What is RPC?

- Remote procedure call (RPC)*
  - procedure calls executed in a different address space.
  - usually in a form of client/server interaction to invoke calls.
  - can be between any two entities with different processes and have different address space.

- Example**,

**https://www.geeksforgeeks.org/remote-procedure-call-rpc-in-operating-system/
Why RPC style tests in TF-M?

+ Current test framework is built as part of NS binary.
  - Inflexible, as tests increase, the memory requirement increases.
  - Because of low footprint platform boards, we must run different binaries running subset of tests at a time.
    + E.g. some Musca boards can't afford the whole PSA ACK test suite in one go.

+ Running more complex test frameworks targeted to PSA APIs is not feasible due to the limitation of the environment where the test runs.
  - E.g. mbed TLS based PSA regression

+ Current test output is provided as text logs along the UART channel.
  - Making it difficult to parse on the host system to understand failure.
Why RPC style tests in TF-M?

+ Therefore, we require a solution which can interrogate with the board programmatically.

+ RPC tests makes it easier to scale test cases.
  - since the entire test framework can run within host.
  - The size of the framework of the tests running on the NS world remains constant over time.
  - Could help in enabling all the features and tests on target by default.

+ This allows to have rich test environments and increases flexibility and add more options for automation integration.
  - making it easier to understand failures.

+ Cons:
  - This solution makes it more difficult to simulate threads on the NS environment.
  - although the NS tests should focus on API validity rather than verifying more of the NS-ID identification capability of TF-M.
Proposed Framework

Our current framework looks like,
Proposed Framework

New proposed test framework

HOST

ERPC-TESTS

Test Suites

eRPC Client

SERIAL

TARGET

TF-M-TESTS

NS-Client Application

Light-weight eRPC Server

NSE

SE

TF-M
Proposed Framework

In the proposed test framework,

- On the target,
  - There is a ns client application running RPC test framework.
  - Includes a lightweight server handler which receives and handles service calls.
  - Server session runs indefinitely (Until requested to stop)
- On the host,
  - Contains all testsuites without target limitations.
  - Has RPC client making service request and receiving processed data.
  - Tests can be built separately to the target binaries.
  - Can run multiple times for long as server session is active.
- The communication is done over Serial-UART channel.
RPC interface

- In this work, we have used eRPC* framework for the RPC interface
  - Lightweight
  - Easy to integrate for our use case.
    - Supports abstraction over CMSIS-UART drivers which we use in our platforms
  - Helps with serializing and de-serializing data into byte-streams
  - Transports them via common communication channels (serial-UART for our use case)
  - At each end this data is interpreted into a function call and corresponding arguments

- Memory footprints is very low.
- Licensing: Unrestrictive BSD 3-clause

*https://github.com/EmbeddedRPC/erpc
Proposed Software Model
Proposed Software Model

Host Program
Proposed Software Model

Host Program

- Main host side program.
- Handles client rpc init and deinit.
- Calls tests/secure_fw/non_secure_suites.c
Proposed Software Model

Host Program

- The prototype of the testsuite functions are same.
  - Based on IPC or Library mode, the corresponding interface is used.
• The incoming service calls are handled by shim functions.
  • Every TF-M service api has an id which is used to identify the function or the type of interface call used.
• It calls rpc_host_handler to package these data along with invec-outvec parameters.
Proposed Software Model

Host Program

- Packages parameters (invecs, outvecs) and properties of the call, and other data into *rpc packet*.
- This package is sent to eRPC to transmit to the target.
Proposed Software Model

Host Program

- **Main applications**
  - calls ns_secure_testsuite()

- **Tests**
  - psa_calls/tfm_ns_dispatcher

- **Shim TF-M services**
  - rpc_ns_interface()

- **RPC Host Handler**
  - rpc_invoke_handler()

Target Program

- **RPC COM**
  - TF-M services
  - rsa_calls/tfm_ns_dispatcher

  - RPC Target Handler
  - rpc_invoke_handler()

- **Receives eRPC data.**
- **Un-packages invecs, outvecs, types of call, and other data from rpc package.**
- **Based on the type of call, TF-M services are called.**
Proposed Software Model

'RPC Sequence' in this work is defined as a set of,

- `tfm_rpc_invoke_handler` handles tfm services and calls and returns the status of this event. A `tfm_rpc_packet` is sent to the server which includes all the data necessary to handle a remote tfm service call.
- `tfm_rpc_get_packet` fetches the data after a handler invocation. The processed data is sent if there was no error with the previous service handling.
Executing tests

- We have evaluated the framework by running tests for TEST_NS_ATTESTATION, TEST_NS_AUDIT, TEST_NS_CRYPTO, TEST_NS_ITS, TEST_NS_PS*, TEST_NS_PLATFORM.
  - They run and pass as expected.

- We can build the binaries by setting the macro “-DTEST_RPC_API=ON” on our existing buildsystem.
  - Currently, host is Linux system.

- Execute following command to run the host program,
  
  `<cmake_build_folder>/host_rpc/tfm_rpc_host -p <target portname> -e`

*To get around the limitation of multiple threads for the Protected Storage test suites, we have stubbed those functions since we don’t need them currently.*
The memory footprint of target (for tfm_ns binary) is given as follows,

<table>
<thead>
<tr>
<th></th>
<th>Lib Model (in B)</th>
<th>IPC Model (in B)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FLASH</td>
<td>RAM</td>
</tr>
<tr>
<td>No Tests</td>
<td>14088</td>
<td>13984</td>
</tr>
<tr>
<td>With NS Tests</td>
<td>129116</td>
<td>25152</td>
</tr>
<tr>
<td>With RPC_NS tests*</td>
<td>22280</td>
<td>14240</td>
</tr>
</tbody>
</table>

The advantage of this framework is that RPC_NS test figure is going to stay the same irrespective to the complexity and the number of test cases on the host-side.

*Enabled TEST_NS_ATTESTATION, TEST_NS_AUDIT, TEST_NS_CRYPTO, TEST_NS_ITS, TEST_NS_PS, TEST_NS_PLATFORM.
Demo 1
Use case: Python Wrapper prototype

- Using RPC framework, we can interrogate with the board in real-time.
  - Helps understanding failures easily.

- To evaluate this functionality we have used CFFI as our backend to link with rpc_host shared library.
  - Easy to integrate for our current use.
  - No additional learning of wrapper languages or maintenance.
  - Compatible with Python 2 and 3.
Usecase: Python Wrapper prototype

Preparing host client using following code.

```python
from tfmrpc import crypto, rpc

_rpc = rpc.rpc('tfmrpc/wrapper_defs/rpc.h', '../libtfm_rpc_host.so')
_crypto = crypto.crypto('tfmrpc/wrapper_defs/crypto.h', '../libtfm_rpc_host.so')

portname = _rpc.new('char[]', '/dev/ttyACM0')
_rpc.tfm_rpc_host_init(portname)
```
Use case: Python Wrapper prototype

- Defining variables

```python
_attro = _crypto.psa_key_attributes_t.new( \
    _type = 9216, \
    _bits = 0, \
    _lifetime = 0, \
    _id = 0, \
    _usage = 1, \
    _alg = 0)

_data = _crypto.new('char[]', 'This is py_wrapper test')
_data_length = 24
_key = _crypto.new('psa_key_id_t *)
```

- An example to call a tf-m service from host is given below:

```python
_crypto.psa_import_key(_attr, _data, _data_length, _key)
```
Demo 2
References

+ https://github.com/EmbeddedRPC/erpc/wiki
+ https://embeddedrpc.github.io/
+ https://cffi.readthedocs.io/