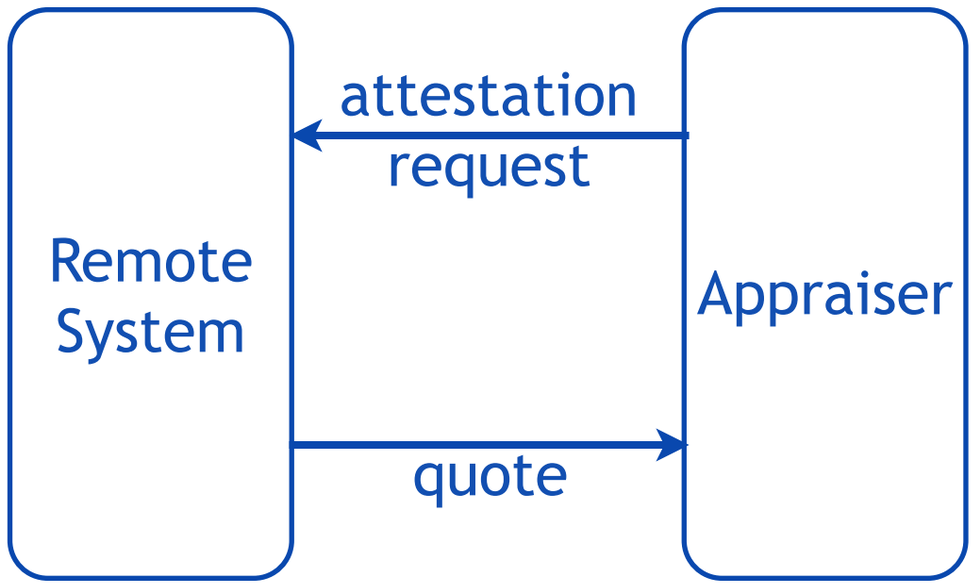
Verifying the Trusted Platform Module

Perry Alexander, Brigid Halling
Information and Telecommunication Technology Center
Electrical Engineering and Computer Science
The University of Kansas
{palexand,bhalling}@ku.edu

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

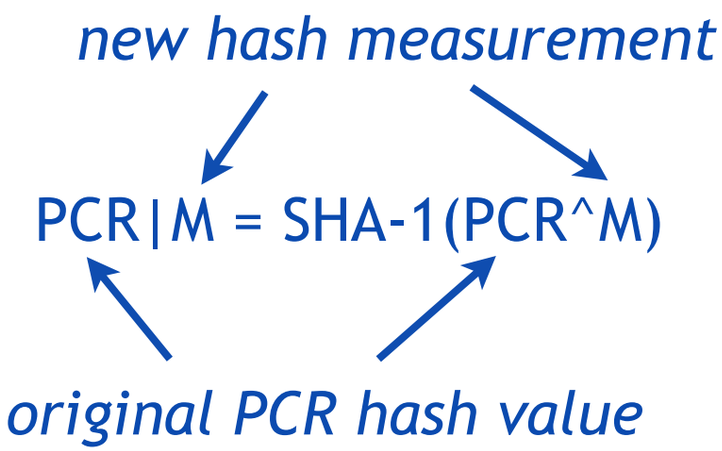


The TPM's Role

- Provides and Protects Roots of Trust
  - Storage Root Key (SRK) - root of trust for storage
  - Endorsement Key (EK) - root of trust for reporting
- Quote generation
  - high integrity quotes - ($\{|RS|\}_{AIK^{-1}}$, SML, $\{|n,PCR_{0\text{-}m}|\}_{AIK^{-1}}$)
  - high integrity evidence - (<E,n>, $\{|\#E,PCR,n|\}_{AIK^{-1}}$)
- Sealing data to state
  - $\{D,PCR\}_K$ will not decrypt unless PCRs = current PCRs
  - data is safe even in the presence of malicious machine
- Binding data to TPMs and machines
  - ($\{K^{-1}\}_{SRK}$,K) - $\{D\}_K$ cannot be decrypted unless SRK is installed
  - ($\{J^{-1}\}_K$,J) - $\{D\}_j$ cannot be decrypted unless K and SRK are installed

Platform 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

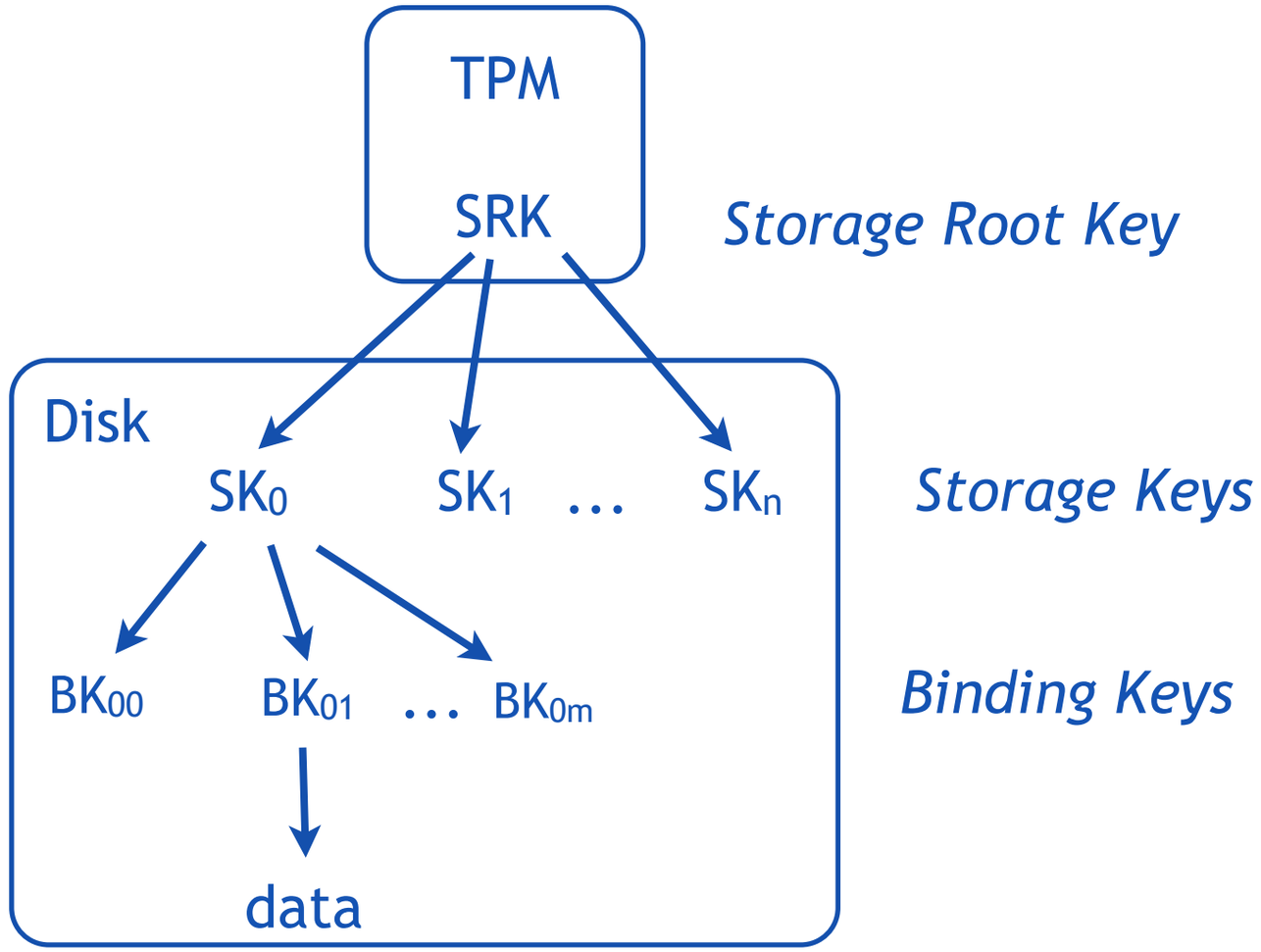


PCR|M ≠ M|PCR

order matters!

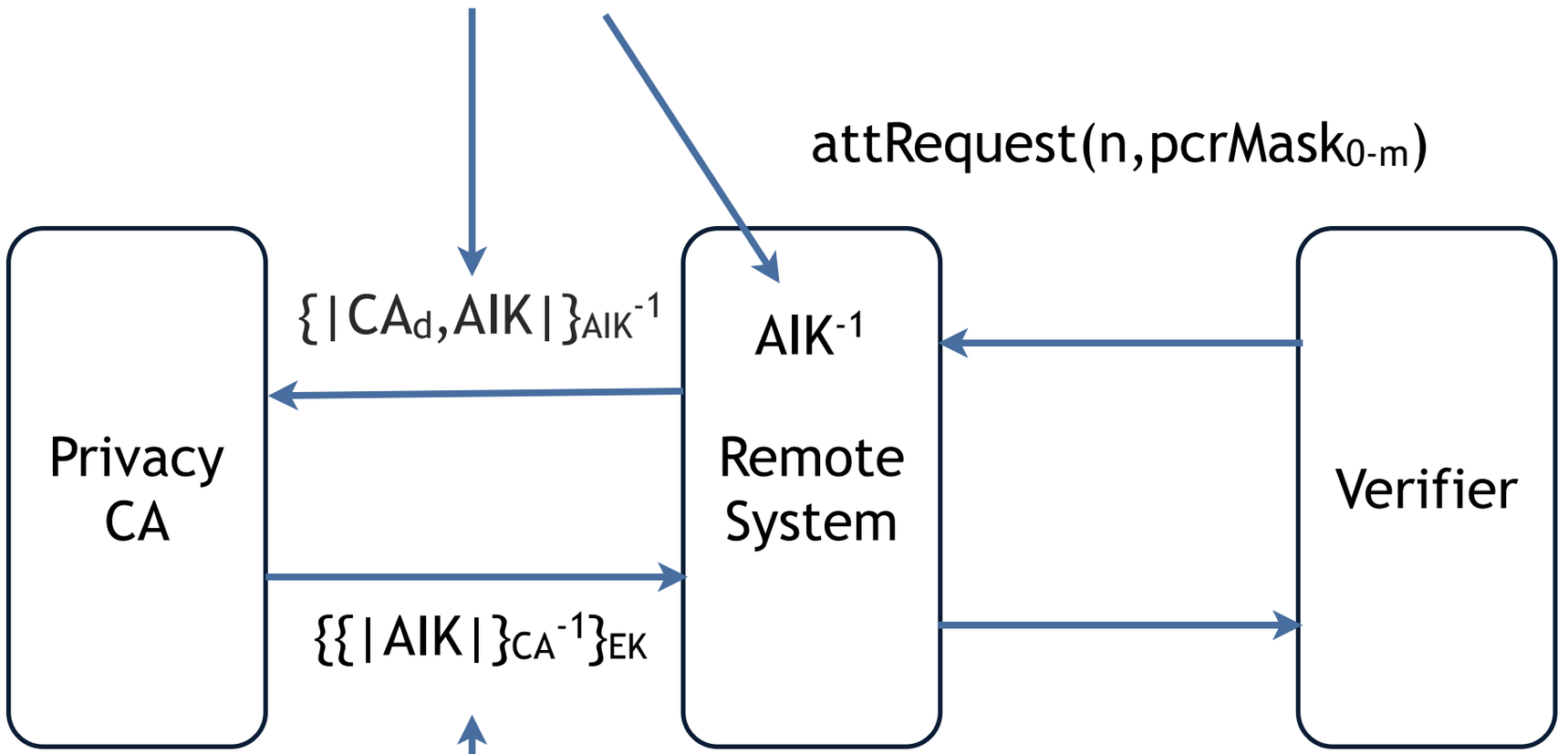
Keys and Data

- 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



Generating Quotes

Fresh (AIK,AIK^{-1}) generated by TPM_MakeIdentity



PCR_0 = Reference PCR_0
PCR_1 = Reference PCR_1
PCR_2 = Reference PCR_2

$PCR_k \neq$ Reference PCR_k
...
PCR_m = Reference PCR_m

$(\{|\{|AIK|\}_{CA^{-1}}|\}_{AIK^{-1}}, \{|n,PCR_{0-m}|\}_{AIK^{-1}})$

Certificate binds AIK to Remote System
Decrypted by TPM_ActivateIdentity

Quote generated by TPM_Quote

The TPM Specification

- Developed by the Trusted Computing Group
  - 1.2 fielded in most enterprise PCs
  - 2.0 awaiting formal approval
  - coupled with a Trusted Software Stack (TSS) definition
  - virtualization and mobile specifications under development
- Structured English specification
  - 700 pages over three volumes
  - tables and text much like a CPU description

TPM Specification

16.1 TPM_Extend

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_Extend.
4	20	3S	20	TPM_PCRVALUE	outDigest	The PCR value after execution of the command.

16.1 TPM_Extend

Actions

1. Validate that pcrNum represents a legal PCR number. On error, return TPM_BADINDEX.
2. Map L1 to TPM_STANY_FLAGS -> localityModifier
3. Map P1 to TPM_PERMANENT_DATA -> pcrAttrib [pcrNum]. pcrExtendLocal
4. If, for the value of L1, the corresponding bit is not set in the bit map P1, return TPM_BAD_LOCALITY
5. Create c1 by concatenating (TPM_STCLEAR_DATA -> PCR[pcrNum] || inDigest). This takes the current PCR value and concatenates the inDigest parameter.
6. Create h1 by performing a SHA-1 digest of c1.
7. Store h1 to TPM_STCLEAR_DATA -> PCR[pcrNum]
8. If TPM_PERMANENT_FLAGS -> disable is TRUE or TPM_STCLEAR_FLAGS -> deactivated is TRUE
   a. Set outDigest to 20 bytes of 0x00
9. Else
   a. Set outDigest to h1

"It is easier to be gigantic than beautiful"

- Friedrich Nietzsche

Verifying the TPM?

- Define and verify abstract specification of TPM behavior
  - typed, abstract data representation
  - abstract state state transformation
  - preconditions, postconditions, and invariants
  - command sequencing using state monad
- Validate using abstract models for common protocols
  - certificate authority based attestation
  - direct anonymous attestation
  - data and key migration
- Develop a concrete specification of TPM behavior
  - bit-level description of state and state transformation
  - verify commands and protocols
  - verify weak bisimulation between abstract and concrete models

Command Specification

- Commands generate output and modify state
- Defined by cases over
  - `tpmAbsInput` - abstract command type
  - `tpmAbsState` - abstract state type
  - `tpmAbsOutput` - abstract output type
- Command sequencing using a restricted state monad

```
TPM_ActivateIdentity(a:(wrapKey?),k:(symKey?)): State =
    modifyOutput(
        (LAMBDA (s:tpmAbsState) :
             outputCom(s,ABS_ActivateIdentity(a,k)))
        (LAMBDA (s:tpmAbsState) :
             executeCom(s,ABS_ActivateIdentity(a,k))));
```

Modeling State

```
tpmAbsState : TYPE =
[#
  restore : restoreStateData,
  memory : mem,
  ek : (asymKey?),
  srk : (asymKey?),
  keyGenCnt : K,
  keys : KEYSET,
  pcrs : PCRS,
  permData : PermData,
  permFlags : PermFlags,
  stanyData : StanyData,
  stanyFlags : StanyFlags,
  stclearData : StclearData,
  stclearFlags : StclearFlags
#];
```

- srk - Storage Root Key
- ek - Endorsement Key
- pcrs - Platform Configuration Registers

Modeling State Transformation

```
executeCom(s:tpmAbsState,c:tpmAbsInput) : tpmAbsState =
  CASES c OF
    ABS_LoadKey2(k): loadKey2State(s,k),
    ABS_Extend(h,n) : extendState(s,n,h),
    ABS_ActivateIdentity(a,k) : activateIdentityState(s,a,k),
    ...
    ELSE s
  ENDCASES;

activateIdentityState(s:tpmAbsState,a:(wrapKey?),k:(symKey?)):
tpmAbsState =
    s WITH [`keys := addKey(a,srk(s),keys(s))];
```

Modeling Output

```
outputCom(s:tpmAbsState,c:tpmAbsInput) : tpmAbsOutput =
  CASES c OF
     ABS_Seal(k,data) = sealOut(s,k,data),
     ABS_MakeIdentity(a,k,auth) : makeIdentityOut(s,a,k,auth),
     ABS_ActivateIdentity(a,k) : activateIdentityOut(s,a,k),
     ...
     ELSE outNothing
  ENDCASES;
```

```
activateIdentityOut(s:tpmAbsState,a:(wrapKey?),k:(symKey?)) :
tpmAbsOutput =
  IF checkKeyRoot(a,srk(s))
     THEN outSymKey(k)
     ELSE outNothing
  ENDIF;
```

Modeling Command Sequencing

```
aik_usage: THEOREM
  FORALL (aik:(tpmKey?),
          b:(tpmNonce?),
          pm:PCR_SELECTION,
          state:tpmAbsState):
    LET (a,s) = runState(
                TPM_Init
                >> TPM_Startup(TPM_ST_CLEAR)
                >> CPU_senter
                >> CPU_sinit
                >>= TPM_Quote(aik,b,pm)
                (state)
IN NOT checkKeyRoot(aik,srk(s)) =>
      a=OUT_Error(TPM_INVALID_KEYUSAGE);
```

sequence

bind

Verification and Validation

- Define and verify state invariants over state change
  - establishes safety properties with respect to all commands
  - uses PVS type system extensively
- Verify individual command execution
  - establishes individual command execution correctness
  - define and verify command pre- and postconditions
  - verify state invariants over state change
- Define and verify common protocols
  - establishes validity of command and sequencing model
  - define and verify protocol pre- and postconditions
  - verify invariants over protocol execution

Command Postconditions

```
activate_identity_post : THEOREM
  FORALL (aik:(wrapKey?),k:(symKey?)) :
    LET (out,s) = runState(
      TPM_ActivateIdentity(aik,k)
      >>= unit)
      (tpmStartup) IN
out = IF checkKeyRoot(aik,srk(s))
      THEN outSymKey(k)
      ELSE outNothing
      ENDIF
AND checkKeyRoot(aik,srk(s)) =>
  member(wkey(aik),keys(s));
```

execute command

verify output is correct

verify state is correct

Command Invariants

```
srk_unchanged: THEOREM
  (FORALL (s:tpmAbsState,
           c:tpmAbsInput):
    NOT(ABS_Init?(c)
        OR ABS_Startup?(c)
        OR ABS_TakeOwnship?(c)) =>
      srk(s) = srk(executeCom(s,c)));
```

```
pcrs_unchanged: THEOREM
  (FORALL (s:tpmAbsState,c:tpmAbsInput):
    NOT(ABS_Startup?(c)
        OR ABS_Init?(c)
        OR ABS_sinit?(c)
        OR ABS_senter?(c)
        OR ABS_Extend?(c)) =>
    pcrs(s) = pcrs(executeCom(s,c)));
```

- Invariants include
  - keys don't change
  - PCRs don't change
  - locality monotonically increases
  - flags and permissions don't change
  - installed keys don't change

Predicate Subtypes

- Using the PVS type-checker to manage verification

```
wellFormed?(s:tpmAbsState):bool =
    wellFormedRestore?(restore(s)) AND ... ;

wellFormedRestore?(r:restoreStateData) : bool =
  valid?(r) =>
  FORALL (i:PCRINDEX) :
      pcrReset(pcrAttrib(permData(r))(i)) =>
        pcrs(r)(i) = resetOne;

stateTransformation : [(wellFormed?)->(wellFormed?)]
```

Generating a Quote

Fresh (AIK,AIK^{-1}) generated by TPM_MakeIdentity

attRequest(n,$pcrMask_{0\text{-}m}$)

Trusted Third Party

$\{|CA_d, AIK|\}_{AIK^{-1}}$

$\{\{|AIK|\}_{CA^{-1}}\}_{EK}$

AIK^{-1}

Remote System

Verifier

PCR_0 = Reference PCR_0
PCR_1 = Reference PCR_1
PCR_2 = Reference PCR_2

$PCR_k \neq$ Reference PCR_k
...
PCR_m = Reference PCR_m

$(\{|\{|AIK|\}_{CA^{-1}}|\}_{AIK^{-1}}, \{|n, PCR_{0\text{-}m}|\}_{AIK^{-1}})$

Certificate binds AIK to Remote System
Decrypted by TPM_ActivateIdentity

Quote generated by TPM_Quote

Privacy CA Attestation Protocol

```
CA_Protocol: THEOREM
  FORALL (aik:(asymKey?), state:tpmAbsState, n:BLOB, pm:PCRMASK,
          k:(symKey?), sml:SML, auth:AUTHDATA):
   LET (out,s) = runState(
       TPM_Init
       >> TPM_Startup
       >> CPU_senter
       >> CPU_sinit
       >> TPM_MakeIdentity(aik,k,auth)
       >>= CPU_saveOutput(0)
       >>= LAMBDA (a:tpmAbsOutput) :
         IF outIdentity?(a) THEN CA_certify(oidentc(a)) ELSE TPM_Noop(a) ENDIF
    >>= CPU_saveOutput(1)
    >>= LAMBDA (a:tpmAbsOutput) :
         IF outIdentActivation?(a) AND wrapKey?(key(oactc(a)))
            THEN TPM_ActivateIdentity(key(oactc(a)),k) ELSE TPM_Noop(a) ENDIF
    >> CPU_read(0) % identity is output
    >>= LAMBDA (a:tpmAbsOutput) :
         IF outIdentity?(a)
            THEN TPM_Quote(oidentaik(a),n,pm) ELSE TPM_Noop(a) ENDIF
    >>= CPU_saveOutput(2)
    >> CPU_BuildQuoteFromMem(2,1,1,sml))(state)
```

Privacy CA Correctness Condition

```
LET key=idKey(s`memory(x)) IN
  makeIdentity?(state,k) AND OUT_MakeIdentity?(s`memory(x)) AND
  certify?(key,idBinding(s`memory(x))) AND OUT_Certify?(s`memory(y)) AND
  wellFormedRestore?(s`restore) AND
  activateIdentity?(tpmRestore(s`restore),key,dat(s`memory(y))) AND
  quote?(key) AND OUT_Quote?(s`memory(z))
  =>                                                    output is correct
  a=OUT_FullQuote(tpmQuote(tpmCompositeHash((#select:=p,pcrValue:=s`pcrs#)),
                           n,signed(private(key),clear)),
                  tpmIdContents(d,key,signed(private(key),clear)),
                  TPM_SUCCESS)
AND                                                     state is correct
s=state WITH [`keyGenCnt:=state`keyGenCnt+2, `memory:=s`memory]
```

Other Correctness Theorems

- Ordering lemmas
  - PCR extension is antisymmetric
  - skipping senter, sinit, or reset is detectable
  - 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
- Protocols
  - CA attestation protocol
  - boot protocol

Current Status

- 40 of approximately 90 instructions modeled
  - focusing thus far critical functionality
  - startup, key management, quote, privacy CA attestation, migration modeled
  - session management, DAA, and flag configuration not modeled
- Effort thus far
  - 3712 LoPVS
  - 120 proofs, most run in a few seconds
  - 1 full time + 1 part time developer for just under a year
- PVS sources are available
  - long term - website coming soon
  - short term - contact palexand@ku.edu

Moving Forward

- Develop better threat model
  - currently assuming simple Yolev-Dao
- Additional TPM features
  - session and authdata management
  - locality enforcement
  - Direct Anonymous Attestation (DAA) support commands
  - auditing and logging
- Additional protocols
  - data migration protocols
  - DAA protocols
  - more complex attestation protocols
- Model and theorem synthesis
  - translation of TPM code sequences to PVS model
  - automated proof synthesis for basic proofs
- vTPM extensions

Related Work

- L. Chen, and B. Warinschi. “Security of the TCG Privacy-CA solution”, in *Proceedings of the 2010 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing, (EUC’10)*, pages 609–616, Washington, DC, USA, 2010.
  - Security of the enhanced TCG Privacy-CA solution.
  - Security of the TCG Privacy-CA solution.
- S. Gürgens, C. Rudolph, D. Scheuermann, M. Atts, and R. Plaga. “Security evaluation of scenarios based on the TCGs TPM specification”, *Computer Security ESORICS 2007*, pages 438–453, 2007.
  - Security evaluation of scenarios based on the TCGs TPM specification.
- S. Delaune, S. Kremer, M. D. Ryan, and G. Steel. “A formal analysis of authentication in the TPM,” In Proceedings of the *Seventh International Workshop on Formal Aspects in Security and Trust* (FAST’10). Springer, 2010.
  - Formal analysis of authentication techniques in the TPM
- S. Delaune, S. Kremer, M. Ryan, and G. Steel. “Formal analysis of protocols based on TPM state registers,” In Proceedings of *The 24th IEEE Computer Security Foundations Workshop (CSF 2011)*, pages 66–82, 2011.
  - Formal analysis of protocols based on TPM state registers