Provisioning secrets with cf_utility from a deployment machine to deployed VM's

Scenario 1

Paul has a deployment machine at home and one or more deployed VM's running in the cloud under SEV. He wants to provide some material to the deployed machines in a confidential, integrity protected, authenticated manner. The deployed VM's were constructed by him on a deployment machine.

Procedure

1. Paul uses CF utilities (cert-utility) to generate a public-private key pair, $Policy - Key_{public}, Policy - Key_{secret}$. He places the self-signed certificate, $Cert_{policy}$, (also generated by *cert-utility*) naming $Policy - Key_{public}$, in the VM image he constructs (so it is part of the SEV measurement) along with the *cf-utility* program and a new program described below call *key-client*. $Policy - Key_{public}$ should be accessible to programs on the VM. He also places the IP address of his deployment machine and two port addresses in the image. The first port will be the "*simpleserver*" port second port will be the "*key-server*" port on his deployment machine. He arranges for the deployed VM, when it first starts, to use *cf-utility* to contact his deployment machine on the *simpleserver* port to get certified and then have *key-client* on the VM contact his deployment machine on the *key-server* port (with the names of the keys he wishes to provision as generated below) to get the confidential material (probably a key). He instructs *key-client* to store this key securely on the VM, for example, by putting it in *cryptstore* (see below). In practice, the url of the deployment machine and the two ports will be specified in command line parameters in a script the VM runs when it first starts on an SEV machine in the cloud; in practice, the name of the keys are arguments *to key-client* on the deployed machine.
2. Paul "measures," using something like virtee, his constructed VM as well as a measurement on his deployment machine[1] (possibly using the simulated-enclave or SEV). and goes through the usual procedure, using *prepare-test.sh* in *vm_model_tools/examples/scenario1*, to construct a policy, for *simpleserver*, specifying these trusted measurements along with the other customary policy stuff. The complete

[1] See below for a slightly different way to do this.

policy will be in *vm_model_tools/examples/scenario1/server* on his machine and will include $Policy - Key_{secret}$ in that material. All this is scripted now.

3. Paul runs *simpleserver* on his deployment machine pointing to the policy. *key-server* incorporates the Certifier framework. Paul runs *key-server,* which contacts *simpleserver* on the same deployment machine to get certified using a private key, *key-server,* generates. He stores the private key, $Key - server_{private}$ and the Admissions certificate, $Cert_{keysever}$ in a safe location (e.g.-*cryptstore*) accessible to programs on his deployment machine. Incidentally, $Cert_{key-sever}$ names $Key - server_{public}$ and the "measurement" of *key-server* and is signed by $Policy - Key_{secret}$. Paul does the same with key-client on this deployment machine obtaining $Cert_{key-client}$ names $Key - client_{public}$.
4. Paul's deployment machine creates one or more keys for encrypting VHDs along with their key names. Paul uses *key-client* (on his deployment machine) to instruct *key-server* (on his deployment machine) to create a policy-protected encrypted version of these keys. The keys and key names are arguments to *key-client*.[2]
5. When a deployed machine starts, it uses *cf-utility* to generate the deployed VM's generated public/private authentication key, $key_{deployed-VM,public}$, $key_{deployed-VM,private}$, as well as the Admissions certificate, $Cert_{key_{deployed-VM,public}}$, obtained from *simpleserver*, and stores them securely on the deployed VM (say in *cryptstore*).
6. Next, Paul invokes *key-client*, on the deployed machine providing the resource names used above (These are arguments to *key-client*). *key-server* on the deployment machine returns the secret key data registered on the deployment machine in step 4. This works because the deployment machine *key-client* has already provisioned the keys to the deployment machine *key-server*, so when *key-client* on the deployed machine asks for them, *key-server* on the deployment machine can retrieve them. *Key-client* on the deployed machine has access to the deployed VM's private key, and corresponding certificate from step 5; it opens a secure channel (using *secure-authenticated-channel* in the certifier) between *key-client* (on the deployed machine) and *key-server* (on the deployment machine) requesting the desired secret material, supplying the key name named in step 4 (see the *key-server* interface below) and stores the key securely, again, say in *cryptstore*. The secure channel is established using $Cert_{key-sever}$, $Key - server_{private}$ and $Cert_{policy}$ on the *key-server* end and $Cert_{key_{deployed-VM,public}}$, $Key - client_{private}$ and $Cert_{policy}$ all provisioned above. The material is sent to the secure channel in plaintext but is encrypted and integrity protected as it's sent over the

[2] Step 4 can be omitted if you use cf_utility, on the deployment machine, to generate keys.

channel. *Key-server* has all the information it needs to grant the access request based on the exchanged certificates although it may do something fancier later (see below).

Paul has accomplished his goal[3].

Scenario 2 (and variations)

Here are some variations on Scenario 1.

Suppose Paul doesn't want to use the deployment machine to provide the services named above. Paul simply uses the same mechanism to provision one or more SEV protected cloud VMs with keys and policy allowing it to provide the same services Paul's deployment machine in Scenario 1 using the very same software.

Paul need not use *simpleserver* to provide a signed certificate to *key-server*, he can simply sign certificates directly using the policy key (since he has the private policy key), but this involves writing a little more software. The above mechanism does NOT require additional software.

More sophisticated versions of *key-client/key-server* can impose additional authentication (say by using *acl-lib*) to provide more granular key distribution.

Work to do

Paul writes no new software and provisions no data (except the policy cert, url address and port numbers mentioned above) in his deployed VMs.

John will write *key-server*, *key-client* and also make sure *cryptstore* is suitable for any of the secure storage described above on the deployed machine. John thinks it will take about a week and that key-servers are probably generally useful.

Paul will tell John the format of the keys he generates, so John can get *key-client* on the deployment machine to retrieve them (maybe Paul wants John to change them from Certifier framework serialized keys on the deployed machine to the same format ---TBD, if so, John will provide the conversion routing in *key-client* to do so).

Key-server Interface

[3] Keys can be retrieved on a machine by using the cf_utility get-item call. You may also want a programmatic interface to these keys.

The arguments to *key-server* are provided in a protobuf. The client supplies:

1. A domain relative resource name (e.g., datica-test/storage-disk key).
2. An action: "store" or "retrieve".
3. Data (For example, a serialized key, if the action is "store").
4. Credentials (a serialized certificate chain) [optional]

Return values are:

1. Return status ("success", "fail", "auth-challenge")
2. Data (For example, the serialized key return if the action is "retrieve" or a nonce if authentication is required)

In the first implementation, all domain members can add or retrieve any key. However, future implementation will add granular access control using *acl-lib*. However, in no case, should *key-server* on the deployment machine should NEVER provide deployment machine secrets (like the deployment machine's private authentication key to any *key-client*.