Remote attestation is a fundamental concept when designing secure protocols using TEEs. One of the best ways to make it clear how this works is with an on-chain demo.

This demo is mainly possible through recent projects like Puffer Finance"RAVE", which are Solidity smart contract implementations of verifiers for SGX remote attestation. We'll walk through an end to end example, starting from the verifier and working backward.

To follow along with the post:

- [Live instance of the smart contract on Sepolia](https://sepolia.etherscan.io/address/0x290ca2ccf25b63521d3297d48b4ea2aaca2cbebb#readContract
- Forge environment for this smart contract GitHub - amiller/gramine-forge
- A Gramine project for the enclave used in this postGitHub - amiller/gramine-dummy-attester

Verification of SGX Remote attestations using Solidity.

Let's start with the on-chain instance of the attestation checker based on RAVE. This happens to be on Sepolia testnet, and is a verification contract written in Solidity.

The contract comes with a utility that decodes an encoded attestation. Essentially this is the Solidity equivalent of the "gramine-sgx-quote-view" utility from Gramine. You can use the"Read Contract" tab of Etherscan to query it. Here's an example response from decodeAttestation

.

[decodeAttestation(bytes) method Response] id string : 3342293282022782940082890185166001797 timestamp string : 2023-11-14T18:22:53.362222 version string : 4 epidPseudonym string : +CUyIi74LPqS6M0NF7YrSxLqPdX3...WiBdaL+KmarY0Je459Px/FqGLWLsAF7egPAJRd1Xn88Znrs= advisoryURL string : https://security-center.intel.com advisoryIDs string : ["INTEL-SA-00161","INTEL-SA-00219","INTEL-SA-00289","INTEL-SA-00334","INTEL-SA-00615"] isvEnclaveQuoteStatus string : CONFIGURATION_AND_SW_HARDENING_NEEDED platformInfoBlob string : 15020065000008000014140204018...58D2D8F75EAA isvEnclaveQuoteBody string : 0200010...40679b71321ccdd5405e4d54a68200000000000000000000000000000000000000000000000000000000000000000000000000 userReportData string : 9113b0be77ed5d0...00000000

For a sample attestation you can copy and paste into the decodeAttestation

field on Etherscan , copy the following:

0000000000000000000000000000000000000000000000000000000400000000000000000000000000000000000000000000000000000000000076000000000000000000000000000000

The "verify_epid" function is the main function in this demonstration.

function verify_epid(string memory userReportData, bytes memory attestation) public view returns(bool);

It can be found here: gramine-forge/src/MyRave.sol at master · amiller/gramine-forge · GitHub

The high level pseudocode is this:

- The root public key from Intel is hardcoded (but can be independently found from many other places, i.e. gramine source code, web archive snapshot)
- The attestation is parsed into a report and signature from abi encoded bytes.
- The signature is verified against the root public key, as a signature over the report data.
- The MRENCLAVE value from the report, which is a hash over the enclave program, is matched to a reference we pass in
- The "userReportData", which is set by the enclave when running, is also matched against a reference.

Note that the root of trust here starts with the manufacturer, and getting around that is out of scope for this article (although hopefully you can see how smart contracts would be useful for coordinating multiple enclaves from different manufacturers…).

The MRENCLAVE is the hash of the enclave program binary, analogous to how the address is a codehash over the contract bytecode. In this case the MRENCLAVE refers to the dummy attestation enclave explained in a moment, which is e3c2f2a5b840d89e069acaffcadb6510ef866a73d3a9ee57100ed5f8646ee4bb.

Checking the signatures relies primarily on verifying an RSA signature. These have already been implemented in solidity, so these are through imports from dnssec-oracle.

Dummy Attestation service in SGX/Gramine/Python:

Let's look at the enclave that generated these attestations. I defined a dummy enclave that generates remote attestations for any userReportData chosen by the user. It attests to anything, so it's not useful for any secure application. However, an attestation can only be generated from a valid SGX machine, so it's not a trivial function to satisfy either.

github.com

amiller/gramine-dummy-attester/blob/master/dummyattester/enclave/main.py

import sys import hashlib

if **name** == '**main**': while True: # Read a message specifying a user report data (64 bytes) line = sys.stdin.readline() hexreportdata = line.strip() assert len(hexreportdata) == 128

```
    # TODO: abi encoding?
    # Address String
    message = bytes.fromhex(hexreportdata)

    # Set the user data
    with open("/dev/attestation/user_report_data", "wb") as f:
        f.write(message)

    # Read the quote
    with open("/dev/attestation/quote", "rb") as f:
```

This file has been truncated. show original

To make it as conceptually simple as possible, the dummy attester enclave is implemented in python using Gramine:

- It reads a 64 byte string from stdin.
- Writes this to the attestation interface /dev/attestation/user_report_data
- It reads a result from the interface /dev/attestation/quote.
- Return this quote.

Note that you can try reproducing the enclave binary, without having to have an SGX machine at all. The main limitation of this python/gramine approach is that it seems more likely to suffer reproducibility rot, as the particular python libraries packaged here may not remain identical when package maintainers update them. But maybe this can be mitigated. The README does list a particular mrenclave, so you can see if it remains reproducible using docker.

As a best effort service: for a little while following this post, I'll provide this service that generates fresh attestations to play with on demand: http://dummyattest.ln.soc1024.com/9113b0be77ed5d0d68680ec77206b8d587ed40679b71321ccdd5405e4d54a6820000000000000000000000000000000000000000000000000000000000000000

(If the service goes down, it will remain possible to run your own enclave using the docker compose instructions)

Conclusion

This on-chain Solidity attestation checker in this example is just a starting point. It's necessary but not sufficient. Any application or framework using remote attestation would need to add some more constraints to the verification. Most enclaves don't get a straightforward "OK" but come with warnings - which configurations should we tolerate? Do we only care about a single program or a range of acceptable programs? The structure of the userReportData itself is also up to the application to define. And what about DCAP, as this demo appears to be EPID specific?

Answering any of these questions requires deeper understanding of SGX to solve… but, it may only require writing more Solidity code.

The main takeaway from this demonstration is that verifying remote attestations is a naturally on-chain activity. The availability of Solidity libraries for verifying remote attestations now makes Solidity a very plausible way to define remote attestation policies.