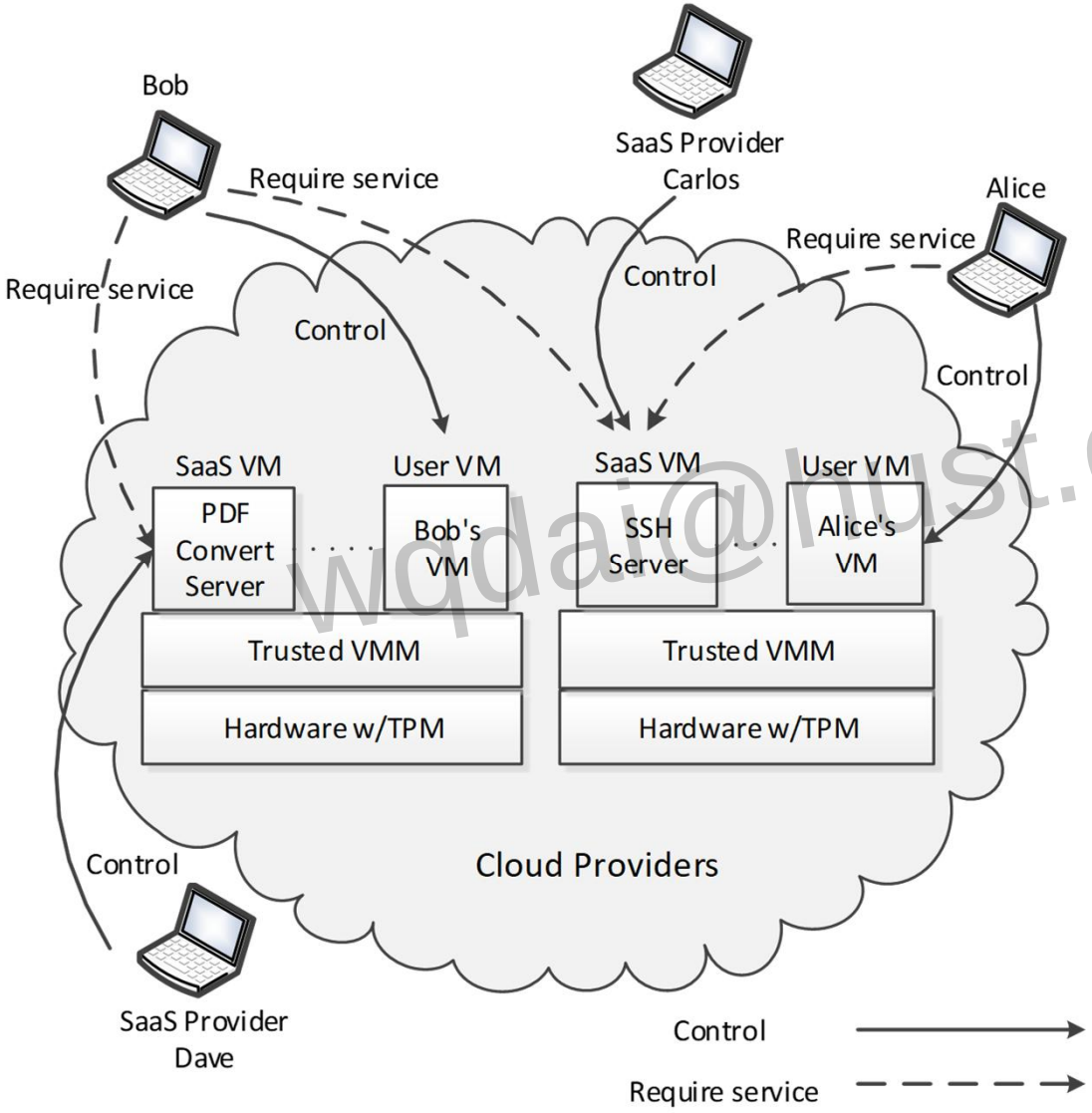
虚拟机安全回滚机制

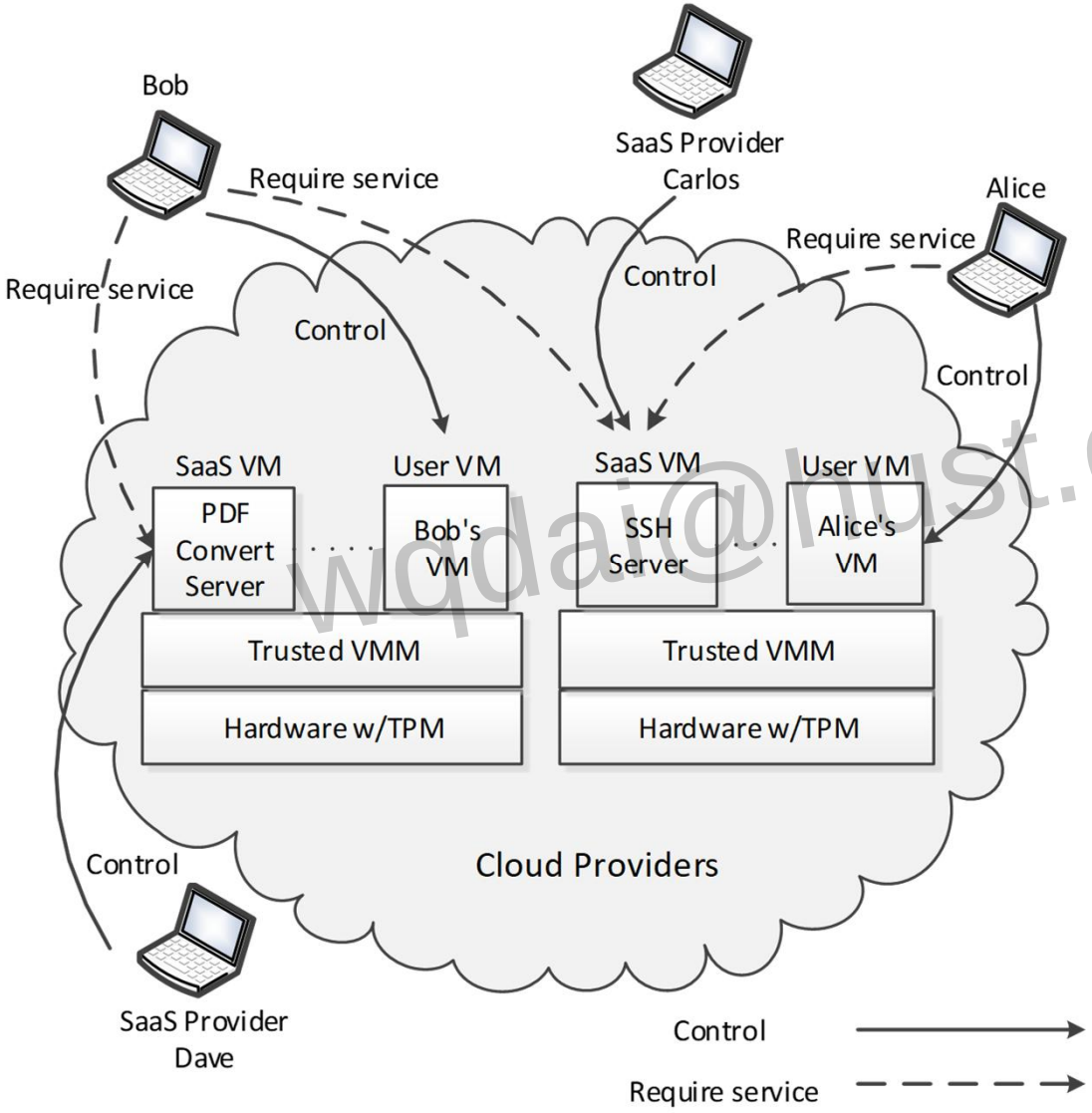
- 问题：
  - 可信计算要求安全状态不可逆转，而云平台支持虚拟机快照与回滚的特点与可信计算相违背，比如一个虚拟机从安全状态回滚到不安全状态，从而导致安全状态不一致。因此如何消除虚拟机回滚造成的影响是一个重要的安全挑战。
  - 现有的虚拟化可信平台芯片（vTPM）并不支持回滚，无法在虚拟机回滚的同时将自身状态回滚到快照时刻的状态。
  - 简单的回滚vTPM数据，也会造成新产生数据的丢失，并且会带来重放攻击。

云环境场景分析-安全状态不一致



SaaS服务商Carlos提供SSH服务给云用户Alice前，正常启动SaaS VM（处于状态$S_{trusted}$），Alice在远程验证后信任该服务，之后Carlos再回滚虚拟机到包含恶意代码的状态($S_{untrusted}$)，由于现有的vTPM不会回滚，因此无论Alice何时验证，VM都将属于可信状态。

云环境场景分析

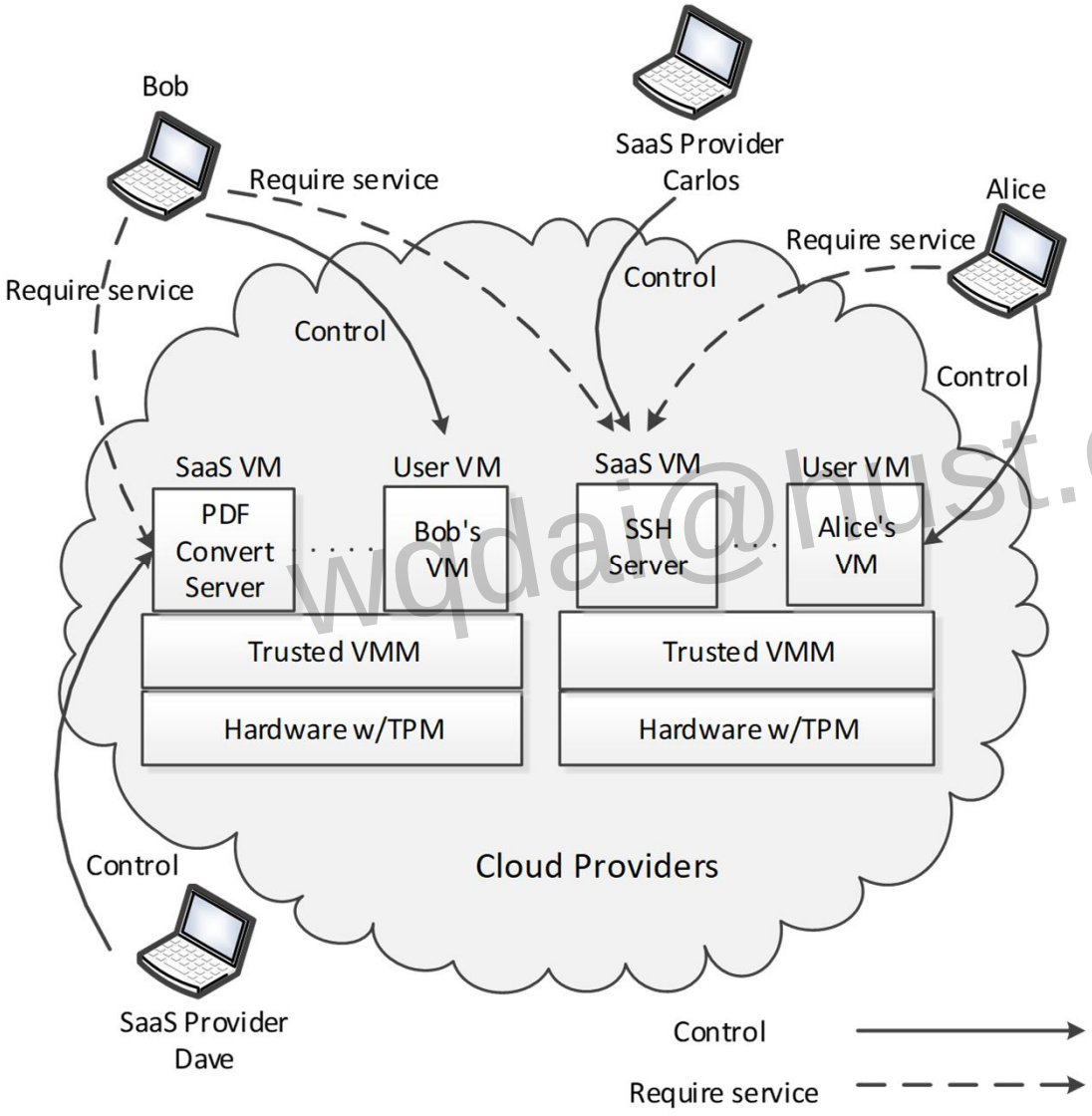


Seal Storage

VM回滚了，PCR没有回滚导致seal的密钥无法解封。

应用程序不支持回滚

云环境场景分析-重放攻击



1.另外简单的回滚vTPM同样会导致重放攻击：尽管vTPM在虚拟机回滚到不可信状态($S_{untrusted}$)也同样回滚，但如果服务商在Alice每次做远程证明的时候都回滚虚拟机到可信状态，则可以利用重放攻击消除入侵痕迹。

2.此外像pdf转换服务提供商Dave使用能免费使用10次的pdf转换软件为Bob提供服务，即使是pdf软件借助vTPM记录实际使用次数，由于Dave在回滚虚拟机的时候vTPM的数据也发生回滚保持一致性，从而导致服务商可以无限次使用免费软件给其他云用户收费服务。

虚拟机安全回滚机制

- 解决思路：
  - 首先为了保证虚拟机安全状态一致性，需要实现在对虚拟机做快照时，也对vTPM的状态做了快照，保证可信执行环境的状态与vTPM的状态一一对应。
  - 为了保证VM在虚拟机在做过一次快照后新增的密钥等安全数据不在回滚过程中发生丢失现象，需要在vTPM回滚后恢复密钥数据等等。
  - 为了解决回滚虚拟机带来的重放攻击问题，在每次创建快照（回滚）时会把快照时间（回滚时间）、执行创建命令的用户信息（执行回滚命令的用户信息）、vTPM当前状态（vTPM回滚前的状态）以及虚拟机当前状态（虚拟机回滚前状态）等信息的度量值分别扩展到两组新增的PCRs中去，从而保证了快照的唯一性，用永远不会发生回滚操作的PCRs来真实地反映了虚拟机的回滚轨迹，使得攻击者无法通过回滚虚拟机来消除攻击痕迹。

虚拟机安全回滚机制

- 解决思路：

$$PCR_{24}^{n} = SHA\text{-}1(PCR_{24}^{n-1} | SHA\text{-}1(snap\_time))$$

$$PCR_{28}^{n} = SHA\text{-}1(PCR_{28}^{n-1} | SHA\text{-}1(roll\_time))$$

$$PCR_{25}^{n} = SHA\text{-}1(PCR_{25}^{n-1} | SHA\text{-}1(user\_UID))$$

$$PCR_{29}^{n} = SHA\text{-}1(PCR_{29}^{n-1} | SHA\text{-}1(user\_UID))$$

$$PCR_{26}^{n} = SHA\text{-}1(PCR_{26}^{n-1} | SHA\text{-}1(vTPM\_snap))$$

$$PCR_{30}^{n} = SHA\text{-}1(PCR_{30}^{n-1} | SHA\text{-}1(vTPM\_snap))$$

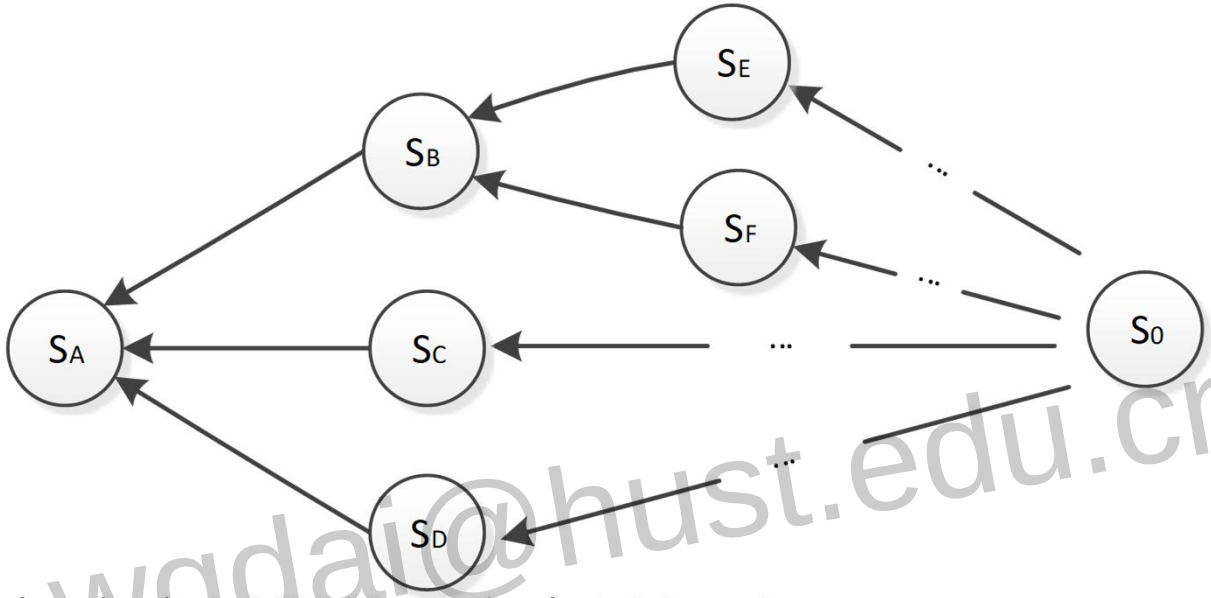
$$PCR_{27}^{n} = SHA\text{-}1(PCR_{27}^{n-1} | SHA\text{-}1(app\_event))$$

$$PCR_{28\sim31}^{n} \neq 0$$

Table 2: New added PCRs

PCRs	Extend info	Can be reset or not	Can be extend or not
PCR_{24}	System time shows when take the snapshot	Can, by rvTPM in rvTPM	Can, by rvTPM in rvTPM
PCR_{25}	User UID shows who make the snapshot	Can, by rvTPM in rvTPM	Can, by rvTPM in rvTPM
PCR_{26}	Measurement of the vTPM snapshot	Can, by rvTPM in rvTPM	Can, by rvTPM in rvTPM
PCR_{27}	Optional, for applications use	Can, by rvTPM in rvTPM	Can be extend by applications
PCR_{28}	User UID shows who rollback the VM	Cannot	Can, by rvTPM in rvTPM
PCR_{29}	System time shows when VM rollback	Cannot	Can, by rvTPM in rvTPM
PCR_{30}	Measurement of the vTPM last snapshot	Cannot	Can, by rvTPM in rvTPM
PCR_{31}	Optional, for applications use	Cannot	Can be extend by applications

虚拟机生命周期



虚拟机生命周期的状态变迁记录：

由于记录回滚事件的PCR28~31是属于永远都不会被重置或者回滚的PCRs，因此同样是虚拟机处于S_A状态，无论是经过S_E到S_B再到S_A还是S_C到S_A，验证者都能通过这几个PCR值，发现虚拟机是否曾经被攻击过（在S_E状态时被窃取了隐私）或者被偷偷缩减计算资源（在S_F状态时内存被减少了一半）或者是软件试用到期（在S_C状态时软件已经达到试用次数）等等。

虚拟机生命周期

$PCR^0_{24\sim31} = 0$

After the user take the snapshot S_1(snap_time, user_UID, vTPM_snap), then the

$$\begin{aligned} PCR^1_{24\sim26} &= SHA\text{-}1(PCR^0_{24\sim26}|SHA\text{-}1(S_1)) \\ &= SHA\text{-}1(0|SHA\text{-}1(S_1)) \end{aligned}$$

Then every time when the VM take the snapshot S_n (snap_time, user_UID, vTPM_snap, n>0), the rvTPM will first reset the $PCR_{24\sim27}$,

$$PCR^{n-1}_{24\sim27} = 0$$

then extend them with the information of the snapshot S_n:

$$\begin{aligned} PCR^n_{24\sim26} &= SHA\text{-}1(PCR^{n-1}_{24\sim26}|SHA\text{-}1(S_n)) \\ &= SHA\text{-}1(0|SHA\text{-}1(S_n)) \end{aligned}$$

When the first time the VM rollback from the state S_B to snapshot S_A: Our rvTPM will restore the vTPM including the newly added $PCR_{24\sim27}$, then the remote can get the snapshot information from these PCRs. So the

$$PCR^A_{24\sim26} = SHA\text{-}1(0|SHA\text{-}1(S_A))$$

The second group of PCRs need to record the rollback event, it will be extended with the rollback information, so the

$$\begin{aligned} PCR^1_{28\sim30} &= SHA\text{-}1(PCR^0_{28\sim30}|SHA\text{-}1(S_B)) \\ &= SHA\text{-}1(0|SHA\text{-}1(S_B)) \end{aligned}$$

$PCR_{28\sim30}$ stand for the birth state to the last state before the rollback event and the $PCR_{24\sim26}$ stand for the state revert to. So here the remote verifier can find the VM is from new one to state S_B then rollback to state S_A.

After then, when the VM running to another state S_C and then roll back to snapshot S_A: Our rvTPM will restore the vTPM including the newly added $PCR_{24\sim26}$.

$$PCR^A_{24\sim26} = SHA\text{-}1(0|SHA\text{-}1(S_A))$$

And then extend the PCR_{28}-PCR_{30}

$$\begin{aligned} PCR^2_{28\sim30} &= SHA\text{-}1(PCR^1_{28\sim30}|SHA\text{-}1(S_C)) \\ &= SHA\text{-}1(SHA\text{-}1(0|SHA\text{-}1(S_B))|SHA\text{-}1(S_C)) \end{aligned}$$

应用程序支持回滚

应用虽然可以通过$PCR_{24\sim30}$得知，虚拟机是否发生了回滚，但应用程序无法确定回滚操作是否影响到它的安全执行。比如试用版pdf转换软件的使用次数并不会影响虚拟机的状态、云用户也不需要知道。

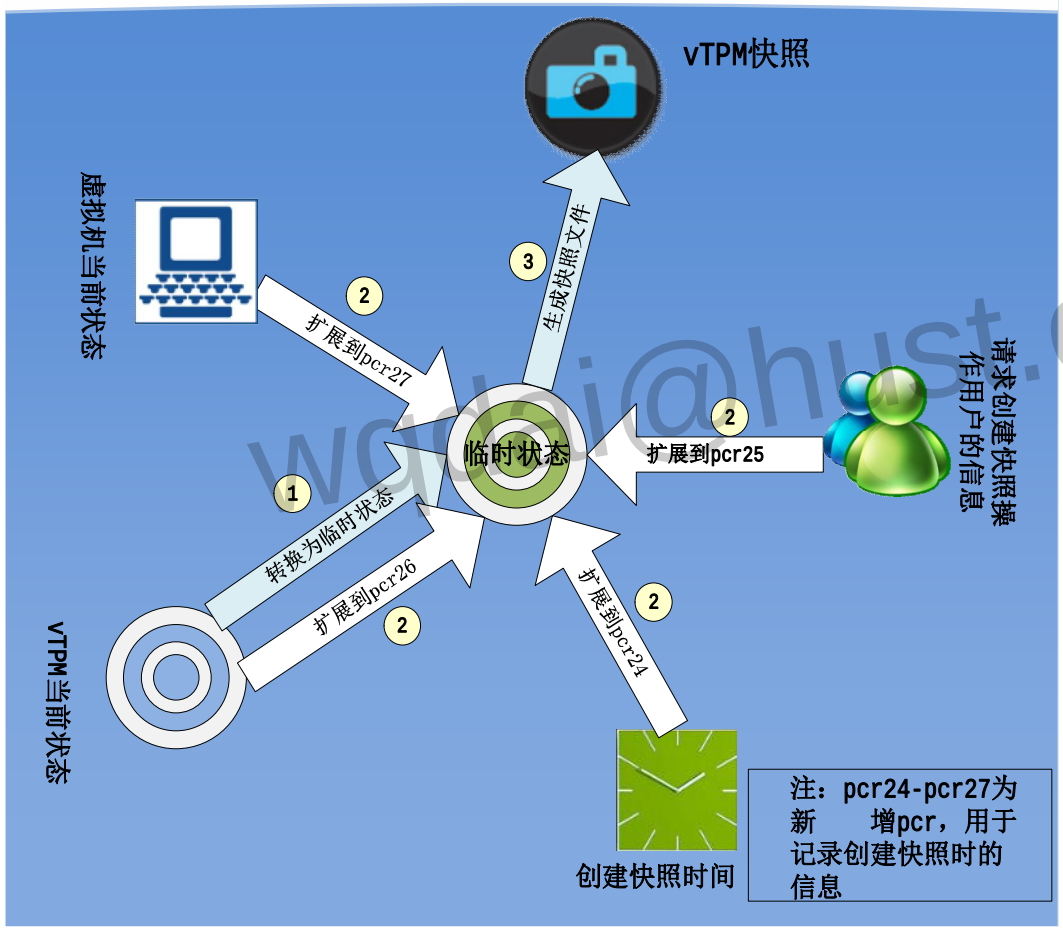
新增PCR31：某些events软件不允许回滚的时候，就扩展到PCR31

$$
\begin{aligned}
PCR_{31}^{n} &= SHA\text{-}1(PCR_{31}^{n-1}|SHA\text{-}1(E_n)) \\
&= SHA\text{-}1(SHA\text{-}1(PCR_{31}^{n-2}|SHA\text{-}1(E_{n-1}))|SHA\text{-}1(E_n)) \\
&= \cdots \\
&= SHA\text{-}1(SHA\text{-}1(PCR_{31}^{0}|\cdots|SHA\text{-}1(E_{n-1}))|SHA\text{-}1(E_n))
\end{aligned}
$$

通过简单的seal和unseal就应用就可以知道其生命周期状态变化了。因为应用一般并不需要知道虚拟机的状态变化，所以这个PCR31在回滚时候并不需要扩展。

虚拟机创建快照

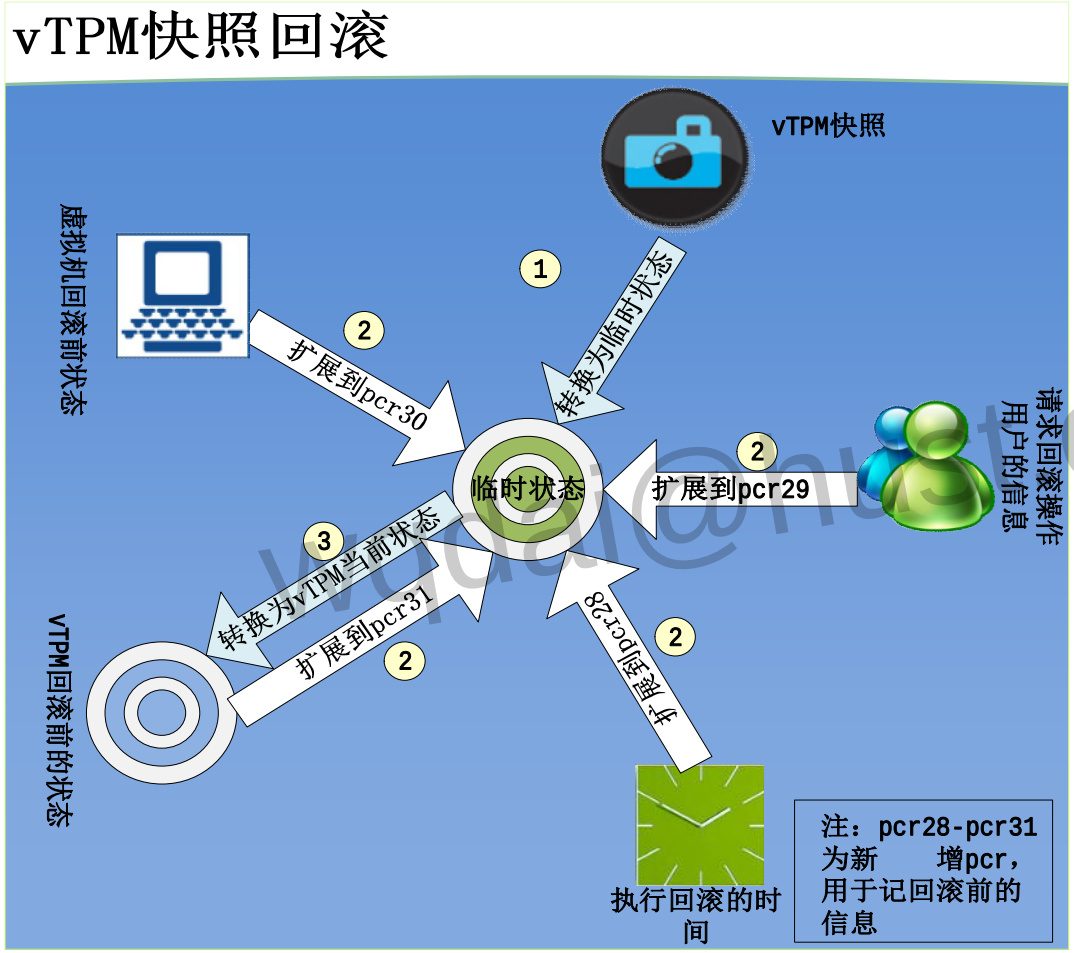
创建vTPM快照时



创建vTPM快照：

1. 将vTPM的当前状态转换为临时状态
2. 把生成快照的时间、请求创建快照操作用户信息、vTPM当前状态以及虚拟机的当前状态扩展到pcr24-pcr27中
3. 最后，把临时状态转换为vTPM快照文件。

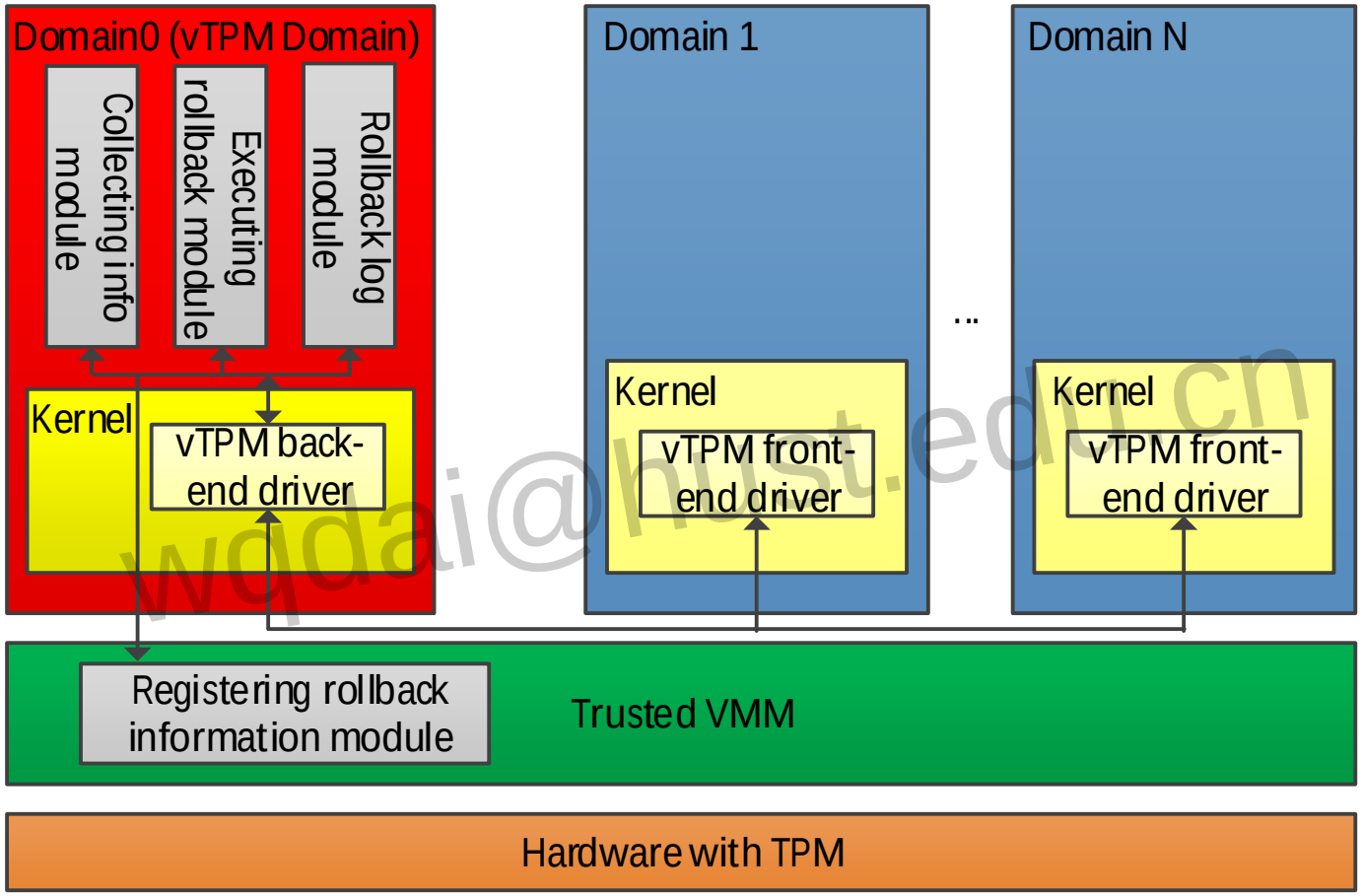
虚拟机安全回滚



vTPM快照回滚：

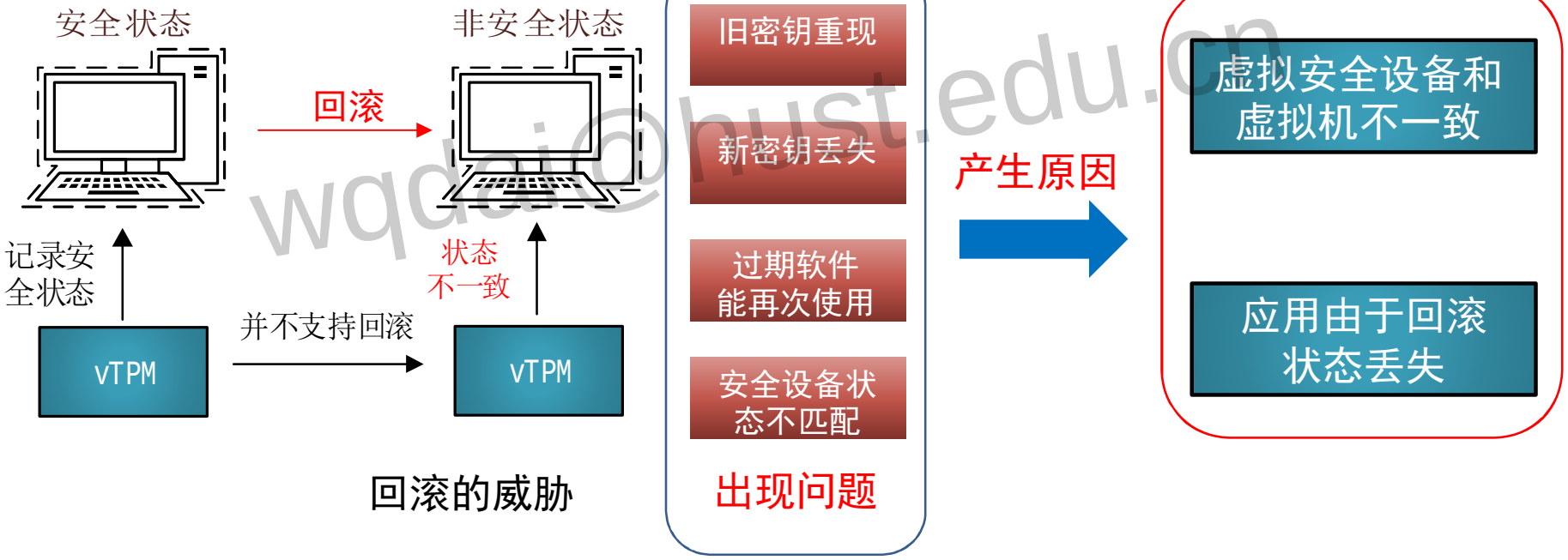
1. 将vTPM的当前状态转换为临时状态
2. 把回滚时间、执行回滚命令的用户信息、虚拟机回滚前的状态以及vTPM回滚前的状态扩展到pcr28-pcr31中
3. 最后，把临时状态转换为vTPM快照文件

解决方案（虚拟机安全回滚机制续）

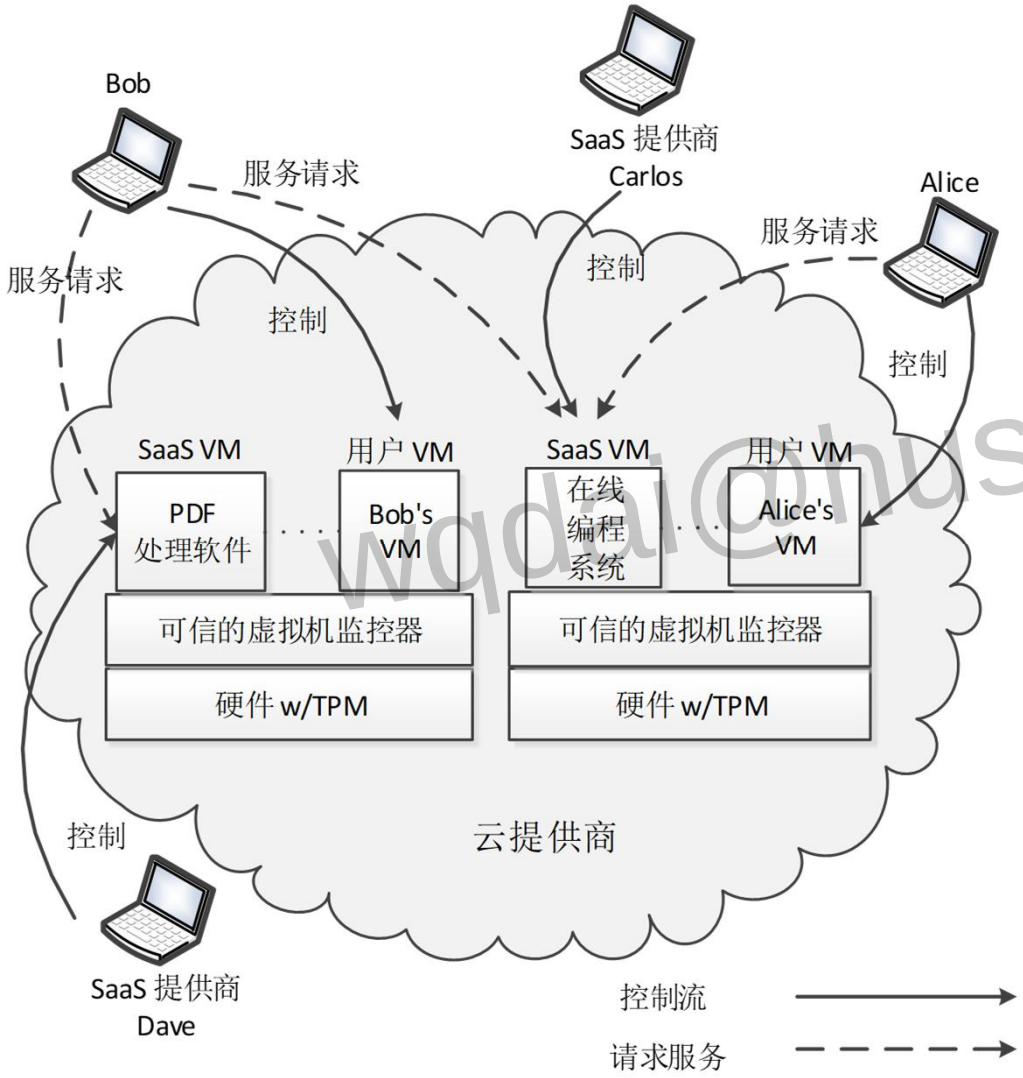


回滚造成vTPM与虚拟机状态不一致

- 虚拟机回滚为云提供了便利，同时也为云环境带来了挑战。



恶意回滚破坏软件生命周期



场景1：Alice和Bob在连接在线编程系统前后后需要进行可信验证。云服务提供商在两次可信验证之间回滚到恶意的运行状态，窃取用户的信息，在进行第二次验证之前将状态恢复。Alice和Bob无法从可信验证中获得虚拟机恶意回滚的信息

场景2：云服务商Dave购买了PDF处理软件，向云用户提供PDF处理函数并且收费。恶意的云服务商Dave利用恶意回滚，保存PDF处理软件的快照，在软件到期之后无限的使用收费软件快照，提供给Bob等用户获得盈利。

解决vTPM由回滚带来的状态不一致

针对vTPM的数据进行分析，采取不同的策略解决数据不一致问题