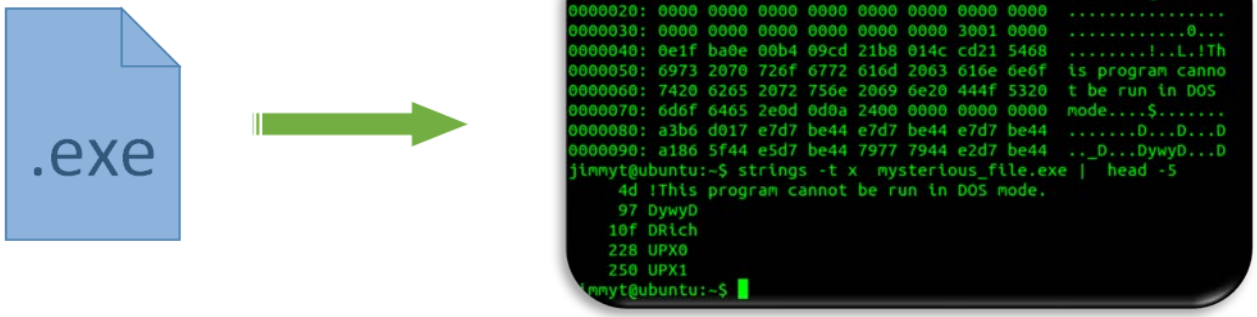
New York University Abu Dhabi
Modern Microprocessor Architectures Lab
nyuad.nyu.edu/momalab

SGXCrypter: IP Protection for Portable Executables using Intel's SGX Technology

Dimitrios Tychalas
Nektarios Georgios Tsoutsos
Michail Maniatakos

Motivation

- Software IP theft
  - 400 billion dollars lost in 2014[1]
  - >50% gamers were using pirated software in 2012[2]
- IP theft typically achieved through:
  - Copying
  - Reverse-Engineering
    - Achievable through binary static analysis



[1] B. Kerr, "Software Piracy Hampering U.S. Manufacturing," http://www.manufacturing:net/news/, 2014. [Accessed: May 10, 2016]
[2] Z. Epstein, "Arrest half the world: More than 50% of computer users pirate software, study finds," http://bgr.com/2012/05/31/digital-piracy-bsa-study-2011, 2011 [Accessed: January 10, 2017]

Solutions

- Code obfuscation
  - Make the code confusing
  - Does not stop motivated attackers
- Break the attacker's tools
  - Study the tool and adapt the code to defend against it
  - Many tools that keep evolving
- Packing
  - Alter code and embed code to reverse the alteration in runtime

Packing

- Packing software (packers)
  - Typical packing → Compression
  - Secure packing → Encryption
    - Also called Crypter
- Unpacking Stub
  - Lucrative target for reverse-engineers
  - Contains the unpacking algorithm or decryption key

Crypter overview

Original Executable

```
401196: 87 8f 4d c5 93 39    xchg   %ecx,0x3993c54d(%edi)
40119c: 6e                    outsb  %ds:(%esi),(%dx)
40119d: 15 50 73 0f 6e        adc    $0x6e0f7350,%eax
4011a2: a9 0e 13 e3 d1        test   $0xd1e3130e,%eax
4011a7: 7c ab                 jl     0x401154
4011a9: 20 8f 5c 0a 5d 3c     and    %cl,0x3c5d0a5c(%edi)
4011af:  3a 34 3d 87 58 68 6e      cmp    0x6e685887(,%edi,1),%dh
4011b6: ae                    scas   %es:(%edi),%al
4011b7: a3 70 03 d3 93        mov    %eax,0x93d30370
4011bc: e4 b9                 in     $0xb9,%al
4011be: ae                    scas   %es:(%edi),%al
4011bf:  99                   cltd
4011c0: c1 39 4f              sarl   $0x4f,(%ecx)
4011c3: 3b c2                 cmp    %edx,%eax
4011c5: d3 a8 bf a9 7a db     shrl   %cl,-0x24855641(%eax)
4011cb: df a2 54 d6 01 6c     fbld   0x6c01d654(%edx)
4011d1: 4e                    dec    %esi
4011d2: 76 66                 jbe    0x40123a
4011d4: d7                    xlat   %ds:(%ebx)
4011d5: 6d                    insl   (%dx),%es:(%edi)
4011d6: f5                    cmc
4011d7: 2e                    cs
4011d8: 2f                    das
4011d9: 7c 95                 jl     0x401170
4011db: dc ff                 fdivr  %st,%st(7)
4011dd: 5d                    pop    %ebp
```

Crypter

"Packed" File

Decryption Stub

```
void aes_decrypt_cbc(void)
{
  // 128bit key
  uint8_t key[16] =  { 0x2b,  0x7e,  0x15,  0x16, 0x28,
                       0xae,  0xd2,  0xa6,  0xab, 0xf7,
                       0x15,  0x88,  0x09,  0xcf,  0x4f,
                       0x3c };
```

Encrypted Executable

```
2e0e 4889 e5df 2f51 2764 e686 681c ba2d
8b34 28e4 7a85 5c4d 8e01 4cb8 1ed6 cf61
5803 ebcc 4d8a 0e17 9ff7 69a3 74fe 4de6
3838 f63b fb32 60ef ccb4 beef 82c2 ce83
0398 6d89 017e c250 11ae f9e9 8d20 d566
e040 5792 b92b e2a9 e7eb 4050 7335 e93a
52b6 511b 2ffd bd4e 1e34 0d96 0bb5 6fe1
8caf d47b 9103 6c5d 9437 f4b5 d9a7 78cc
0206 dd7f 92f9 b32d 3247 109d 9f91 9a03
c2d9 8c4f b37a 300e ab12 e74f f6c0 0b5d
b3e4 28e9 beb8 5d9b f4e0 c3e6 2c8e 8cfd
0fbd 8820 7d91 920c f3da 162f 7162 422e
28d1 e922 532b 6e38 9c89 e4dc ed8e 8815
35c6 66c7 0c47 56fb 3988 f0ee d731 62ee
a64e 5be1 0d30 8a3e b6a9 d98d 34be bda8
80d6 c05e e9db c7c1 04c2 c435 393c da88
340d bad5 e25c 4900 32df a5ec 4e98 f807
d467 b7ff 3e37 278c f94c 6f96 b8dd b8f8
2a1e 6de4 b77f f57c 72f0 92f5 4940 a3f9
f17b f843 4f34 6a9b 292c 46d0 431d c879
fb1b dce2 bd2a 6587 0b5b d759 6c89 e226
25a8 267c 91b8 4172 9f5e 3f16 7ad6 303c
```

State-of-the-art available Crypters

- Yoda's Protector
  - Encryption + Compression
- CrypterBinder
  - Encryption + File binding
- Aegis
  - Custom decryption stub
- Hyperion
  - Crypter developed to avoid disclosing the decryption key
  - Brute-forcing the key during execution
  - High performance overhead

Our contribution

- SGXCrypter: A tool for protecting software IP
  - Strong encryption
  - Decryption key not embedded in packed binary
  - Protects against dynamic analysis leveraging Intel SGX
- Safe key → Secure binary

Presentation outline

- Motivation
- Preliminaries
- SGXCrypter Architecture
- SGXCrypter Security Assessment
- Experimental Results
- Conclusions

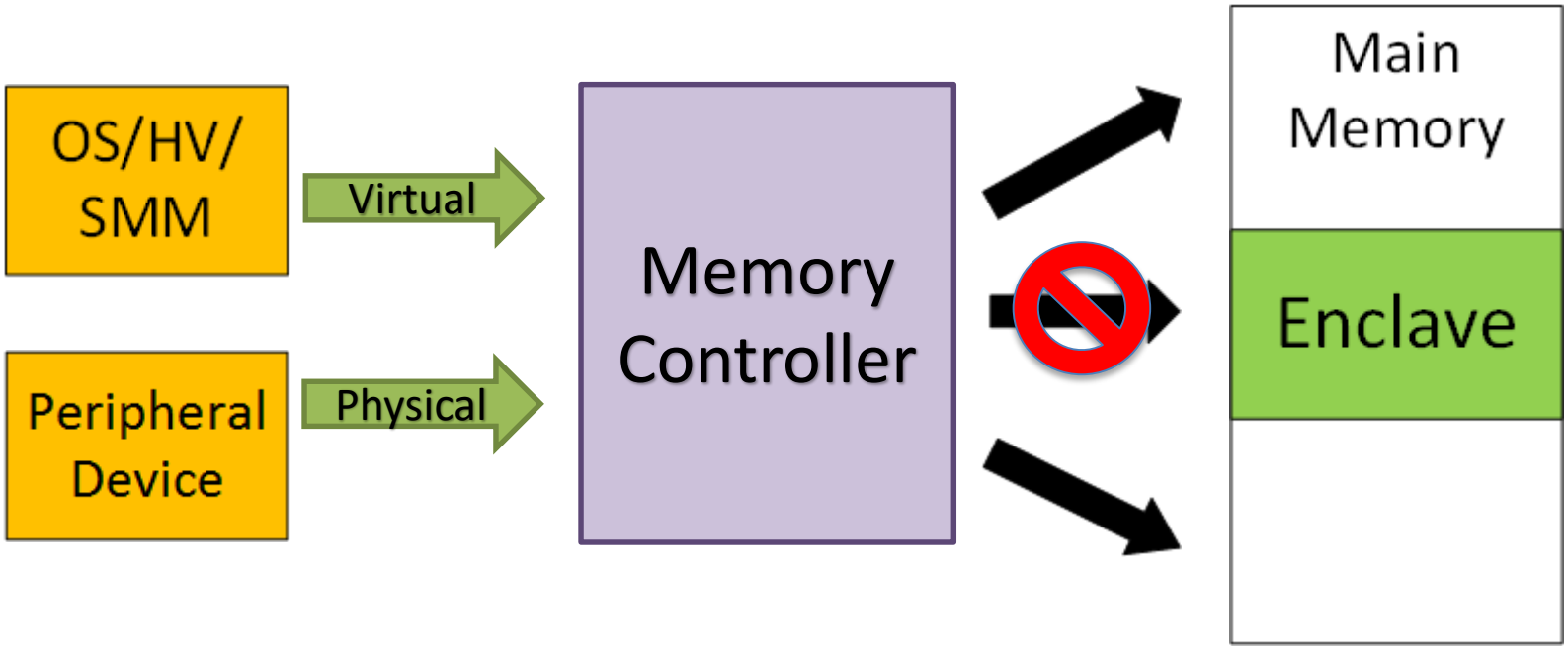
Preliminaries

SGX - Outline (1/2)

- Secure application execution – Enclave
  - Built as a .dll
- Enclave load and retire via special instructions
  - No OS/Hypervisor mediation
- Isolated execution
  - Memory Controller rejects all access

Preliminaries

SGX - Memory access



Preliminaries

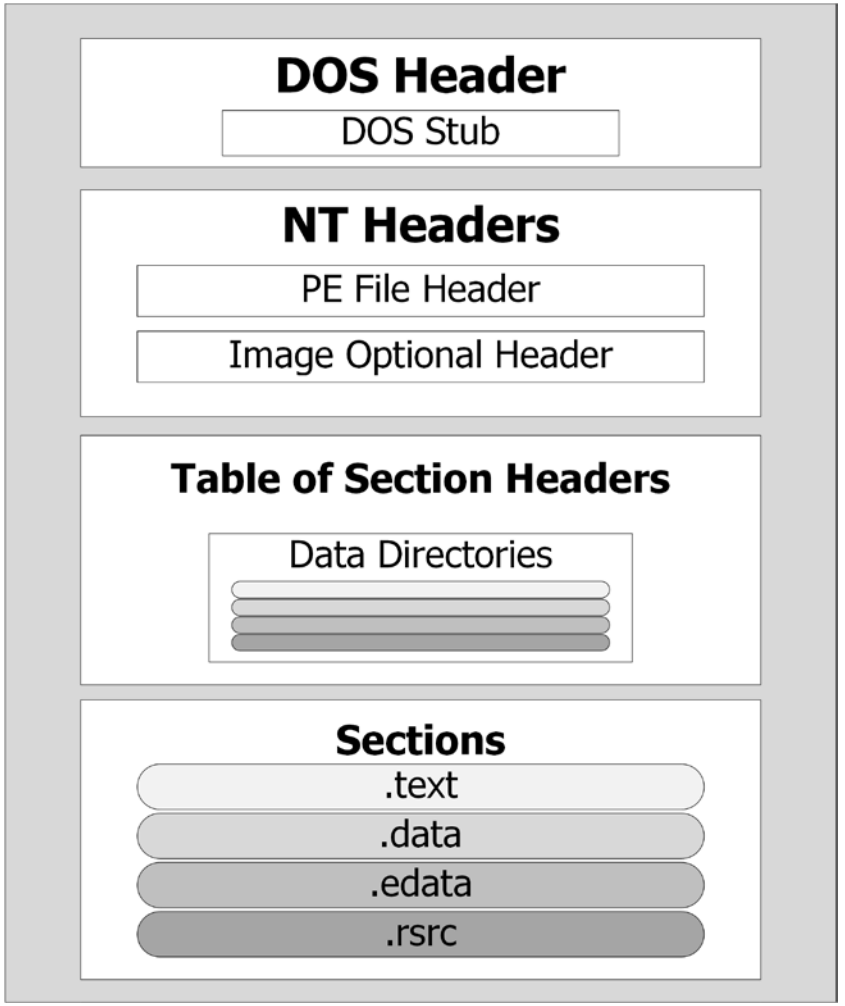
SGX - Outline (2/2)

- Memory encryption
  - Enclave data are encrypted when idle
  - Handled by the Memory Encryption Engine
- Secure enclave communication
  - Diffie-Hellman key exchange
  - Each enclave identified by a unique hash
  - Remote/Local attestation based on this hash

Preliminaries

Portable Executable (PE) files

- Portable Executable
  - Windows OS main executable format
- PE File Header
  - Describes what the rest of the file looks like
  - Essential for building thread context
- Sections
  - .text for code
  - .data for program data
  - .rsrc for resources (like other executables)

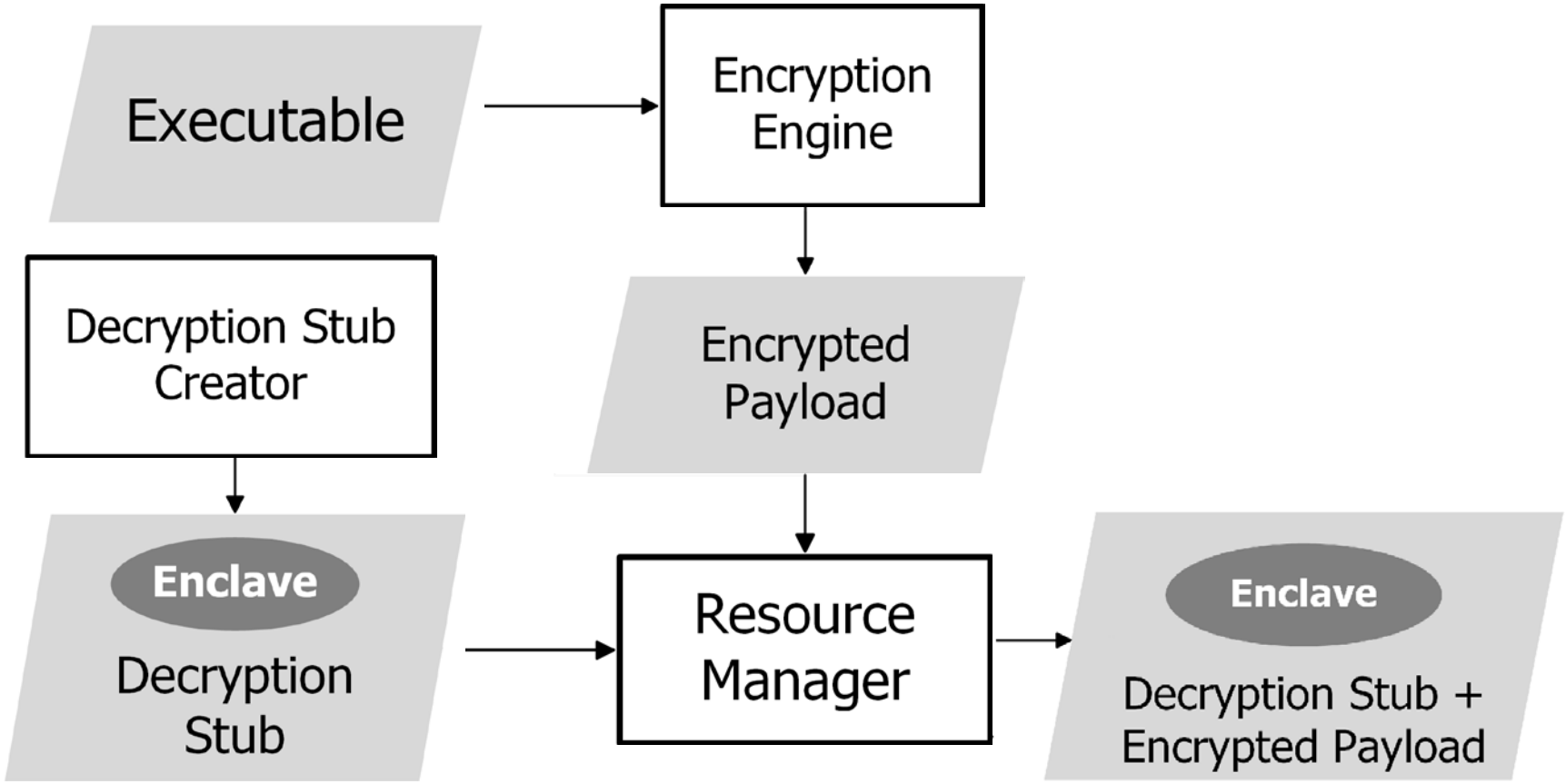


Presentation Outline

- Motivation
- Preliminaries
- SGXCrypter Architecture
- SGXCrypter Security Assessment
- Experimental Results
- Conclusions

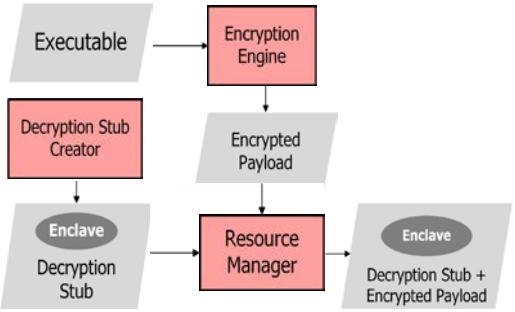
SGXCrypter Architecture

Block diagram



SGXCrypter Architecture

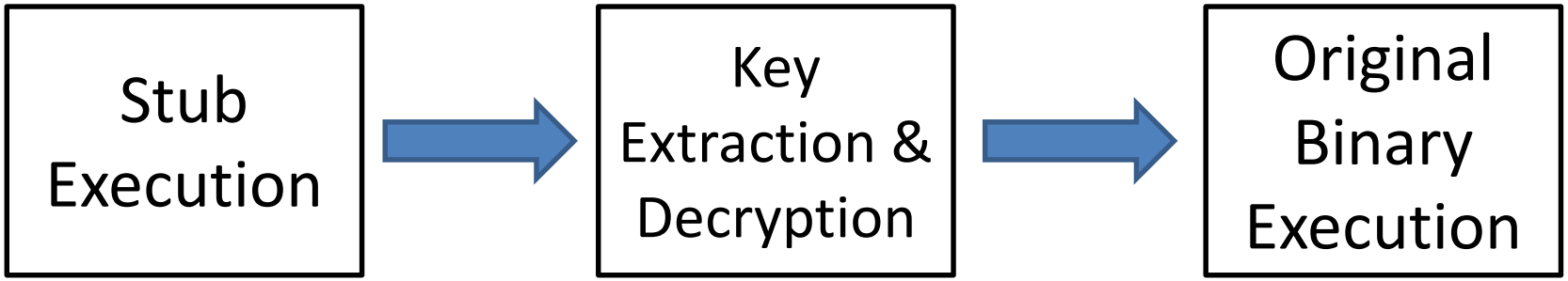
Blocks analyzed



- Encryption engine
  - Straightforward encryption of the target PE
  - AES-256 scheme chosen for this function
- Decryption stub creator
  - Creates the decryption stub
    - Handles the decryption of the encrypted PE (built as an SGX enclave to ensure isolated execution)
    - Creates a thread for the original PE
    - Executes the decrypted program
- Resource Manager
  - Combines the two files
  - Encrypted payload added as a resource

SGXCrypter Execution

- First step: Stub execution
  - Prepares the encrypted binary for decryption
- Second step: Key extraction & decryption
- Third step : Original binary execution
  - Original binary cannot be executed automatically



Execution

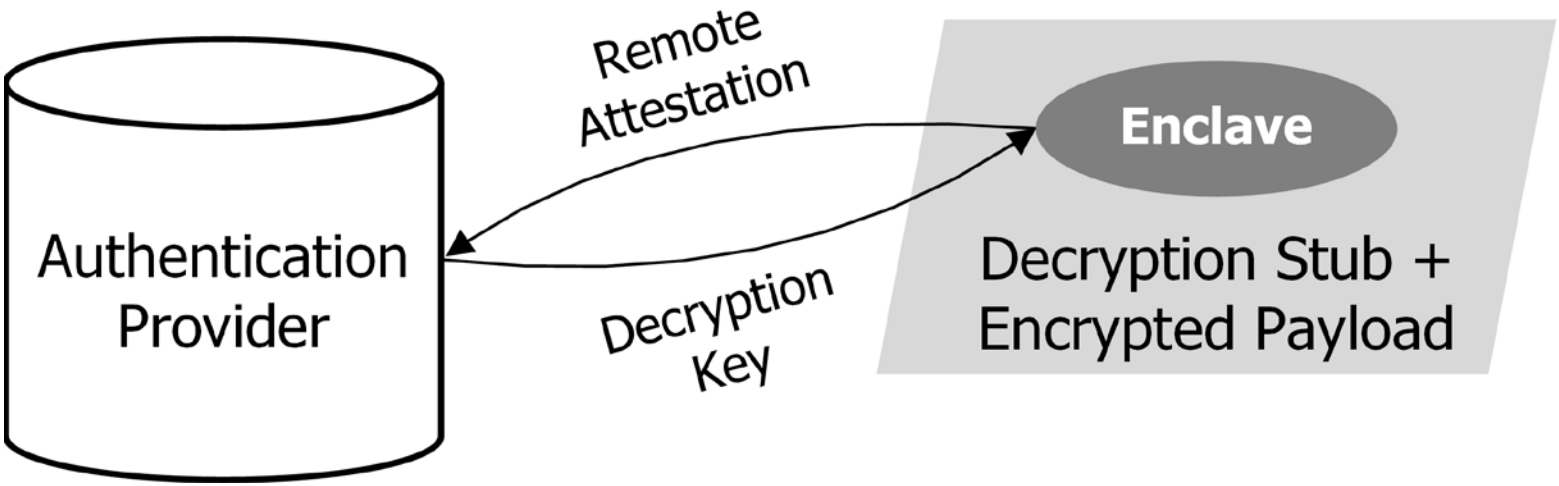
Stub execution

- Pointer for encrypted payload is fetched
  - Based on the way it was embedded in the decryption stub
- Encrypted data are handed to the enclave for decryption
  - 16-byte blocks
  - Copied into a buffer and then the enclave memory

Execution

Key extraction

- Decryption key is fetched
  - Key existing in separate server
  - Authorized via remote attestation
  - Receive key through a secure channel
- Decryption commences
  - Decrypted data replace original in main memory



Execution

Original binary execution

- Step 1: Locate DOS header
  - Contains pointers to the PE header
- Step 2: Build thread context via the PE header
  - Fetch metadata
  - Set memory protection
- Step 3: Execute the thread
  - *ResumeThread* function

SGXCrypter Security Assessment

- Static Analysis
  - Strong encryption scheme protects original PE
  - Decryption key cannot be extracted through disassembly
  - Enclave code accessible but does not provide enough information
- Dynamic Analysis
  - Enclave inaccessible during execution (SGX guarantees)
  - Debug disabled for enclaves build for release
  - Diffie-Hellman key exchange based communication to receive decryption key

Experimental Results

- Antivirus (AV) evasion
  - Standard metric for Crypter evaluation
  - High evasion → stronger protection

Crypter	Benchmark		
	AES	DbgView	Nivdort
SGXCrypter	100%	100%	100%
Aegis	71%	71%	57%
CrypterBinder	74%	94%	69%
Hyperion	40%	40%	43%
Yoda's Protector	91%	94%	77%

Experimental Results

- Initial executable load overhead
  - One time overhead – linear to binary size
  - Results present encryption of the whole binary
    - Partial protection possible

Benchmark	SGXCrypter	Original
HelloWorld (7KB)	265ms	9ms
Hamming (7KB)	271ms	11ms
AES-128 (17KB)	297ms	12ms
SFX RAR (62KB)	374ms	33ms
SFX RAR (474KB)	502ms	53ms
GNU gcc (797KB)	861ms	67ms
SFX RAR (4.8MB)	3686ms	118ms

Conclusions

- SGXCrypter
  - Strong protection for Portable Executables against IP theft
    - Encrypt the PE
    - Protect the key from static & dynamic analysis
  - Solid performance
    - Flawless AV evasion
- Future Work
  - Build SGXCrypter for a Linux platform
  - Support execution within the enclave

Thank you

- Code will be available at: https://github.com/momalab/SGXCrypter
- Questions?