SGX Secure Enclaves in Practice
Security and Crypto Review

JP Aumasson, Luis Merino
This talk

● First review of SGX based on real hardware and SDK
● Revealing undocumented parts of SGX
● Tool and application releases
Props
Victor Costan (MIT)
Shay Gueron (Intel)
Simon Johnson (Intel)
Samuel Neves (Uni Coimbra)
Joanna Rutkowska (Invisible Things Lab)
Arrigo Triulzi
Dan Zimmerman (Intel)
Kudelski Security for supporting this research
What's SGX, how secure is it?

<table>
<thead>
<tr>
<th>Supervisor Instruction</th>
<th>Description</th>
<th>User Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENCLS[EADD]</td>
<td>Add a page</td>
<td>ENCLU[EENTER]</td>
<td>Enter an Enclave</td>
</tr>
<tr>
<td>ENCLS[EBLOCK]</td>
<td>Block an EPC page</td>
<td>ENCLU[EEXIT]</td>
<td>Exit an Enclave</td>
</tr>
<tr>
<td>ENCLS[ECREATE]</td>
<td>Create an enclave</td>
<td>ENCLU[EGETKEY]</td>
<td>Create a cryptographic key</td>
</tr>
<tr>
<td>ENCLS[EDBGRD]</td>
<td>Read data by debugger</td>
<td>ENCLU[EREPORT]</td>
<td>Create a cryptographic report</td>
</tr>
<tr>
<td>ENCLS[EDBGWR]</td>
<td>Write data by debugger</td>
<td>ENCLU[ERESUME]</td>
<td>Re-enter an Enclave</td>
</tr>
<tr>
<td>ENCLS[EEXTEND]</td>
<td>Extend EPC page measurement</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENCLS[EINIT]</td>
<td>Initialize an enclave</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENCLS[ELDB]</td>
<td>Load an EPC page as blocked</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENCLS[ELDU]</td>
<td>Load an EPC page as unblocked</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENCLS[EPA]</td>
<td>Add version array</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

New instruction set in **Skylake** Intel CPUs since autumn 2015
SGX as a reverse sandbox

Protects **enclaves of code/data** from

- **Operating System**, or hypervisor
- BIOS, firmware, drivers
- System Management Mode (**SMM**)
  - aka ring -2
  - Software “between BIOS and OS”
- Intel Management Engine (**ME**)
  - aka ring -3
  - “CPU in the CPU”
- By extension, **any remote attack**
There is no cloud
it's just someone else's computer
Example: make reverse engineer impossible

1. Enclave generates a key pair
   a. Seals the private key
   b. Shares the public key with the authenticated client

2. Client sends code encrypted with the enclave's public key

3. CPU decrypts the code and executes it
A trusted computing enabler
Or secure computing on someone else's computer
Not a new idea, key concepts from the 1980s

Hardware-enforced security requires:

- Trusted computing base
- Hardware secrets
- Remote attestation
- Sealed storage
- Memory encryption
Trusted computing base

- **CPU’s package boundary**: IC, ucode, registers, cache
- Software components used for attestation (mainly QE)

You have to trust Intel anyway if you use an Intel CPU :-)

Caveats: need a **trusted dev environment** for creating enclaves

**Reflections on Trusting Trust**

To what extent should one trust a statement that a program is free of Trojan horses? Perhaps it is more important to trust the people who wrote the software.
Hardware secrets
Two 128-bit keys fused at production:

- Root **provisioning key**
- Root **seal key** (not known to Intel)

Derived keys depend on the seal key, so Intel can't know them

*Figure 5: SGX Key Hierarchy*
Security limitations

Cache-timing attacks on enclave code

- Programs should be constant-time, cache-safe (SGX won't transform insecure software into secure software)

Physical attacks on the CPU

- Need physical access, may destroy the chip (such as laser fault injection attacks)

Microcode malicious patching

- Needs special knowledge, persistence difficult
Vulnerability research

**SGX is complex**, unlikely to be bug-free

Most SGX is black-box, with a large part implemented in ucode :-/

- **Complex instructions** like `EINIT`, `EGETKEY`: partially documented, but all ucode; black-box testing/fuzzing?
- **Platform software**: Drivers, critical Intel enclaves, etc.
- **SDK**: Debug-mode libs available for SGX’ libc and crypto

Vulnerabilities can be disclosed at [https://security-center.intel.com/](https://security-center.intel.com/)
CPU bugs exist
From Intel’s 6th Generation family specs update

<table>
<thead>
<tr>
<th>No Fix</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ENCLU[EGETKEY] Ignores KEYREQUEST.MISCMASK</td>
</tr>
<tr>
<td></td>
<td>ENCLU[EREPORT] May Cause a #GP When TARGETINFO.MISCSELECT is Non-Zero</td>
</tr>
<tr>
<td></td>
<td>ENCLS[ECREATE] Causes #GP if Enclave Base Address is Not Canonical</td>
</tr>
<tr>
<td></td>
<td>ENCLS[EINIT] Instruction May Unexpectedly #GP</td>
</tr>
<tr>
<td></td>
<td>The SMSW Instruction May Execute Within an Enclave</td>
</tr>
<tr>
<td></td>
<td>Intel® SGX Enclave Accesses to the APIC-Access Page May Cause APIC-Access VM Exits</td>
</tr>
</tbody>
</table>
Bugs can be in dependencies

Example: SGX’ aesm_service.exe uses OpenSSL

"ASN.1 part of OpenSSL 1.0.1m 19 Mar 2015"

CVE-2016-2108 doesn't seem exploitable

The ASN.1 implementation in OpenSSL before 1.0.1o and 1.0.2 before 1.0.2c allows remote attackers to execute arbitrary code or cause a denial of service (buffer underflow and memory corruption) via an ANY field in crafted serialized data, aka the "negative zero" issue.

Publish Date : 2016-05-04 Last Update Date : 2016-06-10
Can SGX be patched?
Yes for most of it, including trusted enclaves & microcode

1.4 Upgrading the TCB
The architecture of SGX was designed so that if certain classes of vulnerabilities are discovered in SGX, they can be removed by an upgrade to the platform. This is referred to as TCB Recovery. It is desirable in those cases that the new TCB be reflected in the platform’s attestations. The

The memory encryption crypto **cannot** be patched (hardware)

TCB version verified during remote attestation
Developing for SGX
Setup

- Purchase an SGX-enabled Skylake CPU (6th gen)
- Enable SGX in the BIOS (if supported)
- Install MS Visual Studio Professional 2012 (30-days trial)
- Install Intel Platform Software and SDK
At last! Linux SDK and PSW

Released on June 25th

SDK and PSW source code

https://01.org/intel-softwareguard-eXtensions
https://github.com/01org/linux-sgx
https://github.com/01org/linux-sgx-driver
HTTPS download of the SDK? Yes but no

Same issue with the PSW download
Expired SDK installer cert

Observed on **April 7th, 2016**, reported to Intel (now fixed)
Platform Software (PSW)

Required to run SGX enclaves, contains:

- Drivers, service, DLLs
- Intel privileged enclaves:
  - le.signed.dll: Launch policy enforcement
  - qe.signed.dll: EPID, remote attestation
  - pse.signed.dll: Provisioning service
  - pce.signed.dll: Platform certificate (?) [new in 1.6]

All PEs have ASLR and DEP enabled

PEs signed, FORCE_INTEGRITY not set
Linux prebuilt binaries

https://01.org/sites/default/files/downloads/intelr-software-guard-extensions-linux-os/sgx
prebuilt-1.5.80.27216.tar
sha256sum on June 27th:
4d2be629a96ab9fca40b70c668a16448caecd9e44bed47aef02f1c99821d127b

Prebuilt enclaves (LE, QE, PVE) with symbols
Linux SDK & PSW source code

- ~170 kLoCs of C(++)
- BSD License (+ limited patent license)
- Trusted libc derived from OpenBSD's (and some NetBSD)
- Deps: dlmalloc, Protocol Buffers, STLPort, OpenSSL, etc.

Builds shared libraries and CLI tools
SDK
Required to **develop** SGX enclaves and applications.

- **SGX libs**: Intel-custom libc and crypto lib, sgx-specific libs, each coming in two flavours, debug and release
- **Tools**:
  - `sgx_edger8r` to generate glue (C code & headers)
  - `sgx_sign` to sign enclaves with our dev key
- **Example code** incomplete and not fully reliable
Debugging and disassembly

Visual Studio debugger for **debug-mode** enclaves. GDB in Linux.

**Release-mode** enclaves undebuggable, like one big instruction

SGX decoded by the popular disassemblers (IDA, r2, etc.)
SGX license program

- Can't use the real thing easily
  - **Debug** mode is not secure
  - **Release** mode is secure, but needs an Intel approved developer key (human interaction and NDA required)

**Major change ahead:**
Intel will enable custom Launch Enclaves in future CPUs, as recent documents indicate, to enable custom launch policies.
Developing an enclave application

An SGX-based application is partitioned in two parts:

- **Untrusted**: Starts the enclave, interacts with external parties
- **Trusted**: Executes protected code using sealed secrets
  - Its memory can’t be accessed by any other component
- They can call each other ("ecalls" and "ocalls")

**Challenges:**

- Split code in trusted and untrusted domains
- Validate **untrusted inputs** (the OS can’t be trusted)
Development constraints

- Syscalls & some CPU instructions are not allowed
- Enclaves are **statically linked** (all code must be measured)
- Code runs in **ring3 only**, no kernel mode
- Memory limit set during enclave signing (changing in SGX2)
Sealed storage

- Encrypting secrets inside the enclave,
- To store them out!

How does it work?

- AES-GCM
- IV sourced from RDRAND
- Key derived from HW secrets and enclave or signer identity
- Different keys for debug- and production-mode
- Possible replay protection and time-based policies
Remote attestation

We want to:

● Remotely verify the enclave integrity
● Establish a secure channel client–enclave

How does it work?

● Handshake with ECDH key agreement + fresh quote
● Verify enclave hash, signature, version, !debug
● Verify quote with Intel Attestation Service (registration needed)
● If trusted, provision secrets :)
Crypto in SGX

1. Create enclave and call enclave_init_ra(b_pse)
2. sgx_create_pse_session
3. sgx_ra_init(g_sp_pub_key, b_pse)
4. sgx_close_pse_session
5. context
6. sgx_ra_get_msg1(context, enclave_id, sgx_ra_get_ga)
7. msg1
8. ra_network_send_receive
9. msg2
10. sgx_ra_proc_msg2(context, enclave_id, sgx_ra_proc_msg2_trusted_t, sgx_ra_get_msg3_trusted_t, msg2)
11. msg3
12. ra_network_send_receive
13. result

Image: Intel
SGX crypto zoo

- **RSA-3072 PKCS 1.5 SHA-256**, for enclaves signatures
- **ECDSA** over p256, SHA-256, for launch enclave policy checks
- **ECDH and ECDSA** (p256, SHA-256), for remote key exchange
- **AES-128** in **CTR, GCM, CMAC** at various places: GCM for sealing, CMAC for key derivation, etc.

→ **128-bit** security, except for RSA-3072 (≈ 112-bit)

**Memory encryption** engine (hw), cf. Gueron’s RWC’16 talk:

- New universal hash-based **MAC**, provably secure
- **AES-CTR** with custom counter block
Built-in SGX crypto lib: “somewhat limited”

Libraries `sgx_tcrypto.lib` and `sgx_tcrypto_opt.lib`

Cryptography Library

The Intel® Software Guard Extensions Evaluation SDK includes a trusted cryptography library named `sgx_tcrypto`. It includes the cryptographic functions used by other trusted libraries included in the SDK, such as the `sgx_tservice` library. Thus, the functionality provided by this library might be somewhat limited. If you need additional cryptographic functionality, you would have to develop your own trusted cryptographic library.

AES (GCM, CTR), AES-CMAC, SHA-256, ECDH, ECDSA

- Secure, standard algorithms, 128-bit security
- CTR supports weak parameters (e.g. 1-bit counter)
What crypto lib?
Code from Intel’s proprietary IPP 8.2 “gold” (2014)
Only binaries available (debug-mode libs include symbols)

AES_GCMEncrypt

Encrypts a data buffer in the GCM mode.

Syntax

IppStatus ippsAES_GCMEncrypt(const Ipp8u* pSrc, Ipp8u* pDst, int len, IppsAES_GCMState* pState);

Include Files

ippcp.h

Domain Dependencies

Headers: ippcore.h
Libraries: ippcore.lib
SGX crypto lib on Linux

Similar IPP code too, but comes with source code

- In `sdk/tlibcrypto, external/crypto_px, etc.
- SGX public keys in `psw/ae/data/constants/linux`

Clean and safe code compared to most FOSS crypto libs

```c
SGX_EC_COMPOSITE_BASE,  /* field based on composite */
SGX_EC_COMPLICATED_BASE, /* number of non-zero terms in the polynomial (> PRIME_ARR_MAX) */
SGX_EC_IS_ZERO_DISCRIMINANT, /* zero discriminant */
SGX_EC_COMPOSITE_ORDER,  /* composite order of base point */
SGX_EC_INVALID_ORDER,    /* invalid base point order */
SGX_EC_IS_WEAK_MOV,      /* weak Menezes-Okamoto-Vanstone reduction attack */
SGX_EC_IS_WEAK_SSA,      /* weak Semaev-Smart,Satoh-Araki reduction attack */
SGX_EC_IS_SUPER_SINGULAR, /* supersingular curve */
SGX_EC_INVALID_PRIVATE_KEY,   /* !(0 < Private < order) */
SGX_EC_INVALID_PUBLIC_KEY,   /* (order*PublicKey != Infinity) */
SGX_EC_INVALID_KEY_PAIR,     /* (Private*BasePoint != PublicKey) */
```
SDK's AES implementation (Windows)

“To protect against software-based side channel attacks, the crypto implementation of AES-GCM utilizes AES-NI, which is immune to software-based side channel attacks.”

(SDK documentation)

- AES-NI used for the rounds (**AESENC, AESDEC**)
- Not for the key schedule (no **AESKEYGENASSIST**)
- **Table-based implementation** instead with defenses against cache-timing attacks
SDK's AES implementation (Linux)

No AES-NI, textbook implementation instead (slower)
S-box = 256-byte table with basic cache-timing mitigation

```
INLINE Ipp8u getSboxValue(Ipp32u x)
{
    Ipp32u t[offsetof(RijEncSbox)/CACHE_LINE_SIZE];
    const Ipp8u* SboxEntry = RijEncSbox + x%CACHE_LINE_SIZE;
    Ipp32u i;
    for(i=0; i<offsetof(RijEncSbox)/CACHE_LINE_SIZE; i+=4, SboxEntry += 4*CACHE_LINE_SIZE) {
        t[i] = SboxEntry[CACHE_LINE_SIZE*0];
        t[i+1] = SboxEntry[CACHE_LINE_SIZE*1];
        t[i+2] = SboxEntry[CACHE_LINE_SIZE*2];
        t[i+3] = SboxEntry[CACHE_LINE_SIZE*3];
    }
    return (Ipp8u)t[x/CACHE_LINE_SIZE];
}
```

However, AES in prebuilt enclaves to use AES-NI
No weak randomness in SGX’ libc?

SGX' libc does not support the weak `rand()` and `srand()`

Only **RDRAND-based PRNG (not RDSEED):**

```c
sgx_status_t sgx_read_rand(
    unsigned char *rand,
    size_t length_in_bytes
);
```

“there are some circumstances when the RDRAND instruction may fail. When this happens, the recommendation is to **try again up to ten times** (…)” (Enclave’s writer guide)
sgx_read_rand implements the 10x retry

```c
#define _RDRAND_RETRY_TIMES 10

extern "C" uint32_t do_rdrand(uint32_t *rand);

/*
 *  ------------------------------
 *  * extern "C" uint32_t do_rdrand(uint32_t *rand);
 *  * return value:
 *  *    non-zero: rdrand succeeded
 *  *    zero: rdrand failed
 *  *  ------------------------------
 */

DECLARE_LOCAL_FUNC do_rdrand

mov $RDRAND_RETRY_TIMES, %ecx

.Lrdrand_retry:
    .byte 0x0F, 0xC7, 0xF0 /* rdrand %eax */
    jc .Lrdrand_return
    dec %ecx
    jnz .Lrdrand_retry
    xor %rax, %rax
    ret

.Lrdrand_return:

#ifdef LINUX32
    mov SE_WORDSIZE(%esp), %ecx
#else
    mov %rdi, %rcx
#endif
    movl %eax, (%ecx)
    mov $1, %rax
    ret

public do_rdrand

proc near do_rdrand
    mov edx, 0Ah
    @rdrand_retry:
        rdrand eax
        jb short @rdrand_return
        dec edx
        jnz short @rdrand_retry
        xor rax, rax
        ret
    ;
    @rdrand_return:
        mov [rcx], eax
        mov rax, 1
        ret
    do_rdrand endp
```
Crypto DoS warning

RDRAND / RDSEED are the only non-SGX SGX-enabled instructions that an hypervisor can force to cause a VM exit.

Can be used to force the use of weaker randomness.

3.6.2 RDRAND and RDSEED Instructions

These instructions may cause a VM exit if the “RDRAND exiting” VM-execution control is 1. Unlike other instructions that can cause VM exits, these instructions are legal inside an enclave. As noted in Section 6.5.5, any VM exit originating on an instruction boundary inside an enclave sets bit 27 of the exit-reason field of the VMCS. If a VMM receives a VM exit due to an attempt to execute either of these instructions determines (by that bit) that the execution was inside an enclave, it can do either of two things. It can clear the “RDRAND exiting” VM-execution control and execute VMRESUME; this will result in the enclave executing RDRAND or RDSEED again, and this time a VM exit will not occur. Alternatively, the VMM might choose to discontinue execution of this virtual machine.

NOTE

It is expected that VMMs that virtualize Intel SGX will not set “RDRAND exiting” to 1.
Beware weak crypto

Toy crypto lib in /sdk/sample_libcrypto/

/*
 * This sample cryptography library was intended to be used in a limited
 * manner. Its cryptographic strength is very weak. It should not be
 * used by any production code. Its scope is limited to assist in the
 * development of the remote attestation sample application.
 */
The quoting enclave (QE)

Critical for remote attestation:

1. Verifies an enclave's measurement (create by the \texttt{EREPORT} instruction)
2. Signs it as EPID group member
3. Create a QUOTE: an \textit{attestation} verifiable by third parties

Uses an undocumented \texttt{custom crypto} scheme...
Quoting enclave's crypto

Random 16-byte key and 12-byte IV
Details in https://github.com/kudelskisecurity/sgxfun
Quoting enclave's crypto

- Hybrid encryption, IND-CCA (OAEP) + IND-CPA (GCM)
- SHA-256(K) leaks info on K, enables time-memory tradeoffs
- No forward secrecy (compromised RSA key reveals prev. Ks)
- RSA-2048 ~ 90-bit security level
Enhanced Privacy ID anonymous group signatures

Signatures verified to belong to the group, hiding the member that signed

Issuer, holds the "master key", can grant access to the group

**Group** = CPUs of same type, same SGX version

Members sign an enclave's measurement anonymously

Verifier ensures that an enclave does run on a trusted SGX platform
EPID implementation
Not in microcode, too complex
Not in SGX libs, but in theQE and PVE binaries

Undocumented implementation details:

- Scheme from https://eprint.iacr.org/2009/095
- Barretto-Naehrig curve, optimal Ate pairing

Pubkey and parameters provided by Intel Attestation Service (IAS)
Our projects
SGX and crypto applications

SGX lets you use the CPU as a **hardware key store** to easily realize complex functionalities such as:

- Fully homomorphic encryption
- Multiparty computation
- Secure remote storage
- Proxy reencryption
- Secure delegation
- Encrypted search
Reencryptions
Transform ciphertext $\text{Enc}(K_1, M)$ into ciphertext $\text{Enc}(K_2, M)$:

- Without exposing plaintext nor keys to the OS
- **Symmetric keys** only, no private key escrow!
- **Sealed** keys and policies:
  - Which keys can I encrypt to/from?
  - Which clients can use my key? When does it expire?

Our PoC: multi-client, single-server

https://github.com/kudelskisecurity/sgx-reencrypt
Reencryptions security

**Goal**: leak no info on plaintext, secret keys, key IDs, policies

**Limitations:**

- OS may tamper with sealed blobs, but the enclave will notice it
- No **trusted clock**: OS can bypass the key expiration, cannot implement reliable time-based policies
- Sealed keys are fetched on every reencrypt request: OS can see which pairs are used together
request = (ClientID, nonce, kID0, kID1, C0)

box = \texttt{crypto\_box}(pk\text{-}enc, request)

\texttt{crypto\_open}(box)

\begin{align*}
\text{If policy check fails: } & \text{response} = \text{nonce } \| \text{err0 } \| \text{C0} \\
\text{If } (P = \text{Dec(key0, C0)}) \text{ fails: } & \text{response} = \text{nonce } \| \text{err1 } \| \text{C0} \\
\text{response} & = \text{nonce } \| \text{OK } \| \text{Enc(key1, P)}
\end{align*}

box = \texttt{crypto\_box}(pk\text{-}cli, response)

\texttt{crypto\_open}(box)

\textit{(C0 in error responses to make them indistinguishable from legit responses)}
Reencryptions implementation

- Curve25519 key agreement, Salsa20-Poly1305 auth'd enc.
  - SGX'd TweetNaCl: compact minimal standalone crypto lib
  - Mutual authentication client - enclave

- No remote attestation implemented:
  - `generate_keypair` in a trusted environment

- Interfaces (NaCl boxed request + response):
  - `register_key`: seals a new key + policy, returns key ID
  - `reencrypt`: given a ciphertext and 2 key IDs, produces a new ciphertext if the policy is valid, errs otherwise
Command-line tools

At [https://github.com/kudelskisecurity/sgxfun](https://github.com/kudelskisecurity/sgxfun)

- `parse_enclave.py` extracts metadata from an enclave: signer and security attributes, build mode, entry points, etc.
- `parse_quote.py` extracts information from a quote: EPID group ID, key hash, ISV version, encrypted signature, etc.
- `parse_sealed.py` extracts information from sealed blobs: key policy, payload size, additional authenticated data, etc.

DEMO!
Conclusion
Black Hat sound bytes

- Intel® SGX allows you to run trusted code on a remote untrusted OS/hypervisor, which has many cool applications.

- Many complex software and crypto components need to be secure so that SGX lives up to its promises.

- We are not disclosing major security issues, but presenting undocumented aspects of the SGX architecture.
Open questions

● How bad/exploitable will be bugs in SGX?
● Will cloud providers offer SGX-enabled services?
● Will board manufacturers enable custom LEs in their BIOS?
● Will open-source firmware (such as coreboot) support SGX?
● Will SGX3 use post-quantum crypto? :-)

---

The document presents a list of open questions related to SGX, the Intel Security Guard Extensions. The questions cover various aspects, including the severity of bugs, adoption by cloud providers, support for custom LEs in BIOS, and compatibility with open-source firmware. The last question humorously wonders about the use of post-quantum crypto in SGX3.
Main references

- Intel's official SGX-related documentation (800+ pages)
  - Intel Software Guard Extensions Programming Reference, first-stop for SGX
  - SDK User Guide, SGX SDK API reference
  - Intel’s Enclave Writer’s Guide
- Baumann et al, Shielding Applications from an Untrusted Cloud with Haven, USENIX 2014
- Beekman, https://github.com/jethrogb/sgx-utils
- Costan & Devadas, Intel SGX Explained, eprint 2016/086
- Gueron, Intel SGX Memory Encryption Engine, Real-World Crypto 2016
- Gueron, A Memory Encryption Engine Suitable for General Purpose Processors, eprint 2016/204
- Hoekstra et al, Using Innovative Instructions to Create Trustworthy Software Solutions, HASP 2013
- Ionescu, Intel SGX Enclave Support in Windows 10 Fall Update (Threshold 2)
- NCC Group, SGX: A Researcher’s Primer
- Rutkowska, Intel x86 considered harmful
- Rutkowska, Thoughts on Intel’s upcoming Software Guard Extensions (parts 1 and 2)
- Shih et al, S-NFV: Securing NFV states by using SGX, SDN-NFVSec 2016
- Shinde et al, Preventing Your Faults from Telling Your Secrets: Defenses against Pigeonhole Attacks, arXiv 1506.04832
- Li et al, MiniBox: A Two-Way Sandbox for x86 Native Code, 2014
Prior works

Some stuff already published, mostly without code:

- MIT’s Costan & Devadas “Intel SGX Explained” (essential!)
- Microsoft’s Haven about SGXing full apps (influenced SGX2)
- Microsoft’s VC3: SGXed Hadoop/MapReduce
- CMU & Google’s 2-way sandbox
- Birr-Pixton’s password storage (first PoC released publicly?)
- Juels et al.'s Town Crier authenticated data feeds
Thank you!

Slides and white paper at
https://github.com/kudelskisecurity/sgxfun

@veorq @iamcorso
https://kudelskisecurity.com