSslGlobals::m_dwClientLifespan
schannel!CSslGlobals::m_dwServerLifespan
schannel!CSslGlobals::m_dwSessionTicketLifespan
```

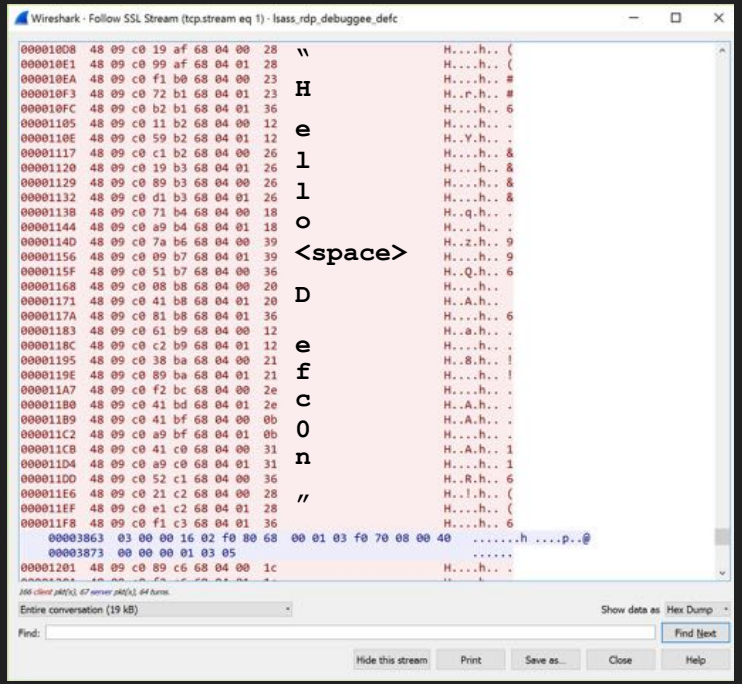
# Extracting the Secrets

- Volatility & Rekall plugins
- By default (no args):
  - will automatically find lsass
  - will scan the heap
    - Can be configured to scan Writeable VADs, or full VAS
  - dumps to stdout in wireshark format
    - Can dump verbose object as json
- Hoping to have functionality integrated into PowerShell module soon
  - Got busy : <

```
defc0n% 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
```

# Decrypting an RDP Session (Ephemeral 🔑 XCHG)




Keycodes from: https://msdn.microsoft.com/en-us/library/aa299374.aspx

# Decrypting an RDP Session (Ephemeral 🔑 XCHG)

DEMO TIME

# Fin

@TinRabbit_

# 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