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

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 (e.g., keys) 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

[2] Step 4 can be omitted if you use cf-utility, on the deployment machine, to generate keys. Even if you don't use cf-utility to generate the keys, you can use the cf-utility (put-item function) to put keys in cryptstore without using *key-client* to provision *key-server* provided your provisioning a *key-server* on the same machine.

secure channel in plaintext but is encrypted and integrity protected as it's sent over the 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.

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

This functionality is now part of the Certifier Framework and is described below.

[3] Keys can be retrieved from cryptstore on a machine by using the cf-utility get-item call. You can use a programmatic interface to cryptstore as well.

Using *key-client* and *key-server* to provision keys

key_client and *key-server,* in the `$CERTIFIER_ROOT/vm_model_tools/src` directory, are programs which use an existing cryptstore initialized by *cf-utility.exe*, for example, one that results from the following calls in `$CERTIFIER_ROOT/vm_model_tools/examples/scenario1`:

```
./prepare-test.sh fresh dom0
./prepare-test.sh all dom0
./run-test.sh fresh dom0
./run-test.sh run se dom0
```

This sequence of calls will certify the VM and result in the files `policy_store.dom0` and `cryptstore.dom0` as well as support files in the subdirectory `cf_data`. *key_client* and *key-server* will not run unless the domain has been initialized and certified, as above. *key-server* stores saved values in `cryptstore.dom0` on the machine it runs on. `cryptstore.dom0` is encrypted using a key sealed using *Seal* in the enclave. Items in *cryptstore* can be exportable or not. *key-server* will not return items which are not exportable (like the private authorization key for the domain).

key-client makes a request over a secure channel to *key-server* on the same or different machine. It can request either that *key-server* store a new secret in `cryptstore.dom0` which *key-client* transmits in a request or that *key-server* retrieves a secret in `cryptstore.dom0` on the *key-server* machine. Here is an annotated set of argument (there are more default arguments in each program.

```
$CERTIFIER_ROOT/vm_model_tools/src/cf_key_server.exe
  --policy_domain_name=dom0
      This specifies the domain name.
  --encrypted_cryptstore_filename=cryptstore.dom0
      This is the location of the cryptstore relative to data_dir specified below.
  --enclave_type=simulated-enclave
      This is the enclave type.
  --policy_store_filename=policy_store.dom0
      This is the location of the policy store relative to data_dir specified below.
  --policy_key_cert_file=policy_cert_file.dom0
      This is the location of the policy cert relative to data_dir/cf_data.
  --data_dir=./
```

Key-client has some additional arguments annotated below.

```
$CERTIFIER_ROOT/vm_model_tools/src/cf_key_client.exe
  --policy_domain_name=dom0
  --encrypted_cryptstore_filename=cryptstore.dom0
  --enclave_type=simulated-enclave
  --policy_store_filename=policy_store.dom0
  --policy_key_cert_file=policy_cert_file.dom0
  --input_format=raw, --input_file=client.in
```

For a store request, the secret to be stored is in the file specified by the argument to --input_file. If the format is "raw" it is just a binary blob; if the format is "cryptstore-entry", must contain a serialized cryptstore-entry. The input file name is not relative to data_dir.

```
  --output_format=raw, --output_file=client.out
```

For a retrieve request, the retrieved secret is written to the file whose name is specified --- output_file. If the format is "raw" it is just a binary blob; if the format is "cryptstore-entry", it is a serialized cryptstore-entry. The output file name is not relative to data_dir.

```
  --action=store
```

This is the action to be performed; it is either "store" or "retrieve."

```
  --key_server_url=localhost
```

This is the URL of the *key-server key-client* wishes to access.

```
  --key_server_port=8120
```

This is the port of the *key-server key-client* wishes to access.

```
  --data_dir=./
  --resource_name=key-client-test-key
```

This is the "tag" in cryptstore associated with the item to be stored or received.

```
  --version=0
```

This is the key version of the secret, if 0, the version is the latest on retrieval or the latest version + 1 for storage.

Here is a sequence of three calls consisting of starting *key_server*, having *key_client* ask to store a new key and then using *key-client* to retrieve the key. *key-server* runs as a tcp service.

```
$CERTIFIER_ROOT/vm_model_tools/src/cf_key_server.exe \
   --policy_domain_name=dom0 \
   --encrypted_cryptstore_filename=cryptstore.dom0 \
   --enclave_type=simulated-enclave \
   --policy_store_filename=policy_store.dom0 \
   --policy_key_cert_file=policy_cert_file.dom0 \
   --data_dir=./ &

$CERTIFIER_ROOT/vm_model_tools/src/cf_key_client.exe \
   --policy_domain_name=dom0 \
    --encrypted_cryptstore_filename=cryptstore.dom0 \
    --enclave_type=simulated-enclave \
    --policy_store_filename=policy_store.dom0 \
    --policy_key_cert_file=policy_cert_file.dom0 \
    --data_dir=./ \
    --resource_name=key-client-test-key \
```

```
    --version=0 \
    --input_format=raw \
   --output_format=raw \
   --input_file=client.in \
   --output_file=client.out \
   --action=store

$CERTIFIER_ROOT/vm_model_tools/src/cf_key_client.exe \
    --policy_domain_name=dom0 \
    --encrypted_cryptstore_filename=cryptstore.dom0 \
    --enclave_type=simulated-enclave \
    --policy_store_filename=policy_store.dom0 \
    --policy_key_cert_file=policy_cert_file.dom0 \
    --data_dir=./ \
    --resource_name=key-client-test-key \
    --version=0 \
    --input_format=raw \
    --output_format=raw \
    --input_file=client.in \
    --output_file=client.out \
    --action=retrieve
```

Testing

Once performing the initialization above with cf-utility.exe, described above, you can run a test script by typing

```
./test_script.sh
```

In the `$CERTIFIER_ROOT/vm_model_tools/examples/scenario1` directory.