## REMOTE ATTESTATION EXAMPLE

Remote Attestation code example is available here: https://github.com/sangfansh (https://github.com/sangfansh)

Original unmodified version is available here: https://github.com/svartkanin/linux-sgx-remoteattestation (https://github.com/svartkanin/linux-sgx-remoteattestation)

IAS Service Guide is available here: https://software.intel.com/sites/default/files/managed/7e/3b/ias-api-spec.pdf (https://software.intel.com/sites/default/files/managed/7e/3b/ias-api-spec.pdf)

Before running the code, some settings have to be set in the GeneralSettings.h file:

- The application port and IP
- A server certificate and private key are required for the SSL communication between the SP and the Application (which can be self-signed)
  *e.g. openssl req -x509 -nodes -newkey rsa:4096 -keyout server.key -out server.crt -days 365*
- The SPID provided by Intel when registering for the developer account (https://software.intel.com/en-us/form/sgx-onboarding)
- The certificate sent to Intel when registering for the developer account (https://software.intel.com/en-us/form/sgx-onboarding)
- IAS Rest API url (should stay the same)

To be able to run the above code some external libraries are needed:

- Google Protocol Buffers (should already be installed with the SGX SDK package) otherwise install *libprotobuf-dev*, *libprotobuf-c0-dev* and *protobuf-compiler*

All other required libraries can be installed with the following command:
*sudo apt-get install libboost-thread-dev libboost-system-dev curl libcurl4-openssl-dev libssl-dev liblog4cpp5-dev libjsoncpp-dev*

After the installation of those dependencies, the code can be compiled with the following commands:

*cd ServiceProvider*

*make*
*cd ../Application*
*make SGX_MODE=HW SGX_PRERELEASE=1*

---

The sample application has two parts: ServiceProvider and the Application. The Application needs to prove to the ServiceProvider that it is running on a claimed trusted SGX enclave so that the Service Provider can proceed to provision secret data. The messages exchanged during the remote attestation process are serialized using Google Protobuf (https://github.com/google/protobuf (https://github.com/google/protobuf)).

First let's examine the Application. The entry point of the Application is inside *isv_app.cpp* (ISV stands for Individual Software Vendor). It starts by initiating the MessageHandler that handles the messages to be exchanged during remote attestation.

```
1  #include <iostream>
2  #include <unistd.h>
3
4  #include "LogBase.h"
5
6  using namespace util;
7
8  #include "MessageHandler.h"
9
10 int Main(int argc, char* argv[]) {
11     LogBase::Inst();
12
13     int ret = 0;
14
15     MessageHandler msg;
16     msg.init();
17     msg.start();
18
19     return ret;
20 }
21
22
23 int main( int argc, char **argv ) {
24     try {
25         return Main(argc, argv);
26     } catch (std::exception& e) {
27         Log("exception: %s", e.what());
28     } catch (...) {
29         Log("unexpected exception") ;
```

```
30      }
31
32      return -1;
33  }
```

**view raw (https://gist.github.com/sangfansh/e884ede5fd80082eeedc70f22e8f2e4c/raw/e198aa9228b1cb300bc98c87e7839259cc6a102a/isv_app.cpp) isv_app.cpp (https://gist.github.com/sangfansh/e884ede5fd80082eeedc70f22e8f2e4c#file-isv_app-cpp)** hosted with ❤ by **GitHub (https://github.com)**

MessageHandler (a wrapper of all of the networking functionalities):

The MessageHandler has a protected enclave that handles all the secrets, as well as the generation and processing of cryptographic messages. We can see that MessageHandler has several message generation and handling functions. Specifically they are functions in forms of *generateMsgx()* and *handleMsgx()*. Google Protobuf is used by those functions to serialize the messages to be exchanged via network transportation.

The MessageHandler also has a NetworkManagerServer object. The NetworkManagerServer object enables the Application to act as a server to initialize connection and binds itself to a client via SSL (implementation detail in *server.h*, using object *Server* server*). It also inherits NetworkManager class in order to serialize and send messages.

```
1   #ifndef MESSAGEHANDLER_H
2   #define MESSAGEHANDLER_H
3
4   #include <string>
5   #include <stdio.h>
6   #include <limits.h>
7   #include <unistd.h>
8   #include <iostream>
9   #include <iomanip>
10
11  #include "Enclave.h"
12  #include "NetworkManagerServer.h"
13  #include "Messages.pb.h"
14  #include "UtilityFunctions.h"
15  #include "remote_attestation_result.h"
16  #include "LogBase.h"
17  #include "../GeneralSettings.h"
18
19  using namespace std;
20  using namespace util;
21
```

```
22  class MessageHandler {
23
24  public:
25      MessageHandler(int port = Settings::rh_port);
26      virtual ~MessageHandler();
27
28      sgx_ra_msg3_t* getMSG3();
29      int init();
30      void start();
31      vector<string> incomingHandler(string v, int type);
32
33  private:
34      sgx_status_t initEnclave();
35      uint32_t getExtendedEPID_GID(uint32_t *extended_epid_group_id);
36      sgx_status_t getEnclaveStatus();
37
38      void assembleAttestationMSG(Messages::AttestationMessage msg, ra_samp_response_hea
39      string handleAttestationResult(Messages::AttestationMessage msg);
40      void assembleMSG2(Messages::MessageMSG2 msg, sgx_ra_msg2_t **pp_msg2);
41      string handleMSG2(Messages::MessageMSG2 msg);
42      string handleMSG0(Messages::MessageMsg0 msg);
43      string generateMSG1();
44      string handleVerification();
45      string generateMSG0();
46      string createInitMsg(int type, string msg);
47
48  protected:
49      Enclave *enclave = NULL;
50
51  private:
52      int busy_retry_time = 4;
53      NetworkManagerServer *nm = NULL;
54
55  };
56
57  #endif
```

**view raw (https://gist.github.com/sangfansh/9e159a13dc47ed196facd14a9a2e148a/raw/a432b7128f571515dff4a7cf1317d44a1db330a0/MessageHandler.h) MessageHandler.h (https://gist.github.com/sangfansh/9e159a13dc47ed196facd14a9a2e148a#file-messagehandler-h)** hosted with ❤ by **GitHub (https://github.com)**

msg -> init():

```
1  int MessageHandler::init() {
2      this->nm->Init();
3      this->nm->connectCallbackHandler([this](string v, int type) {
```

```
        return this->incomingHandler(v, type);
    });
}


void MessageHandler::start() {
    this->nm->startService();
}
```

**view raw (https://gist.github.com/sangfansh/788651555f6e7e43ceb8dd346764da4f/raw/42841b05e281f60ee0969ee365ec6d8c5122f992/MessageHandler.cpp) MessageHandler.cpp (https://gist.github.com/sangfansh/788651555f6e7e43ceb8dd346764da4f#file-messagehandler-cpp)** hosted with ♥ by **GitHub (https://github.com)**

```
void NetworkManagerServer::Init() {
    this->server = new Server(this->io_service, this->port);
}
```

**view raw (https://gist.github.com/sangfansh/13a82a3b4e7b5c51bfe67b7bf97cd542/raw/92645e8bc0c279ce901235d56f2806934461b72f/NetworkManagerServer.h) NetworkManagerServer.h (https://gist.github.com/sangfansh/13a82a3b4e7b5c51bfe67b7bf97cd542#file-networkmanagerserver-h)** hosted with ♥ by **GitHub (https://github.com)**

```
Server::Server(boost::asio::io_service& io_service, int port) : io_service_(io_service
            boost::asio::ip::tcp::endpoint(boost::asio::ip::tcp::v4(), port)),
    context_(boost::asio::ssl::context::sslv23) {

    this->context_.set_options(boost::asio::ssl::context::default_workarounds
                               | boost::asio::ssl::context::no_sslv2
                               | boost::asio::ssl::context::single_dh_use);

    this->context_.use_certificate_chain_file(Settings::server_crt);
    this->context_.use_private_key_file(Settings::server_key, boost::asio::ssl::contex

    Log("Certificate \"" + Settings::server_crt + "\" set");
    Log("Server running on port: %d", port);
}
```

**view raw (https://gist.github.com/sangfansh/60737cb869a669cc45f1b1960347c220/raw/e869ca30b1b88a743c675cf1ef79ac5a33b3a138/Server.cpp) Server.cpp (https://gist.github.com/sangfansh/60737cb869a669cc45f1b1960347c220#file-server-cpp)** hosted with ♥ by **GitHub (https://github.com)**

When the MessageHandler msg is initialized using *init()*, the NetworkManagerServer object inside is also initialized. It causes the initialization of Server object, which sets up the SSL io_service socket and the selected port. Then a function *incomingHandler()* is connected to the NetworkManagerServer as the CallbackHandler. This function is responsible for generating all the message replies according to the type of the message that it receives.

As mentioned above, *incomingHandler(string, int)* handles all of the incoming messages and generates corresponding replies. Let's briefly examine this handler (line 395 at *MessageHandler.cpp*).

```
1  vector<string> MessageHandler::incomingHandler(string v, int type) {
2      vector<string> res;
3      string s;
4      bool ret;
5
6      switch (type) {
7      case RA_VERIFICATION: {     //Verification request
8          Messages::InitialMessage init_msg;
9          ret = init_msg.ParseFromString(v);
10         if (ret && init_msg.type() == RA_VERIFICATION) {
11             s = this->handleVerification();
12             res.push_back(to_string(RA_MSG0));
13         }
14     }
15     break;
16     case RA_MSG0: {             //Reply to MSG0
17         Messages::MessageMsg0 msg0;
18         ret = msg0.ParseFromString(v);
19         if (ret && (msg0.type() == RA_MSG0)) {
20             s = this->handleMSG0(msg0);
21             res.push_back(to_string(RA_MSG1));
22         }
23     }
24     break;
25     case RA_MSG2: {             //MSG2
26         Messages::MessageMSG2 msg2;
27         ret = msg2.ParseFromString(v);
28         if (ret && (msg2.type() == RA_MSG2)) {
29             s = this->handleMSG2(msg2);
30             res.push_back(to_string(RA_MSG3));
31         }
32     }
33     break;
34     case RA_ATT_RESULT: {       //Reply to MSG3
35         Messages::AttestationMessage att_msg;
36         ret = att_msg.ParseFromString(v);
37         if (ret && att_msg.type() == RA_ATT_RESULT) {
38             s = this->handleAttestationResult(att_msg);
39             res.push_back(to_string(RA_APP_ATT_OK));
40         }
41     }
```

```
42      break;
43      default:
44          Log("Unknown type: %d", type, log::error);
45          break;
46      }
47
48      res.push_back(s);
49
50      return res;
51  }
```

**view raw (https://gist.github.com/sangfansh/bda856ff5728ab7fff00172e6b98d456/raw/cc66ebdbd199268e2639fb84539f59c18d39ad33/MessageHandler.cpp) MessageHandler.cpp (https://gist.github.com/sangfansh/bda856ff5728ab7fff00172e6b98d456#file-messagehandler-cpp)** hosted with ❤ by **GitHub (https://github.com)**

There are four cases:

- RA_VERIFICATION
- RA_MSG0
- RA_MSG2
- RA_ATT_RESULT

Each case is one type of the messages that are exchanged in time order during remote attestation.

Upon here, the ISV application's MessageHandler has finished initialization.

msg -> start():

```
1  void MessageHandler::start() {
2      this->nm->startService();
3  }
```

**view raw (https://gist.github.com/sangfansh/ec0b603dc914421b58c4454f3e3ef126/raw/6c4f27d387bfd4f2394bcbdd336df7ade9dcd14d/MessageHandler.cpp) MessageHandler.cpp (https://gist.github.com/sangfansh/ec0b603dc914421b58c4454f3e3ef126#file-messagehandler-cpp)** hosted with ❤ by **GitHub (https://github.com)**

```
1  void NetworkManagerServer::startService() {
2      this->server->start_accept();
3      this->io_service.run();
4  }
```

**view raw (https://gist.github.com/sangfansh/4004e50e2bc242b5653fd6ea7dfad4a3/raw/59df2c846843c822810535fdbf9f5252527a2fcf/NetworkManagerServer.cpp) NetworkManagerServer.cpp (https://gist.github.com/sangfansh/4004e50e2bc242b5653fd6ea7dfad4a3#file-networkmanagerserver-cpp)** hosted with ❤ by **GitHub (https://github.com)**

Function *start()* calls NetworkManager's *startService()* function, which calls Server's

*start_accept()* to start SSL service. Upton here, the ISV has started running and is ready for any incoming traffic.

Now let's examine ServiceProvider's structure and initialization process.

*isv_app.cpp* inside ServiceProvider:

```
#include <iostream>
#include <unistd.h>

#include "LogBase.h"
#include "NetworkManager.h"
#include "VerificationManager.h"
#include "UtilityFunctions.h"

using namespace util;

int Main(int argc, char *argv[]) {
    LogBase::Inst();

    int ret = 0;

    VerificationManager *vm = VerificationManager::getInstance();
    vm->init();
    vm->start();

    return ret;
}


int main( int argc, char **argv ) {
    try {
        int ret = Main(argc, argv);
        return ret;
    } catch (std::exception & e) {
        Log("exception: %s", e.what());
    } catch (...) {
        Log("unexpected exception");
    }

    return -1;
}
```

**view raw (https://gist.github.com/sangfansh/2dbd162c0dc46c51d0f046f0fbc8b7ed/raw/fdee870d64d7d65e8c1baa7c3edf25e90d9c8511/isv_app.cpp) isv_app.cpp (https://gist.github.com/sangfansh/2dbd162c0dc46c51d0f046f0fbc8b7ed#file-isv_app-cpp)** hosted

with ♥ by **GitHub (https://github.com)**

It's the ServiceProvider Application itself. Similar to the client's application, it has its own message handler VerificationManager since it acts as the verifier in the remote attestation process.

Inside the VerificationManager, it has a NetworkManagerClient which also inherits NetworkManager and is responsible for the SSL connection as well as serializing and sending messages.

```
1  #ifndef VERIFICATIONMANAGER_H
2  #define VERIFICATIONMANAGER_H
3
4  #include <string>
5  #include <stdio.h>
6  #include <limits.h>
7  #include <unistd.h>
8
9  #include "ServiceProvider.h"
10 #include "NetworkManagerClient.h"
11 #include "LogBase.h"
12 #include "Messages.pb.h"
13 #include "WebService.h"
14
15 using namespace std;
16
17 class VerificationManager {
18
19 public:
20     static VerificationManager* getInstance();
21     virtual ~VerificationManager();
22     int init();
23     vector<string> incomingHandler(string v, int type);
24     void start();
25
26 private:
27     VerificationManager();
28     string prepareVerificationRequest();
29     string handleMSG0(Messages::MessageMsg0 m);
30     string handleMSG1(Messages::MessageMSG1 msg);
31     string handleMSG3(Messages::MessageMSG3 msg);
32     string createInitMsg(int type, string msg);
33     string handleAppAttOk();
34
35 private:
```

```
36      static VerificationManager* instance;
37      NetworkManagerClient *nm = NULL;
38      ServiceProvider *sp = NULL;
39      WebService *ws = NULL;
40  };
41
42  #endif
```

**view raw (https://gist.github.com/sangfansh/d1c1443a08f7e7128fff3203379a4231/raw/2e767f4582148cea2ed5e8f84fe595087e103d0f/VerificationManager.h) VerificationManager.h (https://gist.github.com/sangfansh/d1c1443a08f7e7128fff3203379a4231#file-verificationmanager-h)** hosted with ❤ by **GitHub (https://github.com)**

In addition, it has a WebService object to performs the verification phase with Intel Attestation Service (IAS) using the QUOTE sent from the client enclave. In one word, a ServicePrivider also acts as a wrapper of all the IAS requests and message processing, as well as any encryption key derivation, using WebService.

```
1   #ifndef WEBSERVICE_H
2   #define WEBSERVICE_H
3
4   #include <string>
5   #include <stdio.h>
6   #include <limits.h>
7   #include <unistd.h>
8   #include <stdio.h>
9   #include <curl/curl.h>
10  #include <jsoncpp/json/json.h>
11  #include <iostream>
12
13  #include "LogBase.h"
14  #include "UtilityFunctions.h"
15
16  using namespace std;
17  using namespace util;
18
19  enum IAS {
20      sigrl,
21      report
22  };
23
24  struct attestation_verification_report_t {
25      string report_id;
26      string isv_enclave_quote_status;
27      string timestamp;
28  };
```

```
29
30  struct attestation_evidence_payload_t {
31      string isv_enclave_quote;
32  };
33
34  struct ias_response_header_t {
35      int response_status;
36      int content_length;
37      std::string request_id;
38  };
39
40  struct ias_response_container_t {
41      char *p_response;
42      size_t size;
43  };
44
45  static int REQUEST_ID_MAX_LEN = 32;
46  static vector<pair<string, string>> retrieved_sigrl;
47
48  class WebService {
49
50  public:
51      static WebService* getInstance();
52      virtual ~WebService();
53      void init();
54      bool getSigRL(string gid, string *sigrl);
55      bool verifyQuote(uint8_t *quote, uint8_t *pseManifest, uint8_t *nonce, vector<pair<
56
57  private:
58      WebService();
59      bool sendToIAS(string url, IAS type, string payload,
60                     struct curl_slist *headers,
61                     ias_response_container_t *ias_response_container,
62                     ias_response_header_t *response_header);
63
64      string createJSONforIAS(uint8_t *quote, uint8_t *pseManifest, uint8_t *nonce);
65      vector<pair<string, string>> parseJSONfromIAS(string json);
66
67  private:
68      static WebService* instance;
69      CURL *curl;
70  };
71
72  #endif
```

**view raw (https://gist.github.com/sangfansh/dfa4b517e17cbc9030d0f09326b28698**

**/raw/de3ceed86c5767d8901065955b71e320be11a9a3/WebService.h) WebService.h (https://gist.github.com/sangfansh/dfa4b517e17cbc9030d0f09326b28698#file-webservice-h)** hosted with ❤ by **GitHub (https://github.com)**

vm -> init():

```
1  int VerificationManager::init() {
2      if (this->sp) {
3          delete this->sp;
4          this->sp = new ServiceProvider(this->ws);
5      }
6
7      this->nm->Init();
8      this->nm->connectCallbackHandler([this](string v, int type) {
9          return this->incomingHandler(v, type);
10     });
11 }
```

**view raw (https://gist.github.com/sangfansh/f935847f93926df81b4f4b7c1bace686/raw/cccded8ec6bc0a1a6e04dac5c4cae0409cdc885b/VerificationManager.cpp) VerificationManager.cpp (https://gist.github.com/sangfansh/f935847f93926df81b4f4b7c1bace686#file-verificationmanager-cpp)** hosted with ❤ by **GitHub (https://github.com)**

First, dynamically allocate a new ServiceProvider to ensure freshness of secret. Then the function initializes NetworkManagerClient that will connect to the Server (the Application) using SSL. Finally, a CallBackHandler similar to the one of the Application is set up. The handler itself is the function incomingHandler(string, int) inside VerificationManager that will handle incoming messages coming from the Application during remote attestation (line 132 *at VerificationManager.cpp*).

```
1  vector<string> VerificationManager::incomingHandler(string v, int type) {
2      vector<string> res;
3
4      if (!v.empty()) {
5          string s;
6          bool ret;
7
8          switch (type) {
9          case RA_MSG0: {
10             Messages::MessageMsg0 msg0;
11             ret = msg0.ParseFromString(v);
12             if (ret && (msg0.type() == RA_MSG0)) {
13                 s = this->handleMSG0(msg0);
14                 res.push_back(to_string(RA_MSG0));
15             }
16         }
17         break;
```

```
18          case RA_MSG1: {
19              Messages::MessageMSG1 msg1;
20              ret = msg1.ParseFromString(v);
21              if (ret && (msg1.type() == RA_MSG1)) {
22                  s = this->handleMSG1(msg1);
23                  res.push_back(to_string(RA_MSG2));
24              }
25          }
26          break;
27          case RA_MSG3: {
28              Messages::MessageMSG3 msg3;
29              ret = msg3.ParseFromString(v);
30              if (ret && (msg3.type() == RA_MSG3)) {
31                  s = this->handleMSG3(msg3);
32                  res.push_back(to_string(RA_ATT_RESULT));
33              }
34          }
35          break;
36          case RA_APP_ATT_OK: {
37              Messages::SecretMessage sec_msg;
38              ret = sec_msg.ParseFromString(v);
39              if (ret) {
40                  if (sec_msg.type() == RA_APP_ATT_OK) {
41                      this->handleAppAttOk();
42                  }
43              }
44          }
45          break;
46          default:
47              Log("Unknown type: %d", type, log::error);
48              break;
49          }
50
51          res.push_back(s);
52      } else {    //after handshake
53          res.push_back(to_string(RA_VERIFICATION));
54          res.push_back(this->prepareVerificationRequest());
55      }
56
57      return res;
58  }
```

**view raw (https://gist.github.com/sangfansh/eadaa5c6e6d10fe43e3a562c8efd7f4f/raw/1b4f150edd29d034b1956b68b19c30e6a62481af/VerificationManager.cpp) VerificationManager.cpp (https://gist.github.com/sangfansh/eadaa5c6e6d10fe43e3a562c8efd7f4f#file-verificationmanager-cpp)** hosted with ❤ by **GitHub (https://github.com)**

There are also four cases of handling messages:

- RA_MSG0
- RA_MSG1
- RA_MSG3
- RA_APP_ATT_OK

Notice that at the end of the handler, it initializes the message stream with a RA_VERIFICATION type string and the actual REQUEST. By doing so right after SSL handshake, the ServiceProvider can immediately start the RA process by forwarding the remote attestation request to the Application.

vm -> start():

```
void VerificationManager::start() {
    this->nm->startService();
    Log("Remote attestation done");
}
```

**view raw (https://gist.github.com/sangfansh/dbee6d345177c73d1ca2c1914aa5da3c/raw/dcd4619dd51ef43322814497f7772126cf5c8983/VerificationManager.cpp) VerificationManager.cpp (https://gist.github.com/sangfansh/dbee6d345177c73d1ca2c1914aa5da3c#file-verificationmanager-cpp)** hosted with ❤ by **GitHub (https://github.com)**

```
void NetworkManagerClient::startService() {
    this->client->startConnection();
}
```

**view raw (https://gist.github.com/sangfansh/6969fb1cf21cdcd19e496a81a909947a/raw/aab739d6f4e9c946e8097644cbbb968d41ef4349/NetworkManagerClient.cpp) NetworkManagerClient.cpp (https://gist.github.com/sangfansh/6969fb1cf21cdcd19e496a81a909947a#file-networkmanagerclient-cpp)** hosted with ❤ by **GitHub (https://github.com)**

```
void Client::startConnection() {
    Log("Start connecting...");

    boost::system::error_code ec;
    boost::asio::connect(socket_.lowest_layer(), this->endpoint_iterator, ec);

    handle_connect(ec);
}

void Client::handle_connect(const boost::system::error_code &error) {
    if (!error) {
        Log("Connection established");

        boost::system::error_code ec;
```

```
15          socket_.handshake(boost::asio::ssl::stream_base::client, ec);
16
17          handle_handshake(ec);
18      } else {
19          Log("Connect failed: %s", error.message(), log::error);
20      }
21  }
22
23  void Client::handle_handshake(const boost::system::error_code& error) {
24      if (!error) {
25          Log("Handshake successful");
26
27          auto ret = this->callback_handler("", -1);
28          send(ret);
29      } else {
30          Log("Handshake failed: %s", error.message(), log::error);
31      }
32  }
```

**view raw (https://gist.github.com/sangfansh/d1728c9fb3567734b1afd2d4a65ef457/raw/463d120f7f326990789276bc74e2eabe80caff8c/Client.cpp) Client.cpp (https://gist.github.com/sangfansh/d1728c9fb3567734b1afd2d4a65ef457#file-client-cpp)** hosted with ❤ by **GitHub (https://github.com)**

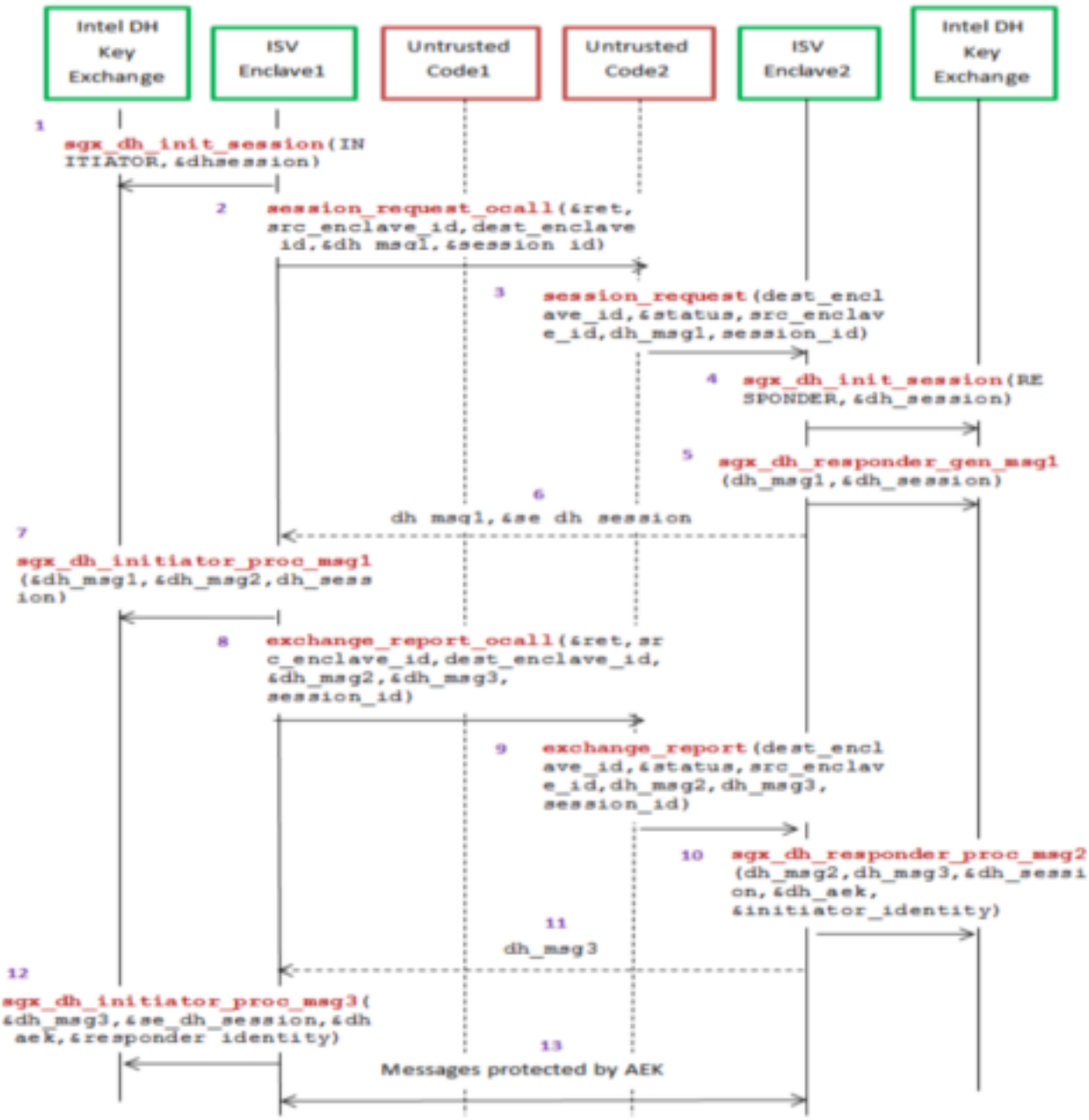
NetworkManagerClient starts the service by connecting the SSL client to server. Then the client starts the SSL handshake process with server. We can see that the *Client::handle_handshake()* function triggers the remote attestation by sending out the attestation request mentioned above if handshake is successful.

Upon here, ServiceProvider and Application are connected and remote attestation process has started.

---

Now let's examine both incomingHandler(string, int) in *MessageHandler.cpp* and *VerificationManager.cpp* to reflect the remote attestation process between SP and ISV described in the tutorial.

Summary of function calls:




(http://www.sgx101.com/wp-content/uploads/2017/09/Screen-Shot-2018-07-01-at-8.03.26-PM.png)

(Please refer to "*Messages.proto*" for message structure details.)

(Please refer to the two incomingHandler's attached below for actual implementation.)

| *MessageHandler.cpp* | *VerificationManager.cpp* |
|---|---|
| <—— | RA_VERIFICATION<br>Send InitialMessage. |

| | |
|---|---|
| Receives the requestMsg, type is RA_MSG0.<br>*GenerateMSG0()* to get Extended Group ID (GID).<br>Send MSG0. | ——> |
| | Handle MSG0.<br>Request EPID service if GID is ok.<br>Send back MSG0. |
| <—— | |
| If ServiceProvider is ok with the GID (SP_OK):<br>*initEnclave()*;<br>*createEnclave()* and *init_ra()*;<br>Trusted *sgx_ra_get_msg1()*, which gets GID and generates gx, gy for DH.<br>Send MSG1. | ——> |
| | Handles MSG1.<br>*sp_ra_proc_msg1_req()* gets SigRL form IAS.<br>Save client's public ECCDH key.<br>Generate SP public ECCDH key and save.<br>Generate shared secret.<br>Generate other keys for future communication.<br>Sign MSG2 and CMAC (sigma protocol).<br>Send MSG2. |
| <—— | |

| | |
|---|---|
| Handle MSG2.<br>Call *sgx_ra_proc_msg2()*, which generates REPORT and calls *get_quote()*.<br>*sgx_ra_proc_msg2()* will handle the key exchange in the enclave by ECalling *sgx_ra_proc_msg2_trusted_t()*.<br>Send MSG3 | ——> |
| <—— | Verify MSG3 content (ga, size, sigma MAC).<br>Verify report_data in QUOTE SHA256(ga \| gb \| vk).<br>Verify QUOTE with IAS (using *WebService::verifyQUOTE()* & *ias_verify_attestation_evidence()*).<br>Respond the client with attestation result and sign using SIGMA.<br>Send message type RA_ATT_RESULT. |
| Verify result using MAC.<br>Verify secret payload.<br>Reply RA_APP_ATT_OK. | ——> |
| | Trusted channel established. |

```
vector<string> MessageHandler::incomingHandler(string v, int type) {
    vector<string> res;
    string s;
    bool ret;

    switch (type) {
    case RA_VERIFICATION: {     //Verification request
        Messages::InitialMessage init_msg;
        ret = init_msg.ParseFromString(v);
```

```
10          if (ret && init_msg.type() == RA_VERIFICATION) {
11              s = this->handleVerification();
12              res.push_back(to_string(RA_MSG0));
13          }
14      }
15      break;
16      case RA_MSG0: {             //Reply to MSG0
17          Messages::MessageMsg0 msg0;
18          ret = msg0.ParseFromString(v);
19          if (ret && (msg0.type() == RA_MSG0)) {
20              s = this->handleMSG0(msg0);
21              res.push_back(to_string(RA_MSG1));
22          }
23      }
24      break;
25      case RA_MSG2: {             //MSG2
26          Messages::MessageMSG2 msg2;
27          ret = msg2.ParseFromString(v);
28          if (ret && (msg2.type() == RA_MSG2)) {
29              s = this->handleMSG2(msg2);
30              res.push_back(to_string(RA_MSG3));
31          }
32      }
33      break;
34      case RA_ATT_RESULT: {       //Reply to MSG3
35          Messages::AttestationMessage att_msg;
36          ret = att_msg.ParseFromString(v);
37          if (ret && att_msg.type() == RA_ATT_RESULT) {
38              s = this->handleAttestationResult(att_msg);
39              res.push_back(to_string(RA_APP_ATT_OK));
40          }
41      }
42      break;
43      default:
44          Log("Unknown type: %d", type, log::error);
45          break;
46      }
47
48      res.push_back(s);
49
50      return res;
51  }
```

**view raw (https://gist.github.com/sangfansh/bda856ff5728ab7fff00172e6b98d456/raw/cc66ebdbd199268e2639fb84539f59c18d39ad33/MessageHandler.cpp) MessageHandler.cpp (https://gist.github.com/sangfansh/bda856ff5728ab7fff00172e6b98d456#file-messagehandler-cpp)** hosted with ❤ by **GitHub (https://github.com)**

```
vector<string> VerificationManager::incomingHandler(string v, int type) {
    vector<string> res;

    if (!v.empty()) {
        string s;
        bool ret;

        switch (type) {
        case RA_MSG0: {
            Messages::MessageMsg0 msg0;
            ret = msg0.ParseFromString(v);
            if (ret && (msg0.type() == RA_MSG0)) {
                s = this->handleMSG0(msg0);
                res.push_back(to_string(RA_MSG0));
            }
        }
        break;
        case RA_MSG1: {
            Messages::MessageMSG1 msg1;
            ret = msg1.ParseFromString(v);
            if (ret && (msg1.type() == RA_MSG1)) {
                s = this->handleMSG1(msg1);
                res.push_back(to_string(RA_MSG2));
            }
        }
        break;
        case RA_MSG3: {
            Messages::MessageMSG3 msg3;
            ret = msg3.ParseFromString(v);
            if (ret && (msg3.type() == RA_MSG3)) {
                s = this->handleMSG3(msg3);
                res.push_back(to_string(RA_ATT_RESULT));
            }
        }
        break;
        case RA_APP_ATT_OK: {
            Messages::SecretMessage sec_msg;
            ret = sec_msg.ParseFromString(v);
            if (ret) {
                if (sec_msg.type() == RA_APP_ATT_OK) {
```

```
41                      this->handleAppAttOk();
42                  }
43              }
44          }
45          break;
46          default:
47              Log("Unknown type: %d", type, log::error);
48              break;
49          }
50
51          res.push_back(s);
52      } else {    //after handshake
53          res.push_back(to_string(RA_VERIFICATION));
54          res.push_back(this->prepareVerificationRequest());
55      }
56
57      return res;
58  }
```

**view raw (https://gist.github.com/sangfansh/eadaa5c6e6d10fe43e3a562c8efd7f4f/raw/1b4f150edd29d034b1956b68b19c30e6a62481af/VerificationManager.cpp) VerificationManager.cpp (https://gist.github.com/sangfansh/eadaa5c6e6d10fe43e3a562c8efd7f4f#file-verificationmanager-cpp)** hosted with ❤ by **GitHub (https://github.com)**

NEXT PROJECT › (HTTP://WWW.SGX101.COM/PORTFOLIO/SEALING_PRIMITIVES/)

‹ PREV PROJECT (HTTP://WWW.SGX101.COM/PORTFOLIO/REMOTE_ATTESTATION/)

TWITTER FACEBOOK