Verifying the TPM: How I learned to love the monad

Dr. Perry Alexander

Director, Information and Telecommunication Technology Center

Professor, Electrical Engineering and Computer Science

perry.alexander@ku.edu

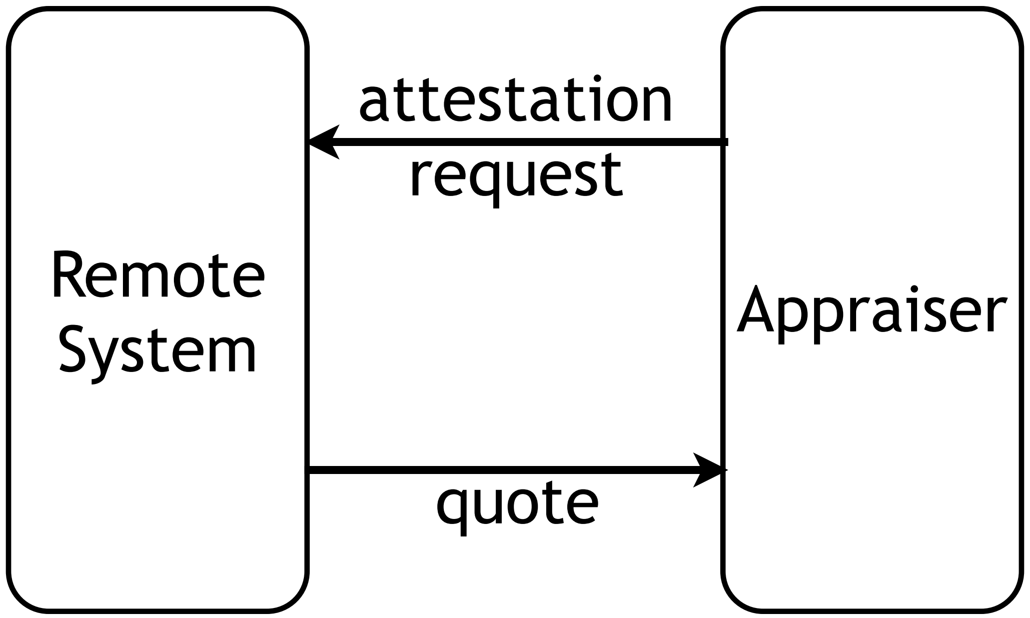
Objectives

- What is a Trusted Processor Module?
  - remote attestation for trustworthiness
  - protecting secrets and locking data to machines
- What are our intellectual tools for verification?
  - bisimulation
  - Galois Connection
  - monadic models
- What do our computational tools look like?
  - modeling languages
  - automated theorem proving
- Would you like to have a “play date” with ITTC?
  - we have neat problems
  - we have fun toys

Remote Attestation

- Appraiser requests a quote
  - specifies information is needed
  - includes a nonce for freshness
- Remote system gathers evidence
  - hashes of executing software
  - hashes of hardware
- Remote system generates a quote
  - evidence describing system
  - the original nonce
  - cryptographic signature
- Appraiser assesses quote
  - correct boot process
  - correct parts
  - evidence integrity

Zero-knowledge proof of trustworthiness



Trusted Processor Module (TPM)

- Platform Configuration Registers (PCRs)
  - extendable hash storage
  - monotonic behavior
- Platform key pairs
  - platform Endorsement Key (EK) uniquely identifying the TPM
  - Storage Root Key (SRK) for wrapping keys
- Cryptographic engine
  - SHA-1 functionality for signature generation
  - DSA cryptographic functionality for sealing and encryption
  - Random number and asymmetric key generators
- Security protocol support
  - quote and signature generation
  - sealing data and wrapping keys

Process Configuration Registers

- PCRs contain measurements
  - SHA-1 hashes of images and data
  - uniquely identifies the state of a system
- Stored in volatile RAM
  - minimum of 12, 120-bit registers
  - monotonic access control
- PCRs are extended rather than set
  - SHA-1 of the PCR concatenated with a new measurement hash value
  - captures the original value, new value, and order
- Records the state of a system and trajectory of states
  - used in attestation to evaluate system state
  - used to seal secrets to system state

new hash measurement

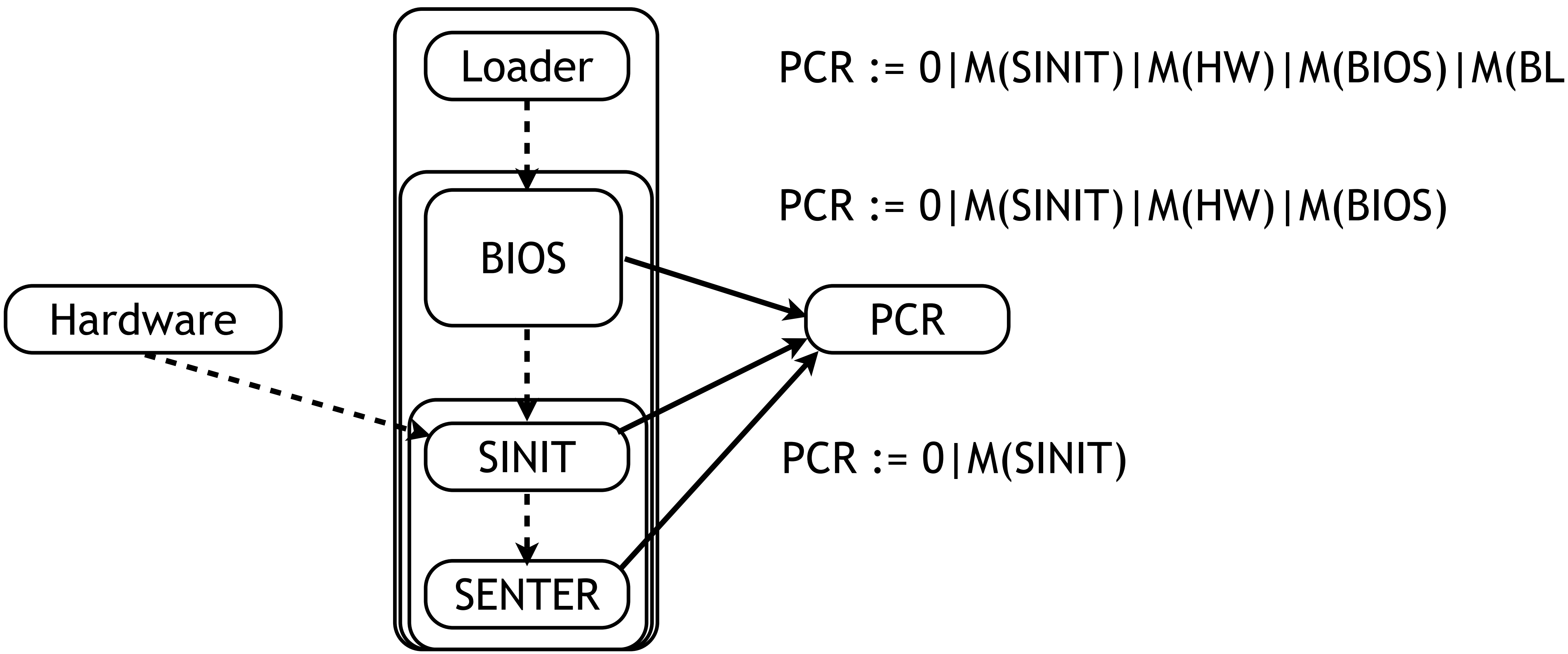
PCR|M = SHA-1(PCR^M)

original PCR hash value

PCR|M ≠ M|PCR

order matters!

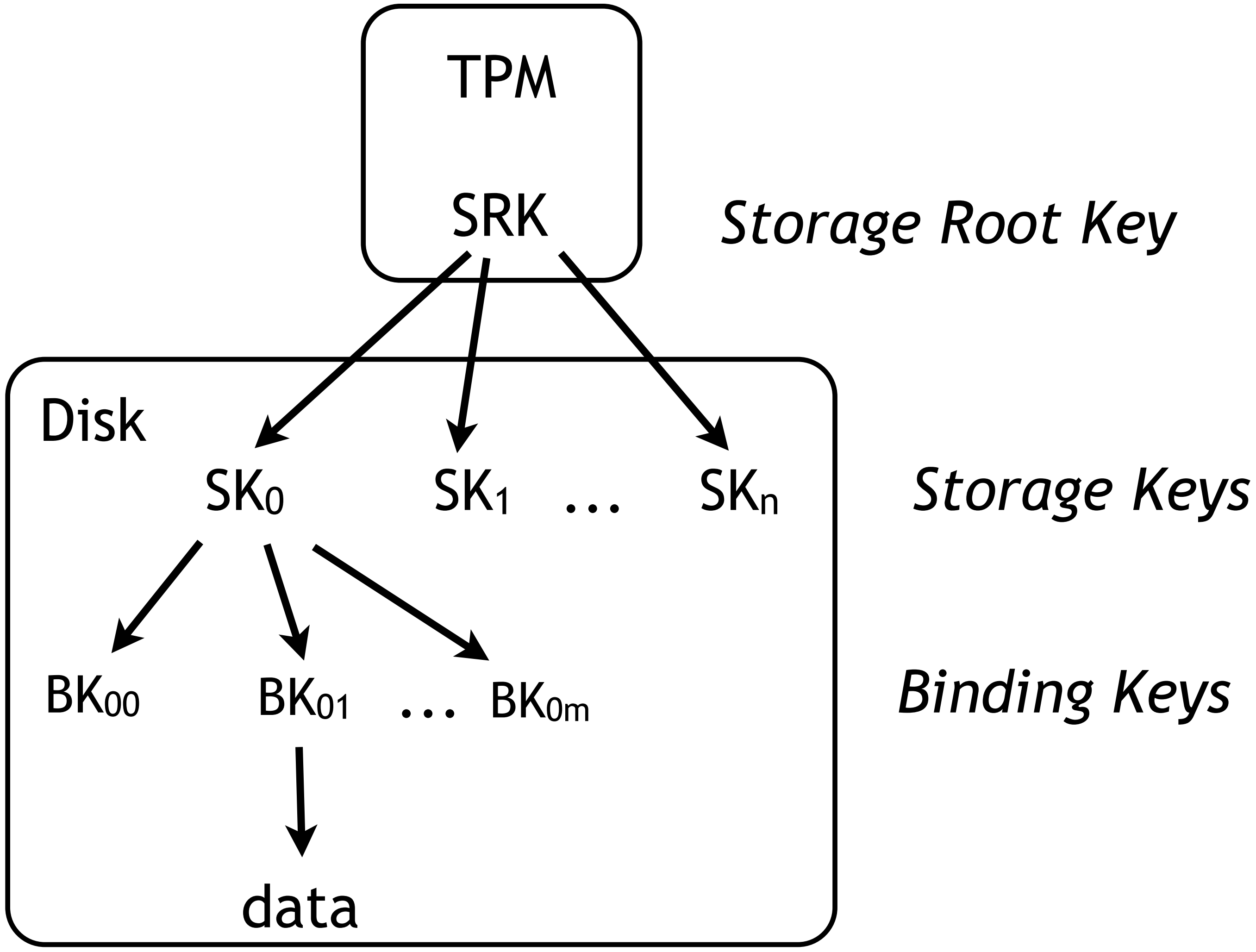
Measured Boot



Core Root of Trust for Measurement
place to stand for the “bottom turtle”

Keys and Data

- Storage Root Key Pair (SRK)
  - generated by TPM when "owned"
  - private key stored in TPM non-volatile RAM
  - public key wraps storage keys on disk
- Storage Keys
  - wrapped key - $(\{SK^{-1}\}_{SRK}, SK)$
  - exclusively used to encrypt keys
- Binding Keys
  - wrapped key - $(\{BK^{-1}\}_{SK}, BK)$
  - encrypts keys and small data
- Wrapped key is sealed
  - TPM PCRs saved when encrypted
  - will not decrypt if TPM PCRs are in a bad state



TPM Functions

- Unique identifier for each TPM
  - factory certificate links TPM to machine
  - nonrepudiation is easier as anonymity is gone
- Quote generation
  - delivering high integrity quotes - $\{|PCR,n|\}_{EK^{-1}}$
  - delivering high integrity evidence - $(\langle E,n\rangle, \{|\#E,PCR,n|\}_{EK^{-1}})$
- Sealing data to state
  - $\{D,PCR\}_K$ will not decrypt unless PCRs = current PCRs
  - data is safe even in the presence of malicious machine
- Linking data to TPMs and machines
  - $(\{K^{-1}\}_{SRK},K)$ - $\{D\}_K$ cannot be decrypted unless SRK is present
  - migrating data securely among TPMs becomes possible

As a core root of trust for measurement and storage that cannot be assessed at run-time the TPM warrants formal verification

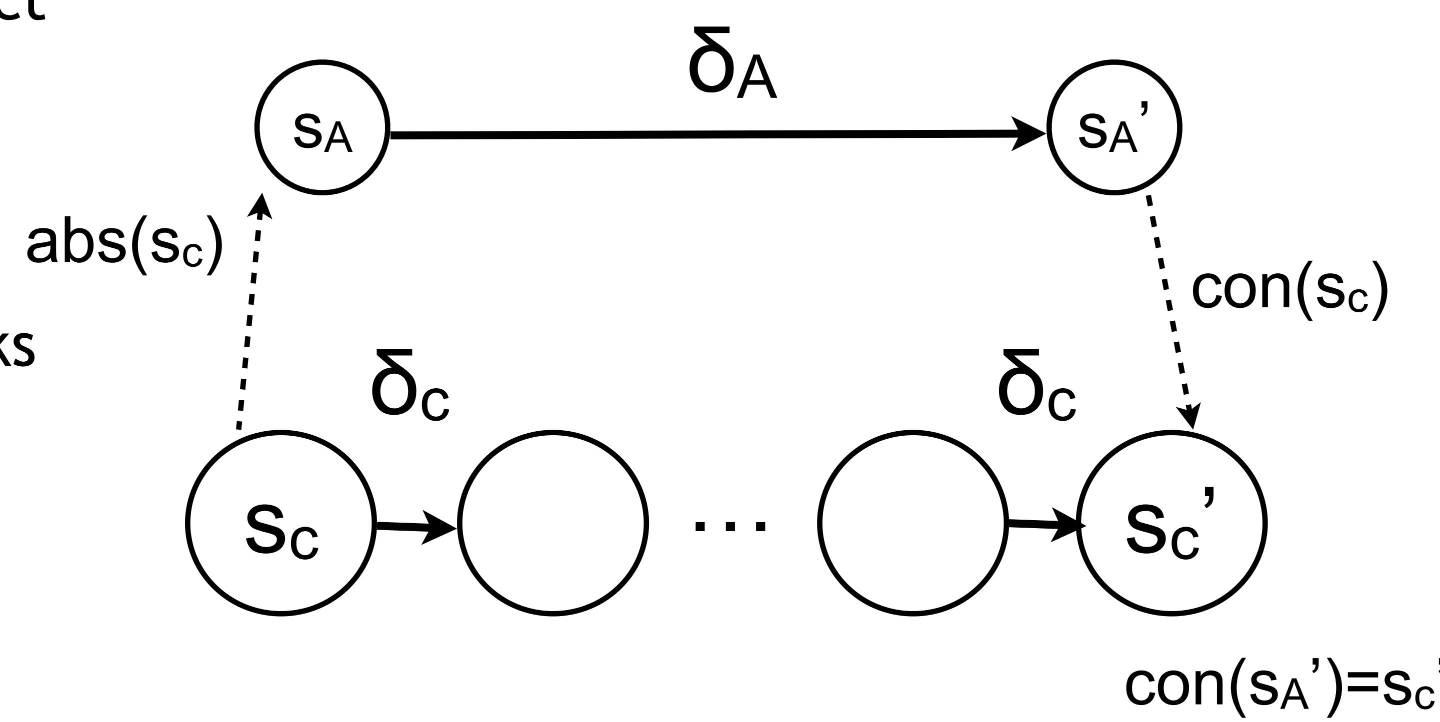
Formally Verifying Systems

- Bisimulation - Milner
  - comparison between abstract requirements
  - defines observational equivalence
- Abstraction Functions - Galois, Cousot
  - relationship between abstract and concrete state spaces
  - defines safety of abstractions
- Monadic Semantics - Moggi, Wadler
  - encapsulate computations
  - define stateful computation in pure languages

Bisimulation

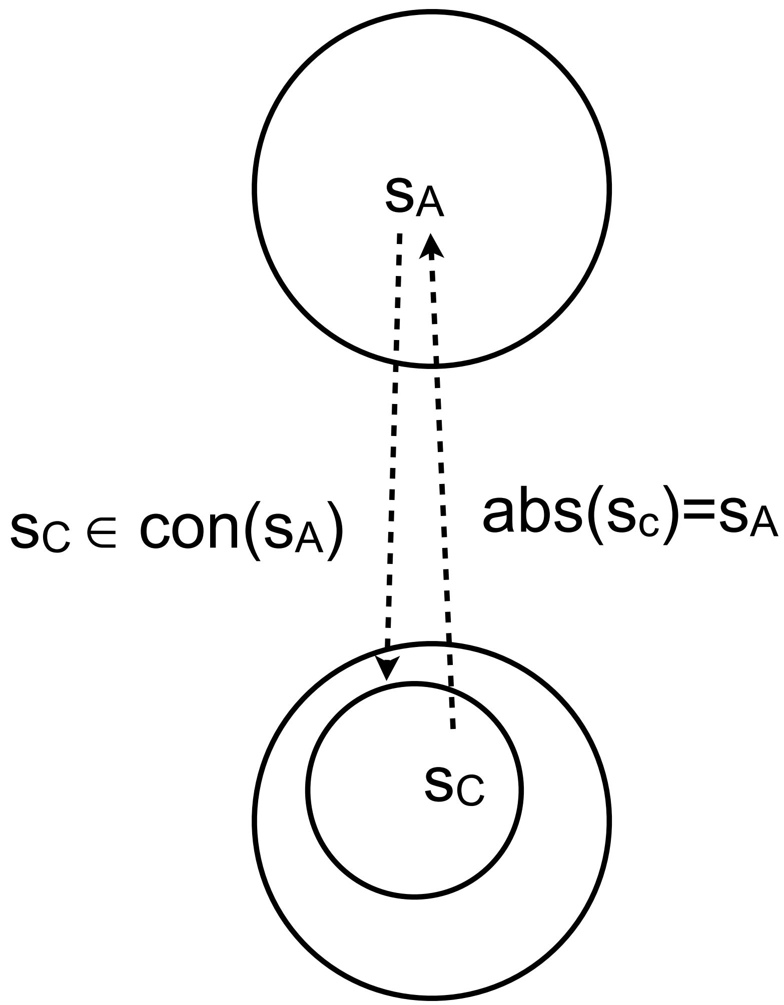
- Does a system meet requirements?
  - implementation *exhibits* requirements
  - homomorphism between abstract and concrete
- Behavioral equivalence
  - bisimulation between models
  - if it walks like a duck and quacks like a duck...
- Two dominant forms
  - bisimulation implies one-to-one state mapping
  - weak bisimulation ignores intermediate states
- Milner's model for equivalence in process algebras

$$abs(s_C)=s_A \Rightarrow abs(\delta_C^*(s_C'))= \delta_A(s_A')$$



Soundness of Abstraction

- Is the abstraction sound?
  - does it discard anything important?
  - properties of the abstraction should hold in the original
- Galois connection
  - between type of s_A and type of s_c
  - really only need half
- Cousot's model for soundness
  - widening and narrowing is alternative
  - Milner also addresses this in process algebras



Monads and Effectful Computation

- Monads encapsulate computations
  - `M` is the monad type constructor
  - `a` is the encapsulated computation
- `unit a` (also called “return”)
  - lifts a into the monad
  - result is trivial computation
- `m >>= f` (called “bind”)
  - forces computation of `M a`
  - calculates `M b` using result
  - returns new a monad ready to run
- Useful for
  - stateful computation semantics
  - completing partial functions
  - error value computation
  - context and environment

```
M a

unit : a → M a

>>= : M a → (a → M b) → M b

>> : M a → M b → M b

1. (return x) >>= f == f x
2. m >>= return == m
3. (m >>= f) >>= g ==
   m >>= λx . (f x >>= g)
```

Automated Proof

- Prototype Verification System (PVS)
  - typed higher-order logic with functions
  - automated rewriting
  - strong decision procedures
  - assistance for establishing $M \vdash \phi$
- Symbolic Analysis Laboratory (SAL)
  - typed linear temporal logic (LTL)
  - BDD and Buchi based checkers
  - deadlock, bounded, and infinite state checkers
  - automated determination of $M \models \phi$
- We verify virtually nothing by hand
  - proofs are not mathematically “interesting”
  - proofs are frequently numerous and huge
  - theory modification necessitates constant re-verification

And we're off…

The State Monad

```
StateMonad[A,S:TYPE+] : THEORY
BEGIN
 State : DATATYPE
 BEGIN
   state(runState:[S->[A,S]]):state?
 END State

 unit(x:A):State = state(LAMBDA (s:S) : (x,s));

 >>= (m:State,f:[A->State]):State =
   state(LAMBDA(s0:S):
          LET (a,s1) = runState(m)(s0) IN
           runState(f(a))(s1));

 >> (m:State,f:State):State =
   state(LAMBDA(s0:S):
          LET (a,s1) = runState(m)(s0) IN
            runState(f)(s1));
```

Encapsulated computation

Lift a state into the monad

Perform the first computation and thread the state through

Perform the second computation with the result of the first

The Monad Laws

```
left_identity: LAW FORALL (a:A,f:[A->State]) :
   unit(a) >>= f = f(a)

right_identity: LAW FORALL (m:State) :
   m >>= unit = m

assocativity: LAW FORALL (m:State,f,g:[A->State]) :
   m >>= f >>= g = m >>= (lambda(x:A): f(x) >>= g)
```

All three laws verified automatically using rewriting and structural equivalence

Useful State Operations

```
% Replace current state
put(a:A,s1:S) : State =
  state(LAMBDA(s0:S):(a,s1))

% Modify current state
modify(a:A,f:[S->S]) : State =
  state(LAMBDA(s0:S):(a,f(s0)))

% Generate output from state
output(f:[S->[A,S]]) : State =
  state(LAMBDA(s0:S):f(s0))

END StateMonad;
```

TPM State Monad

```
tpmState : TYPE = [#
                    srk : (asymKey?),
                    eik : (asymKey?),
                    keys : KEYSET,
                    pcrs : PCRS,
                    locality : LOCALITY
                 #];
```

TPM state

```
IMPORTING StateMonad[tpmOutput,tpmState];
```

State monad with TPM state as S and TPM output as A

```
%% Unknown state
tpmUnknown : tpmState

%% Power on state
tpmPower : tpmState =
  (# pcrs:=pcrsPower, locality:=0, keys:=emptyset,
     srk:=inverse(srkVal), eik:=inverse(eikVal) #);
```

Useful TPM state values

KU INFORMATION & TELECOMMUNICATION TECHNOLOGY CENTER The University of Kansas

TPM Command Definitions

```
tpmExtend(s:tpmState,n:PCRINDEX,h:HV) : tpmState =
  s WITH [`pcrs := pcrsExtend(pcrs(s),n,h)];
```

Extending a PCR

```
extendPCR(h:HV,n:PCRINDEX):State =
  modify(outNothing,(LAMBDA (s:tpmState): tpmExtend(s,n,h)));
```

```
tpmDecryptKey(s:tpmState,d:(encrypt?)) : BLOB =
  CASES key(d) OF
    wrapKey(w,k) : IF member(inverse(w),add(srk(s),keys(s)))
                   THEN blob(d)
                   ELSE nothing
                   ENDIF
    ELSE nothing
  ENDCASES;
```

Unwrapping a key

```
decryptTPM(d:(encrypt?)) : State =
  output((LAMBDA (s:tpmState):(outBlob(tpmDecryptKey(s,d)),s)));
```

Sequencing TPM Commands

```
runState(powerTPM
         >> senterResetTPM
         >> senterHashSinitTPM
         >> installKeyTPM(k)
         >> decryptTPM(encrypt(k,b))
         >>= unit)
        (tpmUnknown)
```

Using the state monad

Which is clearer?

```
decryptTPM(encrypt(k,b),
  installKeyTPM(k,
   senterHashSinitTPM(
    senterResetTPM(
     powerTPM(tpmUnknown))))
```

Using explicit state passing

And Finally a Theorem

```
replay_detection: THEOREM FORALL (d:B, asp:B, n0,n1:B) :
      LET (q0,s0) = runState(
                     extendPCR(hash(asp),0)
                    >> extendPCR(hash(n0),0)
                    >> extendPCR(hash(d),0)
                    >> quotePCR
                    >>= unit)
                    (tpmReset),
          (q1,s1) = runState(
                     extendPCR(hash(asp),0)
                    >> extendPCR(hash(n1),0)
                    >> extendPCR(hash(d),0)
                    >> quotePCR
                    >>= unit)
                    (tpmReset) IN
        n0/=n1 => q0/=q1;
```

Extend the PCR with n0

Extend the PCR with n1

Proved using skolemization, rewriting and structural equivalence:

PVS> (grind)
PVS> (decompose-equality)

If the nonces are different, the quotes are different

Sample Theorems

- Ordering lemmas
  - PCR extension is antisymmetric
  - skipping senter is detectable
  - skipping reset is detectable
  - reset takes us to a known state
  - quote returns the correct PCRs
- Boot integrity
  - wrong MLE element boot detectable via quote
  - wrong boot order detectable via quote
- Key installation
  - wrapped keys are not installed if wrapping key is not installed
  - key chaining has integrity
  - unsealing secrets has integrity

Status and Future Work

- Current model is defined at the requirements level
  - identify correctness conditions and continue verification
  - complete the instruction set model
  - verify important command sequences
- Verify implementation model
  - define state and state monad
  - define and verify abstraction function from concrete state to abstract state
  - verify bisimulation relationship for command set

Things to think about...

- Correctness by construction
  - techniques demonstrating correctness at synthesis time
  - techniques for composing correct components into correct systems
- Coalgebraic/comonadic specification
  - better for non-terminating systems
  - compilation from coalgebras to operational systems
- Defining institutions among semantic domains
  - understand and verify interactions among semantically different models
  - examine emergent properties of systems-of-systems
- Type systems
  - dependent typing and applications to verifiation
  - encoding and checking invariants as types
  - encoding and checking general proof conditions as types
- Stop worrying about the proof
  - finding the proof automatically - like type checking
  - just show that it exists and move on