Local Attestation:

In file “local.pv”, the local attestation of SGX and symmetric encrypted communication are established and evaluated via “ProVerif”. First, let’s go through the syntaxes and briefly illustrate the process. Type “G” is basically defined as keys here in the model, this is partly because, in the manual of “ProVerif”, the secret key in Diffie-Hellman example illustrate at pp.43 is described as “G”. So, I borrow this from the example. Also the type “exponent” is defined because instead of taking ECDH, I use exponential encryption method as it is easy to illustrate. Function “h()” is used as “sha256” that generates MRENCLAVE. The “eReport()” is to integrate “target_info” and “report_data”, which include a Diffie-Hellman public key. “mac()” is described as “AES128-CMAC” requiring EGETKEY as generation key. However, it is a bit impractical, as in real case, the attesting enclave does not call for a EGETKEY sealing key to generate report but rather generate the whole report at once and is hard-coded which contain the info of the attesting hardware. So instead, I hereby describe the EGETKEY as private so that attacker will not have access to the report key and cannot generate its own report. Then I described destructors and equations constraining the above functions. After that, the two enclaves are defined. The first one “A” being the verifying enclave and the second one “B” being the attesting enclave. Briefly, A sends its own MRENCLAVE (derived from its log file by hashing it) to B. B then generates report body with its own MERENCLAVE and a Diffie-Hellman symmetric key. After sealing the body with the key derived from A’s MRENCLAVE using AES128-CMAC the whole report is then sent back to A. “A” enclave calls its EGETKEY to acquire a report key to verify the MAC and body, so that the data integrity is assured. Also, the accountability of this body is also assured as the MAC report is hard-coded which contains the identity characteristics of the enclave. Hence the local attestation is hereby completed, the remaining part is to send A’s ERREPORT to B for local attestation and exchange A’s Diffie-Hellman public key so that a trusted symmetric encrypted communication could therefore be established. I tried sending message with the generated key and enabled query to attack on the message, the result turns out to be unbreachable. Hence, the authenticity of message is thereby confirmed.

Remote Attestation:

In “remote.pv”, the remote attestation of Intel SGX is accomplished and evaluated. Instead of two enclaves, there are altogether four enclaves working together to simulate remote attestation. They are service provide(SP), application enclave(AE), quoting enclave(QE), and Intel Attestation Service(IAS). The SP want to verify a remote application to ensure that it is running on a secured platform protected by Intel SGX. The SP created a challenge message and send it to the application enclave, containing its SPID (registered on IAS), latest “SigRL” (Signature Revocation List), and a nonce that indicate the signature mode. AE then create a EREPORT, containing the challenge message alongside with freshly generated ephemeral public key and the target enclave’s MRENCLAVE. Having the mac appended to the report body, the whole report is then sent to QE for local attestation, which is clarified before. After verifying the mac message and body message, QE would again call EREPORT to decrypt private attesting key which is exclusively attributed to QE to sign multiple signatures. The first one is its identity signature which varies depends on the signature mode stated by SP. The second one is a knowledge signature over the platform’s MRENCLAVE. The third one is a

knowledge signature that the attestation key does not create any signatures on the “SigRL.”. Then, having all this information packed, the QE generates a quote and a mac message (with SP’s information as key generation). The quote message is then forwarded to AE, and AE forwards it to SP. SP verifies the mac message, ensuring that the message is identical. And then forwards it to IAS, and IAS eventually attest that the platform is indeed running on a SGX protected hardware. And lastly inform SP about the result. The integrity of this protocol is ensured by signing macs when delivering messages. The accountability is ensured by hard-coded EREPORT and Quote function which must contain their own identities, and most importantly, very few functions are limited to specific enclaves such as attesting private key. The authenticity is ensured by again utilizing Diffie-Hellman key attribution and symmetric key encryption. Also, the asymmetric key (attesting key) is initiated during platform initial setup phase, which is the core to the platform’s authenticity.