Technical University of Dresden

Faculty of Computer Science

Systems Engineering 1

First assignment

Report

Student: Anton Skripin
Registration number: 4979949
Study program: DSE

Dresden 2020

Model description, attestation flow.

Local Attestation. The Proverif model for local attestation consists of two subprocesses. Each subprocess represents an enclave. *Enclave_B* is the enclave that sends a challenge message to *Enclave_A* with its identity. *Enclave_A* gets the request and constructs a cryptographic report. This report consists of a report body, protection key, and message authentication code (MAC). In the current model, the report body comprises *CPUSVN*, the personal identity of *Enclave_A*, and some user data. In order to compute the message-authentication key, *Enclave_A* needs to make use of the *cmac* function that must have the following parameters: report body, report key. However, initially, *Enclave_A* has no report key. Therefore, before calling the *cmac* function it must call the function named *derive_key* that returns the report key needed for MAC calculation. *Derive_key* function *takes CPUSVN, protection key, OwnerEpoch, SealFuses, KeyID,* and *Enclave_B* as parameters. The identity of *Enclave_B* is essential because it is assumed that only the enclave can verify the report for whom this report was created. In other words, if the *derive_key* function lacked the identity of *Enclave_B* in its parameters, then *Enclave_B* would not be able to verify the report, and, therefore, the local attestation procedure would fail. After gathering all required params, the *cmac* function calculates the message authentication code, and this code is appended to the cryptographic report. This report is sent to the *Enclave_B* via a public channel. Upon receiving the report, *Enclave_B* first derives the key by calling the *derive_key* function. This operation basically returns the same report key. Thereafter, *Enclave_B* calculates the message authentication code - *MAC_OUTPUT* using the report it had got from *Enclave_A* and report key. Finally, *Enclave_B* compares *MAC_OUTPUT* with the *MAC* from *Enclave_A*. If the result of the comparison is equal, it means that the report was not modified by an attacker, and *Enclave_B* can verify the report.

Remote attestation. The Proverif model for remote attestation contains 5 subprocesses – *Remote_Challenger, SGX_Application, SGX_Application_Enclave, Intel_Quoting_Enclave, and Intel_Attestation_Service*. This model represents the following attestation flow. Firstly, an application receives a request coming outside of the platform (the message is sent by a remote challenger). After that, the application requests, *SGX_Application_Enclave* to produce an attestation. The attestation consists of two stages. The first stage is responsible for conducting a local attestation between *SGX_Application_Enclave* and *Intel_Quoting_Enclave*. *Intel_Quoting_Enclave* verifies the local attestation and sign it using its attestation key. The result of signing is stored in the quote variable. Thereafter, the quote and CMAC are sent to *SGX_Application*. *SGX_Application* simply forwards this message to *Remote_Challenger*. Finally, the *Remote_Challenger* makes use of *Intel_Attestation_Service* to verify the quote. The quote verification is done by the destructor named *checksign*. The result of calling this function is sent to *Remote_Challenger*. *Remote_Challenger* checks whether the result is true. If it is the case, then *Remote_Challenger* triggers the event called *successful_attestation* meaning that the message integrity, message authentication, and nonrepudiation properties are satisfied.

Obtained results.

The summary result provides the following conclusions. First of all, it is said that the attacker is not able to derive the key using the values it possesses in the case of local and remote attestations. Therefore, confidentiality property is guaranteed. Secondly, the invoked event *verified_report* in local attestation indicates that symmetric authentication is correct as well as the invoked event *successful_attestation* in remote attestation shows that the digital signature is valid.