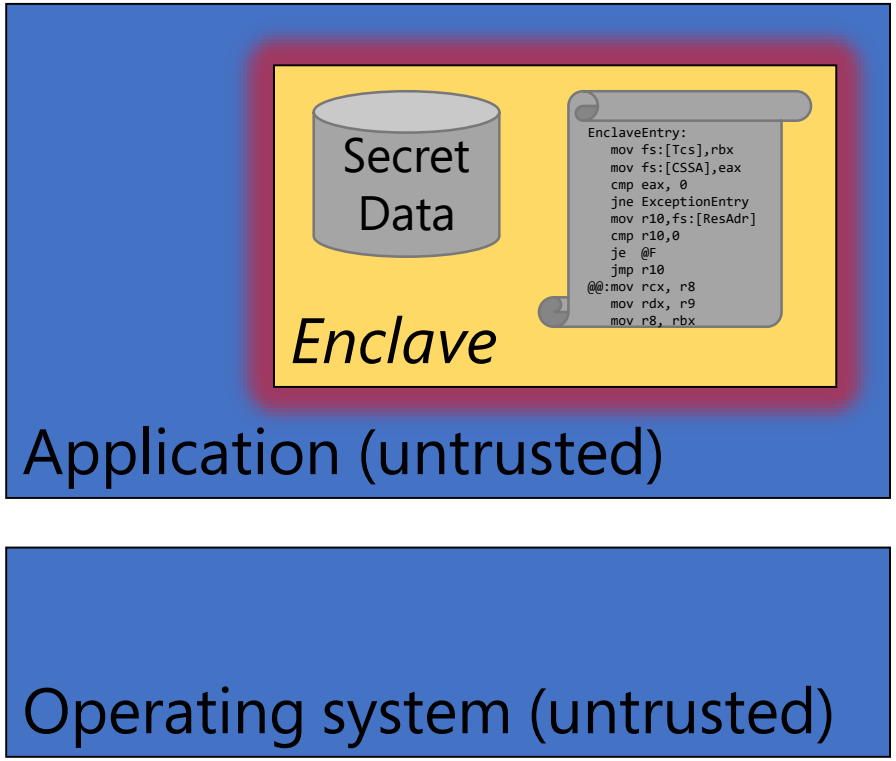
Project Komodo

Andrew Baumann
Microsoft Research

with Andrew Ferraiuolo, Chris Hawblitzel, and Bryan Parno

Intel SGX

- HW-isolated *enclaves*
- Memory encryption
- Remote attestation



SGX is complex

- 18 new instructions in SGX1, 8 more in SGX2
- Changes to exception behaviour, memory access rules
- Interactions with almost everything in the architecture
  - VMX, TXT, SMM, profiling, XSAVE, …

EADD pseudocode

```
IF (DS:RBX is not 32Byte Aligned)
        Then #GP(0); FI;

IF (DS:RCX is not 4KByte Aligned)
        Then #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
        Then #PF(DS:RCX); FI;

TMP_SRCPGE ← DS:RBX.SRCPGE;
TMP_SECS ← DS:RBX.SECS;
TMP_SECINFO ← DS:RBX.SECINFO;
TMP_LINADDR ← DS:RBX.LINADDR;

IF (DS:TMP_SRCPGE is not 4KByte aligned or DS:TMP_SECS is not
4KByte aligned or DS:TMP_SECINFO is not 64Byte aligned or
TMP_LINADDR is not 4KByte aligned)
        Then #GP(0); FI;

IF (DS:TMP_SECS does not resolve within an EPC)
        Then #PF(DS:TMP_SECS); FI;

SCRATCH_SECINFO ← DS:TMP_SECINFO;

(* Check for mis-configured SECINFO flags*)
IF (SCRATCH_SECINFO reserved fields are not zero or !
(SCRATCH_SECINFO.FLAGS.PT is PT_REG or
SCRATCH_SECINFO.FLAGS.PT is PT_TCS) )
        Then #GP(0); FI;

(* Check the EPC page for concurrency *)
IF (EPC page in use)
        Then #GP(0); FI;
IF (EPCM(DS:RCX).VALID != 0)
        Then #PF(DS:RCX); FI;

(* Check the SECS for concurrency *)
IF (SECS is not available for EADD)
        Then #GP(0); FI;

IF (EPCM(DS:TMP_SECS).VALID = 0 or EPCM(DS:TMP_SECS).PT !=
PT_SECS)
        Then #PF(DS:TMP_SECS); FI;

(* Copy 4KBytes from source page to EPC page*)
DS:RCX[32767:0] ← DS:TMP_SRCPGE[32767:0];
CASE (SCRATCH_SECINFO.FLAGS.PT)
{
        PT_TCS:
                IF (DS:RCX.RESERVED != 0) #GP(0); FI;
                IF ( (DS:TMP_SECS.ATTIBUTES.MODE64BIT = 0) and
((DS:TCS.FSLIMIT & 0FFFH != 0FFFH) or (DS:TCS.GSLIMIT & 0FFFH !=
0FFFH) )) #GP(0); FI;
                BREAK;
        PT_REG:
                IF (SCRATCH_SECINFO.FLAGS.W = 1 and
SCRATCH_SECINFO.FLAGS.R = 0) #GP(0); FI;
                BREAK;
ESAC;

(* Check the enclave offset is within the enclave linear address space
*)
IF (TMP_LINADDR < DS:TMP_SECS.BASEADDR or TMP_LINADDR >=
DS:TMP_SECS.BASEADDR + DS:TMP_SECS.SIZE)
        Then #GP(0); FI;

(* Check concurrency of measurement resource*)
IF (Measurement being updated)
        Then #GP(0); FI;

(* Check if the enclave to which the page will be added is already in
Initialized state *)
IF (DS:TMP_SECS already initialized)
        Then #GP(0); FI;

(* For TCS pages, force EPCM.rwx bits to 0 and no debug access *)
IF (SCRATCH_SECINFO.FLAGS.PT = PT_TCS)
        THEN
                SCRATCH_SECINFO.FLAGS.R ← 0;
                SCRATCH_SECINFO.FLAGS.W ← 0;
                SCRATCH_SECINFO.FLAGS.X ← 0;
                (DS:RCX).FLAGS.DBGOPTIN ← 0; // force
TCS.FLAGS.DBGOPTIN off
                DS:RCX.CSSA ← 0;
                DS:RCX.AEP ← 0;
                DS:RCX.STATE ← 0;
        FI;

(* Add enclave offset and security attributes to MRENCLAVE*)
TMP_ENCLAVEOFFSET ← TMP_LINADDR - DS:TMP_SECS.BASEADDR;
TMPUPDATEFIELD[63:0] ← 0000000044444145H; // "EADD"
TMPUPDATEFIELD[127:64] ← TMP_ENCLAVEOFFSET;
TMPUPDATEFIELD[511:128] ← SCRATCH_SECINFO[375:0]; // 48 bytes
DS:TMP_SECS.MRENCLAVE ←
SHA256UPDATE(DS:TMP_SECS.MRENCLAVE, TMPUPDATEFIELD)
INC enclave's MRENCLAVE update counter;

(* Add enclave offset and security attributes to MRENCLAVE*)
EPCM(DS:RCX).R ← SCRATCH_SECINFO.FLAGS.R;
EPCM(DS:RCX).W ← SCRATCH_SECINFO.FLAGS.W;
EPCM(DS:RCX).X ← SCRATCH_SECINFO.FLAGS.X;
EPCM(DS:RCX).PT ← SCRATCH_SECINFO.FLAGS.PT;
EPCM(DS:RCX).ENCLAVEADDRESS ← TMP_LINADDR;

(* associate the EPCPAGE with the SECS by storing the SECS identifier
of DS:TMP_SECS *)
Update EPCM(DS:RCX) SECS identifier to reference DS:TMP_SECS
identifier;

(* Set EPCM entry fields *)
EPCM(DS:RCX).BLOCKED ← 0;
EPCM(DS:RCX).PENDING ← 0;
EPCM(DS:RCX).MODIFIED ← 0;
EPCM(DS:RCX).VALID ← 1;
```

EINIT pseudocode

```
(* make sure SIGSTRUCT and SECS are aligned *)
IF ( (DS:RBX is not 4KByte Aligned) or (DS:RCX is not 4KByte Aligned) )
        Then #GP(0); FI;

(* make sure the EINITTOKEN is aligned *)
IF (DS:RDX is not 512Byte Aligned)
        Then #GP(0); FI;

(* make sure the SECS is inside the EPC *)
IF (DS:RCX does not resolve within an EPC)
        Then #PF(DS:RCX); FI;

TMP_SIG[14463:0] ← DS:RBX[14463:0]; // 1808 bytes
TMP_TOKEN[2423:0] ← DS:RDX[2423:0]; // 304 bytes

(* Verify SIGSTRUCT Header. *)
IF ( (TMP_SIG.HEADER != 06000000E10000000000010000000000h) or
((TMP_SIG.VENDOR != 0) and (TMP_SIG.VENDOR != 00008086h) ) or
(TMP_SIG HEADER2 != 01010000600000006000000001000000h) or
(TMP_SIG.EXPONENT != 00000003h) or (Reserved space is not 0's) )
THEN
        RFLAGS.ZF ← 1;
        RAX ← SGX_INVALID_SIG_STRUCT;
        goto EXIT;
FI;

(* Open "Event Window" Check for Interrupts. Verify signature using embedded public key, q1, and q2. Save upper 352
bytes of the PKCS1.5 encoded message into the TMP_SIG_PADDING*)
IF (interrupt was pending) {
        RFLAG.ZF ← 1;
        RAX ← SGX_UNMASKED_EVENT;
        goto EXIT;

FI

IF (signature failed to verify) {
        RFLAG.ZF ← 1;
        RAX ← SGX_INVALID_SIGNATURE;
        goto EXIT;
FI;

(*Close "Event Window" *)
(* make sure no other Intel SGX instruction is modifying SECS*)
IF (Other instructions modifying SECS)
        Then #GP(0); FI;
IF ( (EPCM(DS:RCX). VALID = 0) or (EPCM(DS:RCX).PT != PT_SECS) )
        Then #PF(DS:RCX); FI;

(* make sure no other instruction is accessing MRENCLAVE or ATTRIBUETS.INIT *)
IF ( (Other instruction modifying MRENCLAVE) or (Other instructions modifying the SECS's Initialized state))
        Then #GP(0); FI;

(* Calculate finalized version of MRENCLAVE *)
(* SHA256 algorithm requires one last update that compresses the length of the hashed message into the output SHA256
digest *)
TMP_ENCLAVE ←¤SHA256FINAL( (DS:RCX).MRENCLAVE, enclave's MRENCLAVE update count *512);

(* Verify MRENCLAVE from SIGSTRUCT *)
IF (TMP_SIG.ENCLAVEHASH != TMP_MRENCLAVE)
        RFLAG.ZF ← 1;
        RAX ← SGX_INVALID_MEASUREMENT;
        goto EXIT;
FI;

TMP_MRSIGNER ← SHA256(TMP_SIG.MODULUS)

(* if INTEL_ONLY ATTRIBUTES are set, SIGSTRUCT must be signed using the Intel Key *)
INTEL_ONLY_MASK ← 0000000000000020H;
IF ( ( (DS:RCX.ATTRIBUTES & INTEL_ONLY_MASK) != 0) and (TMP_MRSIGNER != CSR_INTELPUBKEYHASH) )
        RFLAG.ZF ← 1;
        RAX ← SGX_INVALID_ATTRIBUTE;
        goto EXIT;
FI;

(* Verify SIGSTRUCT.ATTRIBUTE requirements are met *)
IF ( (DS:RCX.ATTRIBUTES & TMP_SIG.ATTRIBUTEMASK) != (TMP_SIG.ATTRIBUTE & TMP_SIG.ATTRIBUTEMASK) )
        RFLAG.ZF ← 1;
        RAX ← SGX_INVALID_ATTRIBUTE;
        goto EXIT;
FI;

( *Verify SIGSTRUCT.MISCSELECT requirements are met *)
IF ( (DS:RCX.MISCSELECT & TMP_SIG.MISCMASK) != (TMP_SIG.MISCSELECT & TMP_SIG.MISCMASK) )
THEN
        RFLAG.ZF ← 1;
        RAX ← SGX_INVALID_ATTRIBUTE;
        goto EXIT
FI;

(* if EINITTOKEN.VALID[0] is 0, verify the enclave is signed by Intel *)
IF (TMP_TOKEN.VALID[0] = 0)
        IF (TMP_MRSIGNER != CSR_INTELPUBKEYHASH)
                RFLAG.ZF ← 1;
                RAX ← SGX_INVALID_EINITTOKEN;
                goto EXIT;
        FI;
        goto COMMIT;
FI;

(* Debug Launch Enclave cannot launch Production Enclaves *)
IF ( (DS:RDX.MASKEDATTRIBUTESLE.DEBUG = 1) and (DS:RCX.ATTRIBUTES.DEBUG = 0) )
        RFLAG.ZF ← 1;
        RAX ← SGX_INVALID_EINITTOKEN;
        goto EXIT;
FI;

(* Check reserve space in EINIT token includes reserved regions and upper bits in valid field *)
IF (TMP_TOKEN reserved space is not clear)
        RFLAG.ZF ← 1;
        RAX ← SGX_INVALID_EINITTOKEN;
        goto EXIT;
FI;

(* EINIT token must be <= CR_CPUSVN *)
IF (TMP_TOKEN.CPUSVN > CR_CPUSVN)
        RFLAG.ZF ← 1;
        RAX ← SGX_INVALID_CPUSVN;
        goto EXIT;
FI;

(* Derive Launch key used to calculate EINITTOKEN.MAC *)
HARDCODED_PKCS1_5_PADDING[15:0] ← 0100H;
HARDCODED_PKCS1_5_PADDING[2655:16] ← SignExtend330Byte(-1); // 330 bytes of 0FFH
HARDCODED_PKCS1_5_PADDING[2815:2656] ← 2004000501020403650148866009060D30313000H;
TMP_KEYDEPENDENCIES.KEYNAME ← LAUNCH_KEY;
TMP_KEYDEPENDENCIES.ISVPRODID ← TMP_TOKEN.ISVPRODIDLE;
TMP_KEYDEPENDENCIES.ISVSVN ← TMP_TOKEN.ISVSVN;
TMP_KEYDEPENDENCIES.OWNEREPOCH ← CSR_SGXOWNEREPOCH;
TMP_KEYDEPENDENCIES.ATTRIBUTES ← TMP_TOKEN.MASKEDATTRIBUTESLE;
TMP_KEYDEPENDENCIES.ATTRIBUTESMASK ← 0;
TMP_KEYDEPENDENCIES.MRENCLAVE ← 0;
TMP_KEYDEPENDENCIES.MRSIGNER ← 0;
TMP_KEYDEPENDENCIES.KEYID ← TMP_TOKEN.KEYID;
TMP_KEYDEPENDENCIES.SEAL_KEY_FUSES ← CR_SEAL_FUSES;
TMP_KEYDEPENDENCIES.CPUSVN ← TMP_TOKEN.CPUSVN;
TMP_KEYDEPENDENCIES.MISCSELECT ← TMP_TOKEN.MASKEDMISCSELECTLE;
TMP_KEYDEPENDENCIES.MISCMASK ← 0;
TMP_KEYDEPENDENCIES.PADDING ← HARDCODED_PKCS1_5_PADDING;

(* Calculate the derived key*)
TMP_EINITTOKENKEY ← derivekey(TMP_KEYDEPENDENCIES);

(* Verify EINITTOKEN was generated using this CPU's Launch key and that it has not been modified since issuing by the
Launch Enclave. Only 192 bytes of EINITOKEN are CMACed *)
IF (TMP_TOKEN.MAC != CMAC(TMP_EINITTOKENKEY, TMP_TOKEN[1535:0] ) )
        RFLAG.ZF ← 1;
        RAX ← SGX_INVALID_EINIT_TOKEN;
        goto EXIT;
FI;

(* Verify EINITTOKEN (RDX) is for this enclave *)
IF (TMP_TOKEN.MRENCLAVE != TMP_MRENCLAVE) or (TMP_TOKEN.MRSIGNER != TMP_MRSIGNER) )
        RFLAG.ZF ← 1;
        RAX ← SGX_INVALID_MEASUREMENT;
        goto EXIT;
FI;

(* Verify ATTRIBUTES in EINITTOKEN are the same as the enclave's *)
IF (TMP_TOKEN.ATTRIBUTES != DS:RCX.ATTRIBUTES)
        RFLAG.ZF ← 1;
        RAX ← SGX_INVALID_EINIT_ATTRIBUTE;
        goto EXIT;
FI;

COMMIT:
(* Commit changes to the SECS; Set ISVPRODID, ISVSVN, MRSIGNER, INIT ATTRIBUTE fields in SECS (RCX) *)
DS:RCX.MRENCLAVE ← TMP_MRENCLAVE;

(* MRSIGNER stores a SHA256 in little endian implemented natively on x86 *)
DS:RCX.MRSIGNER ← TMP_MRSIGNER;
DS:RCX.ISVPRODID ← TMP_SIG.ISVPRODID;
DS:RCX.ISVSVN ← TMP_SIG.ISVSVN;
DS:RCX.PADDING ← TMP_SIG_PADDING;

(* Mark the SECS as initialized *)
Update DS:RCX to initialized;

(* Set RAX and ZF for success*)
RFLAG.ZF ← 0;
RAX ← 0;

EXIT:
RFLAGS.CF,PF,AF,OF,SF ← 0;
```

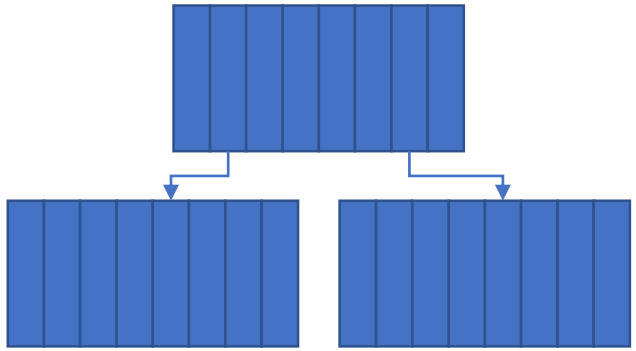
SGX limitations

- Complex instructions
- Implementation bugs
  - CVE-2017-5691
- Vulnerable to old & new side channels
  - Shared caches, TLB, branch predictor, speculation, …
  - “Controlled channel” attacks via paging mechanism
- Assumes particular usage models, limits functionality

Example: memory management

Trusted OS: page table

- Express any page-granular virtual space
- OS updates arbitrarily



SGX: restricted operations

- SGX1: No way to handle faults, allocate memory, or demand load
- SGX2 adds six instructions for dynamic allocation…
- … but 3 years later, we're still waiting
- … and there's still no way to move/remap or share memory

The fundamental problem

- SGX encodes a “mini OS” inside the ISA
- ISA changes are slow to impossible
  1. Convince Intel that your change is worthwhile
  2. Wait for Intel to update the spec
  3. Wait for parts to ship that implement the new spec
  4. Wait for vendors to adopt/deploy new parts…

SGX2

SGX1

- SGX will not be the only instance of this problem
  - e.g. AMD announced memory encryption features

Project Komodo

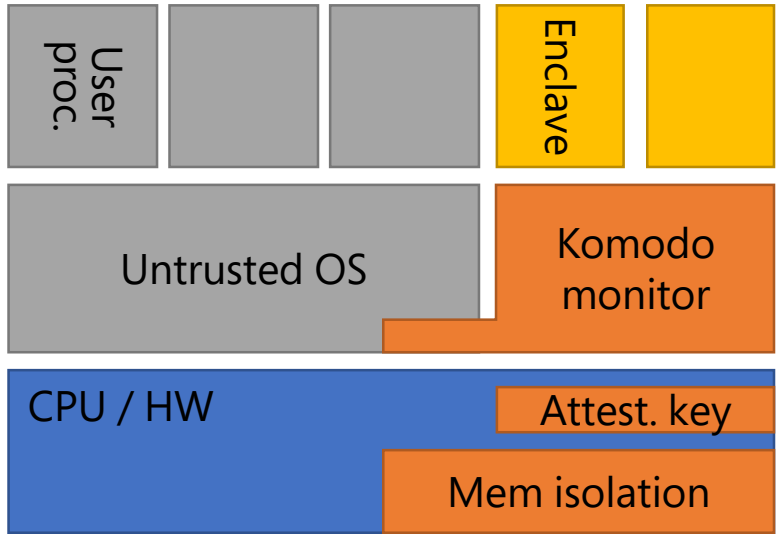
- Simple hardware primitives
- Enclave management in software
- Independent evolution
- Trust through formal verification

What's **right** with SGX?
(properties we seek to preserve)

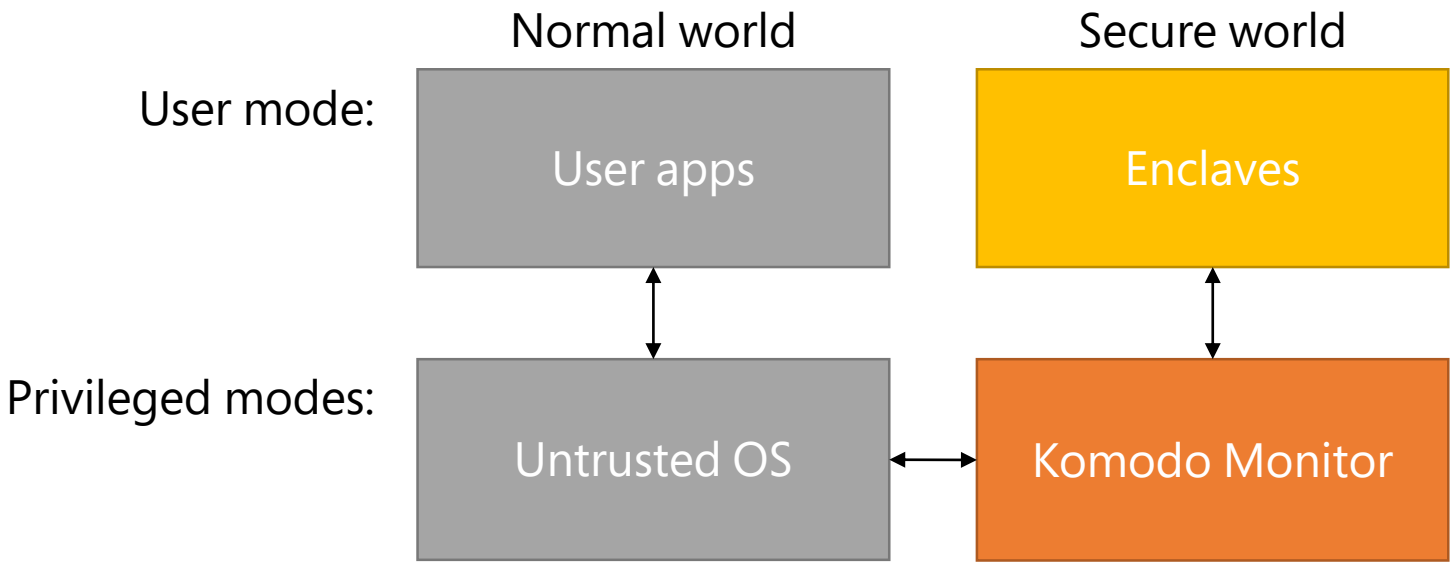
- Avoids changes to processor's fast-path
  - Memory encryption engine
  - Slow-path enclave management instructions
  - Caches, TLBs, core instructions etc. unchanged
- Compatible with existing software
  - (Almost) complete x86 ISA for enclave code
  - OS/hypervisor largely unmodified
- Resource management delegated to untrusted OS
  - SGX acts as a *reference monitor* for OS operations

Komodo architecture

- Hardware provides:
  1. Memory isolation
     - Encryption (Intel, AMD)
     - Isolated RAM (ARM)
  2. Attestation key (TPM, etc.)
  3. Secure randomness
  4. Protection modes for enclave, *monitor*
- Monitor implements security mechanisms
  - Enter/exit enclaves
  - Manage memory mappings
  - Perform attestation



Prototype on ARM TrustZone

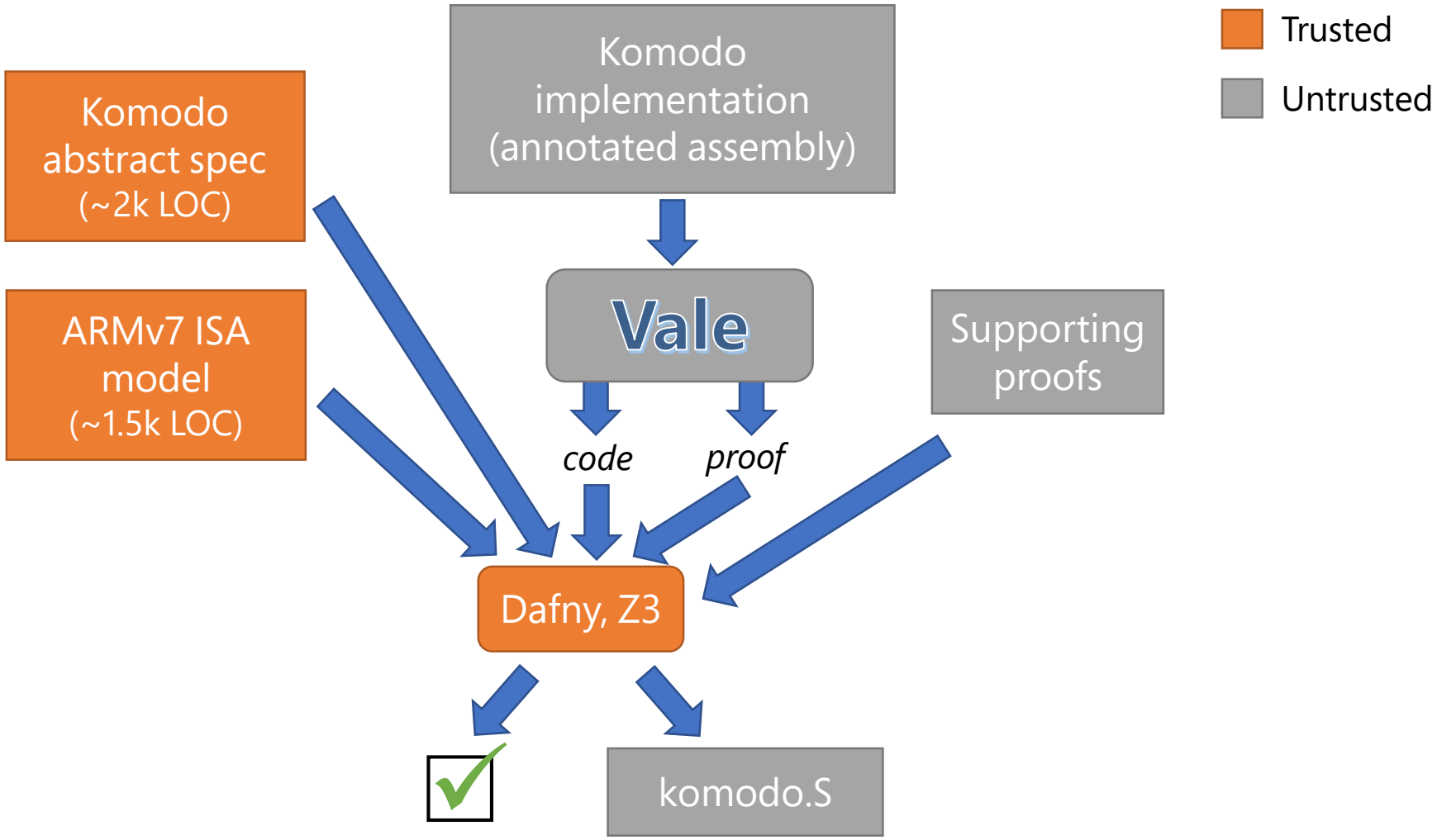


Secure-world physical memory isolated from normal world

Komodo API

- Roughly comparable to SGX
- Untrusted OS API:
  - Create address space, threads, page tables
  - Map secure/insecure pages
  - Finalise (measure) enclave
  - Enter / Resume threads
  - Allocate spare pages
- Enclave API:
  - Create/verify attestations
  - Map/unmap spare pages
  - Exit enclave

Verification overview



Proving security via non-interference

Integrity:

software outside an enclave is noninterfering with its state

Confidentiality:

enclave state is noninterfering with OS-observable state

…modulo declassification of:

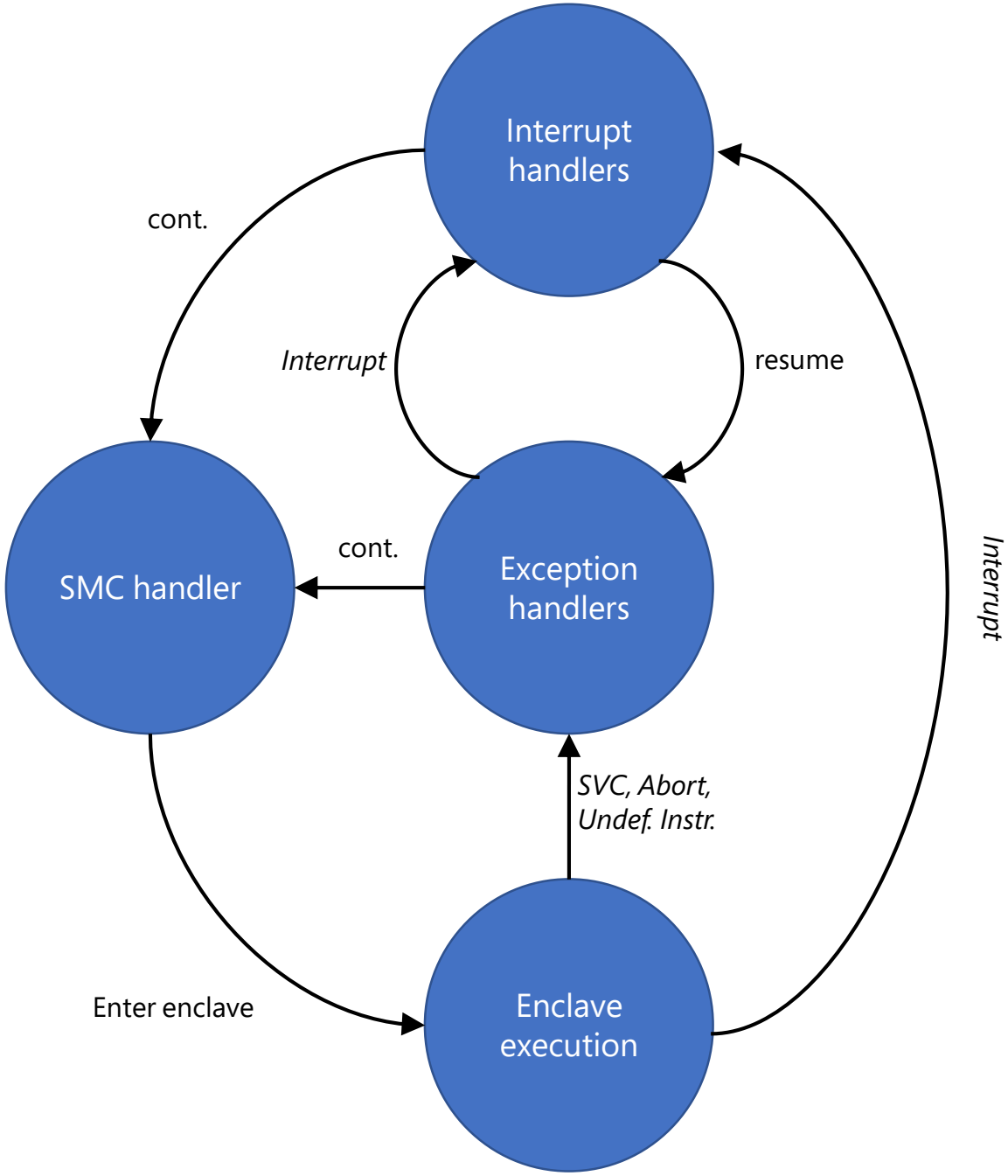
- Communication (parameter/result registers, insecure memory)
- Exception types
- Dynamic allocation side-effects

Verified assembly in Vale

```
procedure ARM_L2PTE(operand mapping:reg, inout operand pte:reg,
                    out operand tmp:reg, ghost absmap:Mapping)
  requires
    SaneState(this);
    @mapping != @pte && @mapping != @tmp && @pte != @tmp;
    @pte != OSP && @tmp != OSP;
    PageAligned(pte); absmap == wordToMapping(mapping);
  ensures
    pte == ARM_L2PTE(old(pte), absmap.perm.w  absmap.perm.x);
    SaneState(this) && SmcProcedureInvariant(old(this), this);
{
  ghost var pa := pte;
  ghost var nxbit := if absmap.perm.x then 0
                     else BitsAsWord(ARM_L2PTE_NX_BIT);
  ghost var robit := if absmap.perm.w then 0
                     else BitsAsWord(ARM_L2PTE_RO_BIT);

  ORR(pte, pte, const(L2PTE_CONST_WORD())); // set const bits

  // compute NX bit
  AND(tmp, mapping, const(KOM_MAPPING_X));
  EOR(tmp, tmp, const(KOM_MAPPING_X));
  lemma_extract_kom_mapping_x(mapping, tmp);
  assert (tmp == 0 || tmp == KOM_MAPPING_X)
      && (tmp == 0 <==> absmap.perm.x);
  LSR(tmp, tmp, 2);
  lemma_shift_nxbit();
  assert (tmp == 0 || tmp == BitsAsWord(ARM_L2PTE_NX_BIT))
      && (tmp == 0 <==> absmap.perm.x);
  assert tmp == nxbit;
  ORR(pte, pte, tmp);

  // compute RO bit
  AND(tmp, mapping, const(KOM_MAPPING_W));
  EOR(tmp, tmp, const(KOM_MAPPING_W));
  lemma_extract_kom_mapping_w(mapping, tmp);
  assert (tmp == 0 || tmp == KOM_MAPPING_W)
      && (tmp == 0 <==> absmap.perm.w);

  LSL(tmp, tmp, 8);
  lemma_shift_robit();
  assert (tmp == 0 || tmp == BitsAsWord(ARM_L2PTE_RO_BIT))
      && (tmp == 0 <==> absmap.perm.w);
  assert tmp == robit;
  ORR(pte, pte, tmp);

  // prove we did the right thing
  lemma_ARM_L2PTE_impl(pa, absmap.perm.w, absmap.perm.x, pte);
}
```

Implementation

- ~21k lines of Dafny & Vale
- Six exception handlers
- Messy HW details:
  - TrustZone modes
  - Register banking
  - Interrupts



Prototype

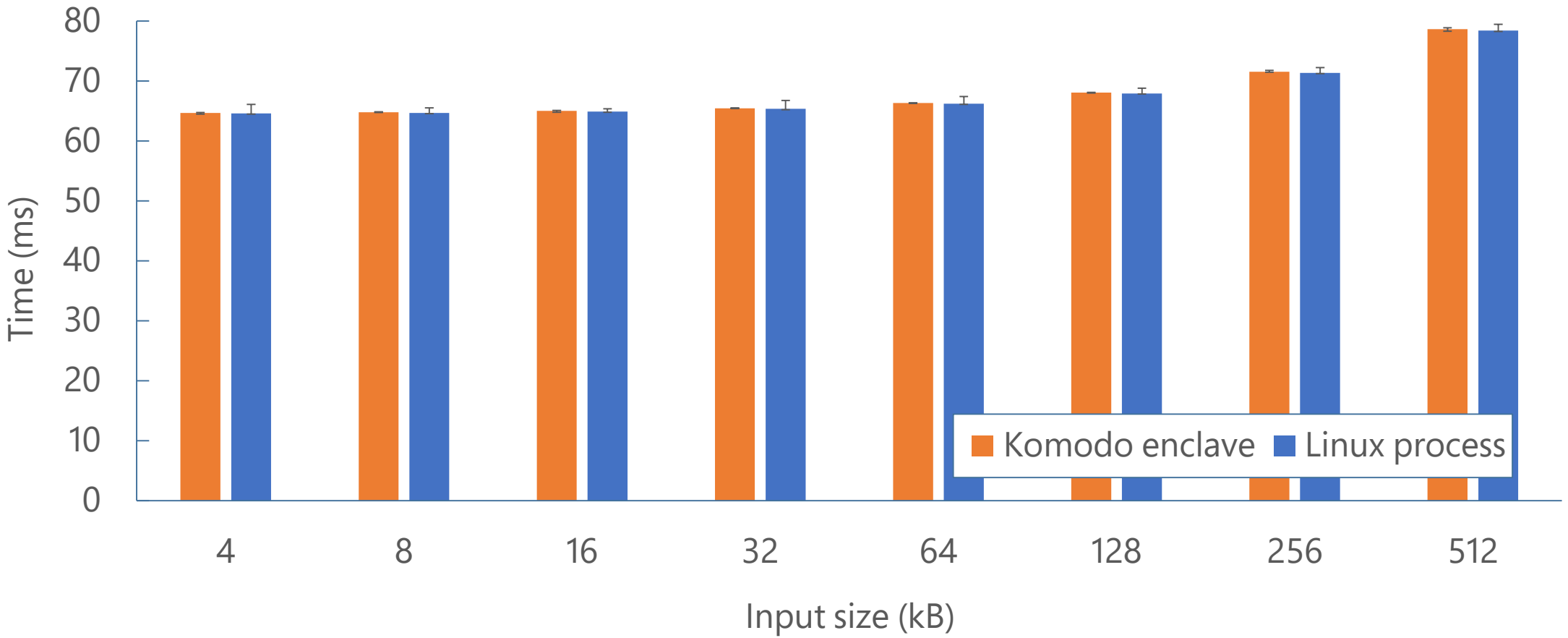
- Runs on Raspberry Pi 2
  - Readily-available TrustZone platform
  - No memory isolation ☹
  - Doesn't affect performance/proofs
- Supporting environment
  - Trusted bootloader
    - loads/configures monitor, boots Linux
    - unverified, but could be…
  - Linux driver for enclave support

Microbenchmark results

Operation	Cycles
Null SMC	123
Enter	496
Resume	625
Enter + Exit	**738**

- cf. SGX: ~7100 cycles enter + exit [Eleos, Eurosys'17]
  - Deeper pipelines, higher clock (2GHz+ vs 900MHz)
  - Why? Maybe: TLB flushes, microcode execution, …?
- Plenty of scope for optimisation

Notary performance



Verification effort

Total effort: approx. 2 person-years

	SLOC
Spec	4,446
Impl	2,710
Proof	18,655

Security	**175**
Correctness	795
ARM model	1,174
Other	2,302

Experiences

- Verification is effective at finding bugs
  - Compared to unverified (but tested) early prototype
- Verification is not a silver bullet
  - Bugs remained in trusted / under-specified code
- Tools are improving, but still have limits
  - Resulting system is correct, but not particularly elegant
  - Timeouts and proof stability are a challenge

Related work

- Enclave-like hardware/software
  - Many proposals, no formal guarantees
  - Sanctum [UsenixSec'16] proposes similar hardware
    - Modified RISC-V, cache partitioning
  - Komodo ≈ verified monitor for Sanctum-like hardware
- Verified kernels (seL4, CertiKOS)
  - Komodo monitor is a little like a microkernel
  - Much simpler: does not handle interrupts or schedule threads
  - Includes attestation, works with unmodified OS

Future work

- Dispatcher upcall interface
  - Self-paging, LibOS support
- Multiprocessor support
  - Biggest limitation & challenge
  - “Big locks” may be sufficient

Conclusion

- SGX implements enclaves entirely as an ISA feature
  - Excessive hardware complexity
  - Limits features, slow to evolve
- Komodo decouples enclave hardware & software
  - Hardware: protection modes, mem isolation/encryption
  - Software: verified monitor implements enclave APIs
- Proven security: enclave confidentiality & integrity
- Demonstrated evolution:

  added dynamic memory in 6 person months

https://github.com/Microsoft/Komodo