



# Verifying the Trusted Platform Module

Brigid Halling, Perry Alexander

Information and Telecommunication Technology Center

Electrical Engineering and Computer Science

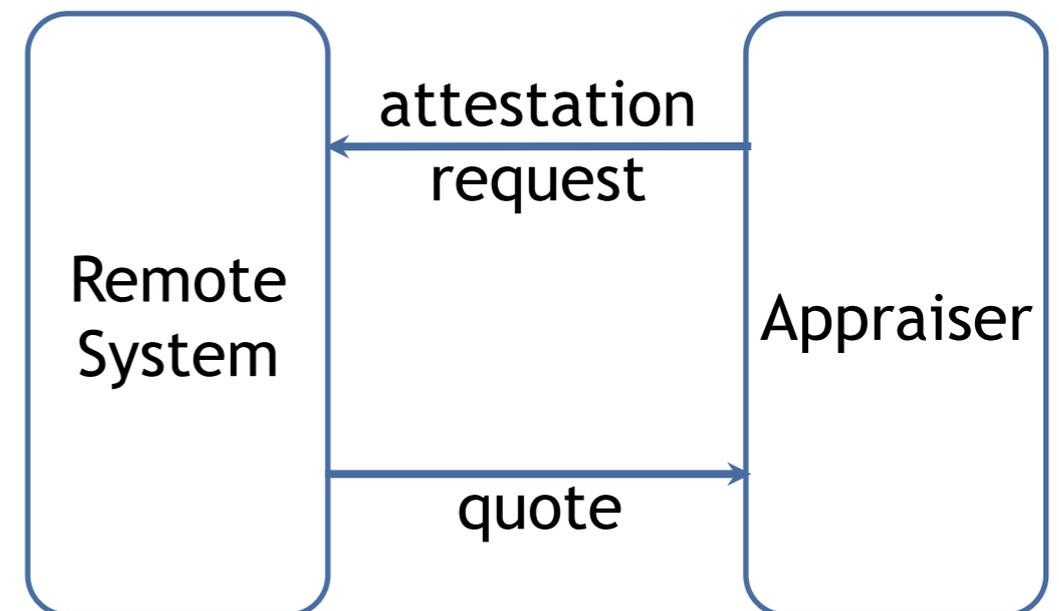
The University of Kansas

{bhalling,palexand}@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



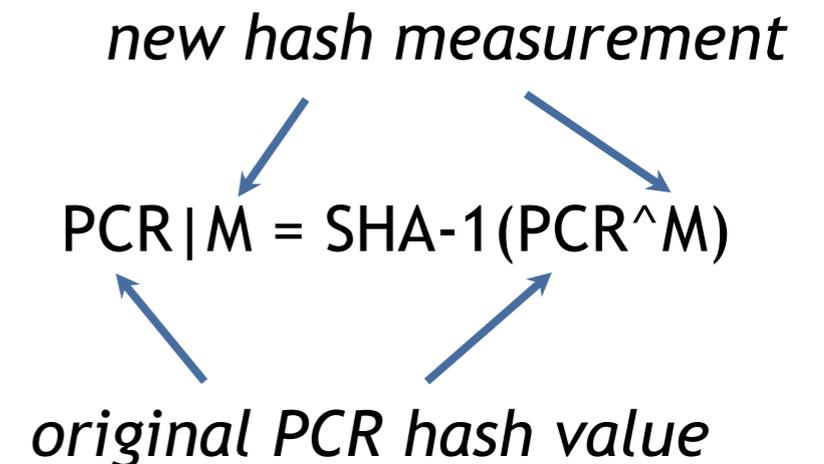
# The TPM's Role

- 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 -  $(\{|RS|\}_{AIK^{-1}}, SML, \{|n, PCR_{0-m}|\}_{AIK^{-1}})$
  - high integrity evidence -  $(\langle E, n \rangle, \{|\#E, PCR, n|\}_{AIK^{-1}})$
- 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



# 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

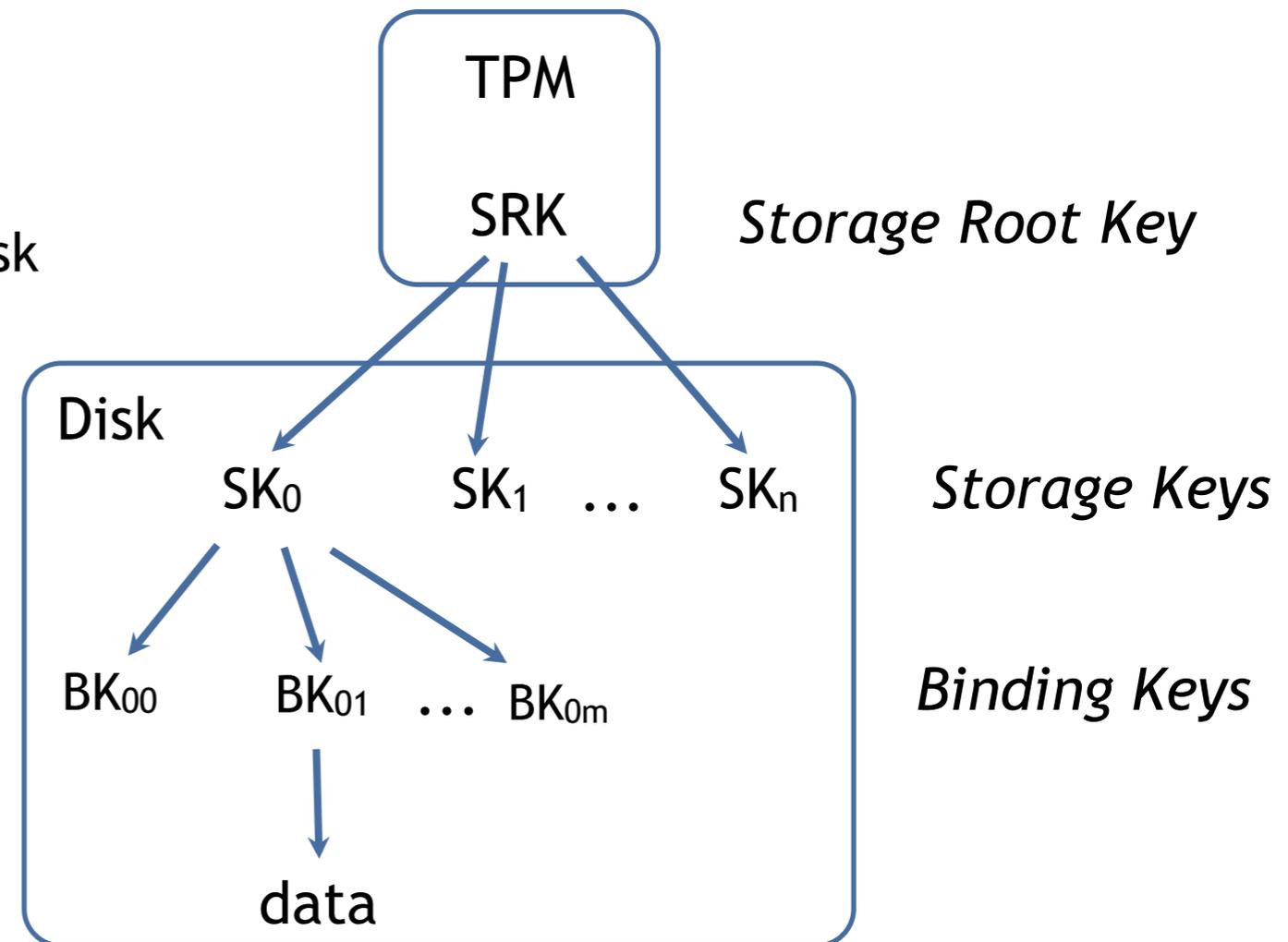


$PCR|M \neq M|PCR$   
*order matters!*



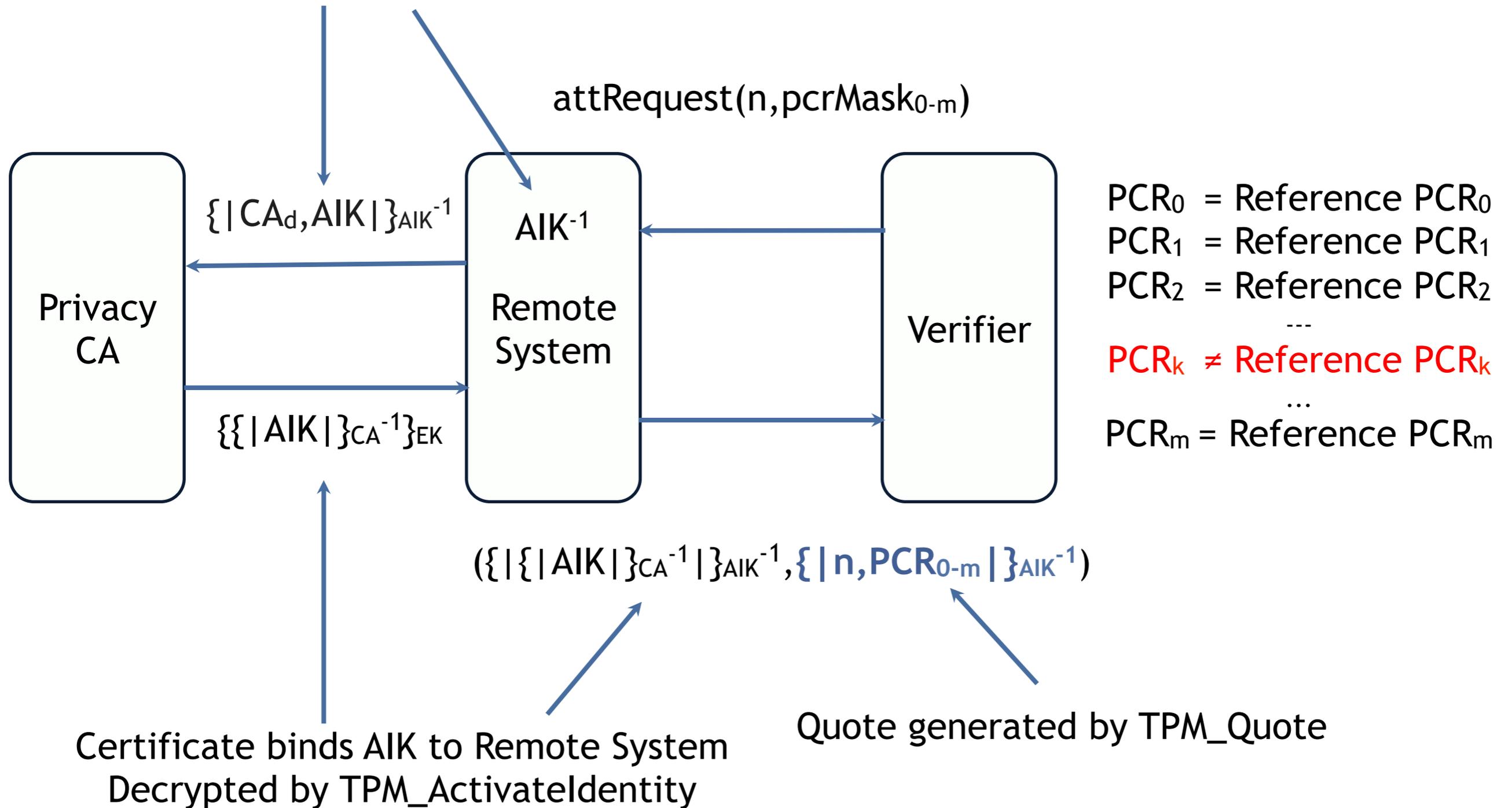
# Keys and Data

- 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



# 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



# 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(p,d) : extendState(s,p,d),  
  ...  
ELSE ENDCASES;
```

```
extendState(s:tpmAbsState,p:PCRINDEX,d:HV) : tpmAbsState =  
  IF 0 <= p <= 23  
    THEN s WITH [ `pcrs := pcrsExtend(s`pcrs,p,d) ]  
    ELSE s  
  ENDIF
```



# Modeling Output

```
outputCom(s:tpmAbsState,c:tpmAbsInput) : tpmAbsOutput =  
  CASES c OF  
    ABS_Seal(k,data) : sealOut(s,k,data),  
    ABS_Extend(p,d) : extendOut(s,p,d),  
    ...  
  ENDCASES;
```

```
extendOut(s:tpmAbsState,p:PCRINDEX,d:HV) : tpmAbsOutput =  
  LET H1=pcrsExtend(s`pcrs,p,d) IN  
  IF p > 23 OR p < 0  
    THEN OUT_Error(TPM_BADINDEX)  
  ELSIF s`permFlags`disable OR s`stclearFlags`deactivated  
    THEN OUT_Extend(reset,TPM_SUCCESS)  
    ELSE OUT_Extend(extend(s`pcrs(p),d),TPM_SUCCESS)  
  ENDIF;
```



# Modeling Command Sequencing

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



# 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

```
extend_post : THEOREM
  FORALL (state:(afterStartup?), p:PCRINDEX, d:HV) :
  LET (a,s) = runState(
    TPM_Extend(p,d)
    (state) IN
  LET H1=pcrsExtend(state`pcrs,p,d) IN
  IF 0 <= p <= 23
  THEN s=state WITH [ `pcrs:=H1] AND
    IF state`permFlags`disable OR
      state`stclearFlags`deactivated
    THEN a=OUT_Extend(reset,TPM_SUCCESS)
    ELSE a=OUT_Extend(extend(state`pcrs(p),d),TPM_SUCCESS)
    ENDIF
  ELSE a=OUT_Error(TPM_BADINDEX) AND s=state
  ENDIF;
```

**execute command**

**verify state is correct**

**verify output is correct**



# Command Invariants

**srk\_unchanged:** THEOREM

```
(FORALL (s:tpmAbsState, c:tpmAbsInput):  
  NOT(ABS_Init?(c)  
    OR ABS_Startup?(c)  
    OR ABS_TakeOwnership?(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 ... ;
```

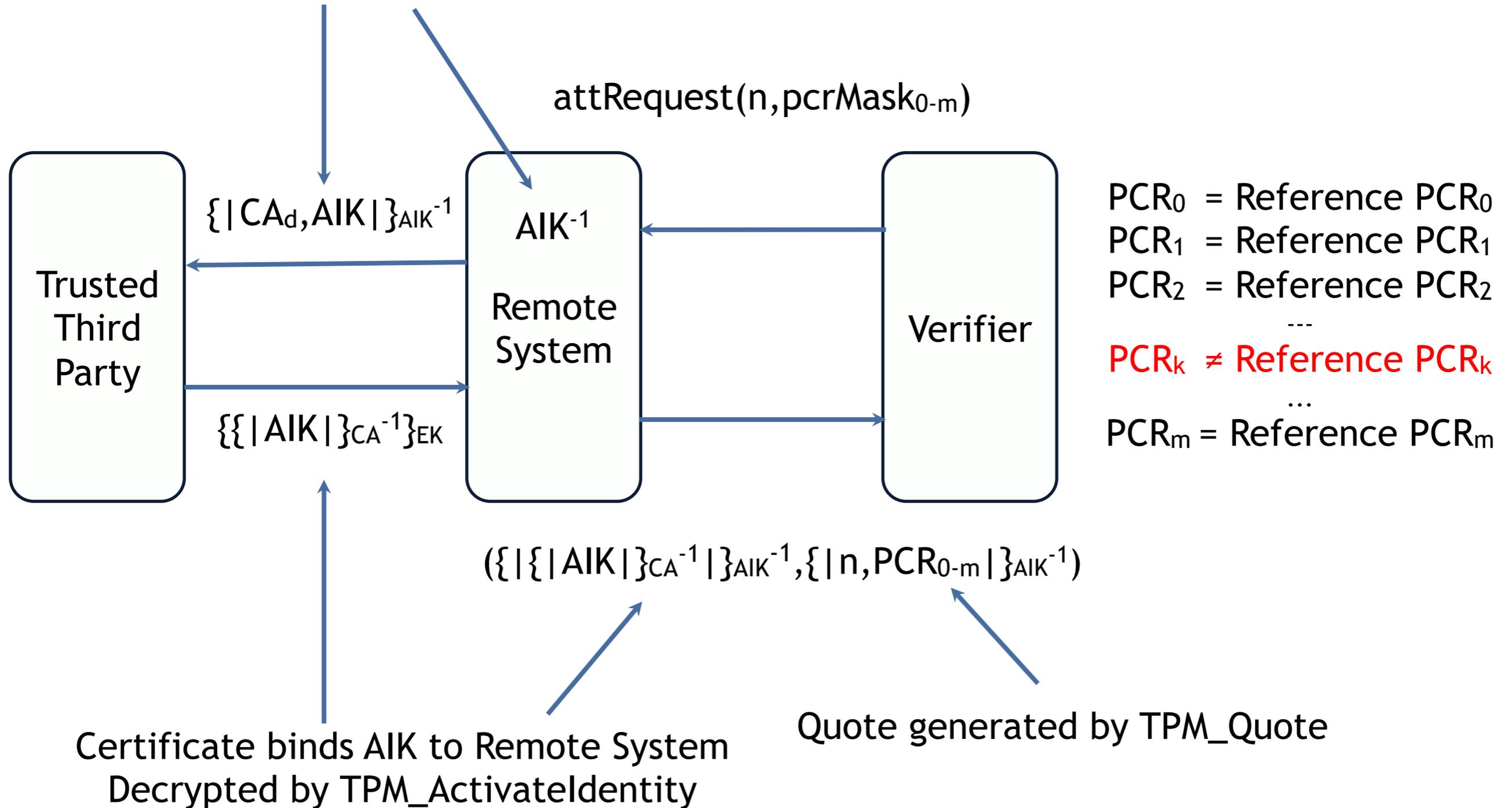
```
wellFormedRestore? (r:restoreStateData) : bool =  
  valid?(r) =>  
  FORALL (i:PCRINDEX) :  
    pcrReset(pcrAttrib(permData(r))(i)) =>  
    pcrs(r)(i) = resetOne;
```

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



# Generating a Quote

Fresh  $(AIK, AIK^{-1})$  generated by TPM\_MakeIdentity



# Privacy CA Attestation Protocol

ca\_protocol:THEOREM

```
FORALL (state:(afterStartup?), d:(tpmDigest?), k:(tpmKey?),
        n:(tpmNonce?), p:PCR_SELECTION, x, y, z:nat) :
LET (a, s) = runState(TPM_MakeIdentity(d, k)
    >>= CPU_saveOutput(x)
    >>= CA_certify(k, i)
    >>= CPU_saveOutput(y)
    >>= TPM_ActivateIdentity(j, d)
    >> CPU_read(x)
    >>= TPM_Quote(k, n, p)
    >>= CPU_saveOutput(z)
    >> CPU_BuildQuoteFromMem(z, x)) (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)) =>
a=OUT_FullQuote(tpmQuote(
  tpmCompositeHash((#select:=p,pcrValue:=s`pcrs#)),
  n,signed(private(key),clear)),
  tpmIdContents(d,key,signed(private(key),clear)),
  CPU_SUCCESS) AND
s=state WITH [`keyGenCnt:=state`keyGenCnt+2, `memory:=s`memory]
```

output is correct

state is correct



# Other Correctness Theorems

- Ordering lemmas
  - PCR extension is antisymmetric
  - skipping sender, 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](mailto: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

