# Cloud Provider's Based Attestation

**Carlos Molina-Jimenez, Jon Crowcroft**
Dept. of Computer Science and Technology, Univ. of Cambridge
{carlos.molina, jon.crowcroft} @cl.cam.ac.uk
**Dann Toliver**
Todaq
dann.toliver@todaqfinance.com

12th January 2024

**Abstract**

One of the central aims of the CAMB project is the development and evaluation of attestables launched on the Morello Board. An attestable is a trusted execution environment that a user can request from the Morello Board's owner (for example, a cloud provider) to execute an exfiltration sensitive application.

A fundamental challenge in this scenario is to develop attestation mechanisms that the user can use to be reassured that the attestable will deliver the security properties that he or she needs. Different approaches with different levels of complexities and trust can be taken to address the issue. In this document we discuss the design and implementation of an attestation mechanism that relies on the trustworthiness of a cloud provider: the central idea is that the cloud provider launches the attestable on a Morello Board and returns to the user a signature on a string of data that describes the technical parameters of the attestable such as the process ID and port number to interact with the attestable.

## 1 Introduction

Remote attestation is a procedure that the user of a software (usually called the verifier or challenger) conducts to verify that the the integrity of the software. In other words, the verifier, verifies that the current configuration of the software is in a well–defined expected state that is considered acceptable for delivering a trustworthy service. The responder to the attestation request is usually called the prover and is response produces evidence, normally a cryptographic token that convinces the verifier.

As discussed in [7], several attestation mechanisms have been suggested that aim at proving different properties, provide different levels of certainly and involve different levels of complexity. For example, some mechanisms attest the

integrity of a single node, more ambitious mechanisms attest the integrity of whole network; some mechanisms rely on hardware (for example the existence of hardware root of trust) while others are involve only software operations.

It is widely acknowledged that remote attestation is simpler to implement and more robust when it is supported by specialised hardware rather than implemented only in software. However, the specialised hardware might not be available in the device under attestation to reduce its complexity, physical size or cost. Within this category falls the current release of the Morello Board— it does not come with built–in specialised hardware to help with remote attestation. This fact has motivated the CAMB project to investigate innovative solutions.

In this report, we discuss the design and implementation of an attestation mechanism that relies on the trustworthiness of a cloud provider. We explain the conceptual idea in Section 2. Intuitively, we assume that cloud provider has strong business and judiciary motivations to deliver what it promises to the client (the verifier) through a cryptographic signature: to deploy the clients application within an attestable launcher on a Morello Board. In Section 3 we discuss a python implementation of the attestable launcher that is capable on launching attestables on a Morello Board with the assistance of library comparmentalization [1].

## 2 Cloud provider's based attestation

The central idea of our approach is based on two assumption:

- Once launched, the attestable observe the security properties that the attestable's designer guarantees:
  1. It is a black box that prevents observation of data and computation.
  2. The code an attestable is running cannot be changed.
- The cloud provider has strong business incentives to launch the attestable correctly, that is, in a manner that the attestable is technically able to meet the two properties mentioned above. The cloud provider expresses his commitment in the attestation document that it signs and which can be regarded as an attestable's certificate signed by a trusted party.

Fig. 1 illustrates the time line of events. In the figure, a single line box labeled *env* represents a conventional execution environment without enhanced security features. Examples are conventional devices like mobile phones, laptops and Morello Boards. The figure shows three *envs*: Alice uses one of them to run her application; Bob uses an *env* to run his *attestablelauncher* and another one to launch, upon Alice's app's request, an attestable. The double line box represents an attestable, that is, an exfiltration resistance execution environment. Attestables can be launched within *envs* that offer facilities for creatting them. The figure shows a Morello Board with cheri–capabilities that the CAMB pro-

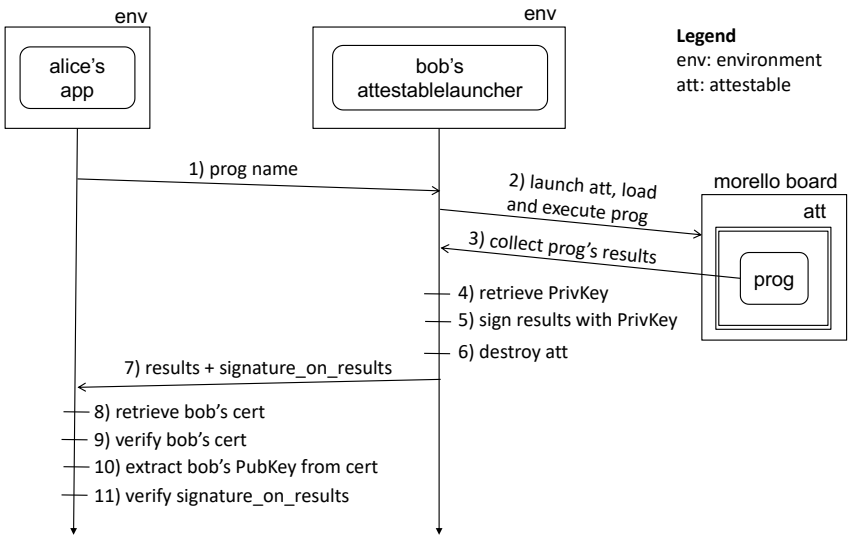


Figure 1: Architecture and event timeline of the attestable launcher.

ject uses to create attestables. *Prog* is a piece of code that results in highly sensitive computation, say because it inputs or outputs high sensitive data.

Let us assume that at some point Alice's application needs to execute a piece of code (show as *prog* in the figure) that she considers exfiltration sensitive.

To be safe, Alice opts to execute *prog* within an attestable rented from a cloud provider (Bob) that offers attestables launcher on Morello Boards deployed in his premises. The procedure develops as follows:

1. Alice's app sends a message to Bob's *attestablelauncher*. that includes either the name of the program or the actual code to execute. To keep the discussion simple, let us assume that Bob is in possession of *prog*, thus, Alice need to indicate *prog name*, as shown in the figure. The *attestablelauncher* is essentially an API that client cloud can use to request the attestables and to interact with the code executed within them.

2. The *attestablelauncher* launches an attestable *att* within a Morello Board, loads it with *prog* and triggers its execution.

3. *prog* sends the result of its execution to the *attestablelauncher*. The results depend on the particularities of *prog*. For instance, if *prog* is not interactive, it can run and return to the *attestablelauncher* the actual output of *prog's* computation and stop. However, *prog* can be interactive; in this case, it return to the *attestablelauncher* the information that Alice's app need to interact with it; for example, the address of the Morello Board, its process ID, the number of the port where it is listening for messages and

a public key. In this discussion, we assume that *prog* is not interactive.

4. The *attestablelauncher* retrieves its private key, for example, from a local file in PEM format [9].
5. The *attestablelauncher* uses its private key to sign *prog's* execution.
6. The *attestablelauncher* destroys the attestable and wipes its memory.
7. The *attestablelauncher* sends a message to Alice's app that includes *prog's* results and the signature produced by the private key.
8. Alice's app retrieves the certificate of Bob's *attestablelauncher*, for example, from a local file in PEM format [9] or from a public certificate repository.
9. Alice's app verifies the status of the certificate, for example, the signature of its issuer and validity period.
10. Alice's app extracts the the *attestablelauncher*'s public key from the certificate.
11. Alice's app verifies the signature: it contrasts the *results* against *signature_on_results* to verify that they match, that is, none of them has been altered in transit.

# 3 Implementation

We have implemented a proof of concept version of Fig. 1 in Python3 and used to launch examples of *prog* written in C within compartments created using library compartmentalisation available in cheriBSD_ver22.12.

The whole code is available for testing from git [6]. Each python module (classes and functions) are thoroughly documented. In this section, we present only a summary.

## 3.1 SSL secure channel and authentication

The interaction between Alice's app and Bob's *attestablelauncher* is conducted over a secure channel created with Openssl [4]. At this experimental stage we use self–signed certificates created with openssl run as command line (`bash-$ openssl genrsa -aes256`). I have OpenSSL 3.2.0 23 installed on a Mac Book Air. The step–by–step procedure is thoroughly documented in [5]. Another alternative is to create the public/private key pairs and certificates directly from python [3] using the Cryptography library [8]. Alice's app uses the secure ssl channel to authenticate Bob's *attestablelauncher* though strictly speaking authentication is not necessary because Alice's app verifies Bob's *attestablelauncher* certificate before using his public her to verify the signature on the results (see step

9 of Fig. 1). However, we leave the secure channel active in the current implementation because it obviated the need to encrypt the results that Bob's *attestablelauncher* sends to Alice's app in step 7 of Fig.1.

### 3.2 The attestable launcher

We have implemented the *attestablelauncher* as a conventional server that listen for connection requests to arrive at a conventional socket port (for example at port number 8888). When a request arrives it creates a thread that runs a handler to deal with the request. The request handler is responsible for executing steps 2–7 of Fig. 1. The current implementation takes advantage of Python3's sub–process management facilities. We use *Popen* [10], which enables the creation of sub–processes from code written in other languages like C.

### 3.3 The attestable

In the current implementation we use the library compartmentalization facilities available from cheriBSD version 22.12 onwards [2]. As explained in [1] and in the *$ man compartmentalization*, a programmer can create compartments at link time. The following code shows how to compile and link a *main.c* program with two compartments: *alicecompartment.c* and *bobcompartment.c*. The code of Bob's

```
/*
  Programmer: Regis Rodolfo Schuch regis.schuch@sou.unijui.edu.br
  Date: 15 Dec 2023, Univ Ijui
  Edited: Carlos.Molina@cl.cam.ac.uk, 17 Dec 2023

  helloAliceBobCompartment.c
  Is a hello world program to experiment with compartmentartalisation on
  the Morello Board.
  It consists of three modules
  1) helloAliceBobCompartment.c
  2) alicecompartment.c and its corresponding alicecompartment.h file
  3) bobcompartment     and its corresponding bobcompartment.h   file

  It has been tested on a Morello Board with cheriBSD version 22.12

   Compilation:
  cm770@morello-camb-1: $ clang-morello -march=morello+c64 -mabi=purecap
                          -c -o alicecompartment.o alicecompartment.c
  cm770@morello-camb-1: $ clang-morello -march=morello+c64 -mabi=purecap -shared
                          -o libaliceexample.so alicecompartment.o
  cm770@morello-camb-1: $ clang-morello -march=morello+c64 -mabi=purecap
                          -c -o bobcompartment.o bobcompartment.c
  cm770@morello-camb-1: $ clang-morello -march=morello+c64 -mabi=purecap -shared
                          -o libbobexample.so bobcompartment.o
  cm770@morello-camb-1: $ clang-morello -march=morello+c64 -mabi=purecap
                          -o helloAliceBobCompartment helloAliceBobCompartment.c
```

```
                        -L. -laliceexample -lbobexample -Wl,
                        --dynamic-linker=/libexec/ld-elf-c18n.so.1

  Execution:
  cm770@morello-camb-1: $ env LD_C18N_LIBRARY_PATH=. ./helloAliceBobCompartment
  Hello from the alicecompartment!
  Hello from the bobcompartment!
*/

#include "alicecompartment.h"
#include "bobcompartment.h"
int main() {
    //Call the compartment function
    alicecompartment();
    bobcompartment();
    return 0;
}
```

```
/*
 * alicecompartment.c
 */
#include <stdio.h>

void alicecompartment() {
    printf("Hello from the alicecompartment!\n");
}
```

```
/*
 * alicecompartment.h
 */
#ifndef COMPARTMENT_H
#define COMPARTMENT_H

void alicecompartment();

#endif // COMPARTMENT_H
```

```
/*
 * bobcompartment.c
```

```
 */
#include <stdio.h>

void bobcompartment() {
    printf("Hello from the bobcompartment!\n");
}
```

```
/*
 * bobcompartment.h
 */
#ifndef COMPARTMENT_H
#define COMPARTMENT_H

void bobcompartment();

#endif // COMPARTMENT_H
```

### 3.4 Signature on results with attestable launcher's private key

As indicated in step 5 of Fig. 1 the *attestablelauncher* signs the results of the execution of *prog*. The *attestablelauncher.py* uses the `sign_msg_payload` method of the *Signpayload* class to place the signature.

```
"""
Signs the payload given as an array of bytes and sign
it under the private key. Return the resulting signature
as an array of bytes.
"""
  def sign_msg_payload(self, pri_key, msg_payload):
      payload= msg_payload.encode()

      signature_on_msg = base64.b64encode(
          pri_key.sign(
              payload,
              padding.PSS(
                  mgf= padding.MGF1(hashes.SHA256()),
                  salt_length= padding.PSS.MAX_LENGTH),
              hashes.SHA256()))
      sig = base64.b64decode(signature_on_msg)
      return sig  # return a byte object that contains UTF-8
                  # sig can be sent directly over a socket to
                  # a remote receiver
```

### 3.5 Verification of the attestable launcher's signature

As indicated in step 11 of Fig. 1 *Alice's app* verifies the signature of the *attestablelauncher* place on the results before accepting them. In the code available from Git, Alice's application is represented by *cliRqExecProgInAttestable.py*. This code uses the `verify_msgpayload` method of the *Verifysignatureonpayload* class to verify the signature.

```
def verify_msgpayload(self, pub_key, msg_payload, signature_on_msgpayload):
    signature= signature_on_msgpayload # encoded as byte obj
                                       # in UTF-8
    payload_contents=msg_payload.encode()
    len(signature_on_msgpayload))
    try:
        pub_key.verify(
            signature,
            payload_contents,
            padding.PSS(
                mgf = padding.MGF1(hashes.SHA256()),
                salt_length = padding.PSS.MAX_LENGTH),
            hashes.SHA256())
        print("\n signature on message payload verified successfully...")
    except cryptography.exceptions.InvalidSignature as e:
        print("ERROR: msg_payload and/or signature_msg failed verification!")
```

## 4 Work in progress and challenges

The current implementation is not finished yet. The Python code that places the signature and verifies has been thoroughly tested on a Mac Book Air, testing of these cryptographic procedures on a Morello Board is pending. A more significant task is to make *prog* interactive. The current implementation is able to receive only the final result from *prog*, that is its output upon completion. The code of the *attestablelauncher*'s has to be enhanced to be able to receive results from *prog* such as its socket port number while *prog* is still executing. A more fundamental challenge is the exploration of the attestation mechanism. We regard cloud provider's based attestation as a pragmatic and simple solution to a hard problem yet we are aware that heavily depends on the trustworthiness of the cloud provider.

## References


[1] Cheri Team. *Library Compartmentalization.* https://www.cheribsd.org/tutorial/23.11/c18n/index.html. visited on 9 Jan 2024. Nov. 2023.

[2] CHERIBSD team. *CheriBSD features.* https://ctsrd-cheri.github.io/cheribsd-getting-started/features/index.html visited on 11 Jan 2023. Jan. 2023.

[3] Cryptographyteam. *X 509 Tutorial*. https://cryptography.io/en/latest/x509/tutorial/. Visited on 10 Jan 2024. Jan. 2024.

[4] PythonDocs. *ssl — TLS/SSL wrapper for socket objects. Source code: Lib/ssl.py.* June 2023.

[5] Carlos Molina-Jimenez. *How to Create Self Signed Certs*. https://github.com/CAMB-DSbD/FEWD/blob/main/docs/howtocreateSelfSignedCerts.txt. Visited on 10 Jan 2024. Oct. 2023.

[6] Carlos Molina-Jimenez, Jon Crowcroft and Dann Toliver. *Attestablelauncher*. https://github.com/CAMB-DSbD/attestablelauncher. Visited on 10 Jan 2024. Jan. 2024.

[7] Ioannis Sfyrakis and Thomas Gross. *A Survey on Hardware Approaches for Remote Attestation in Network Infrastructures*. https://arxiv.org/pdf/2005.12453.pdf. arXiv:2005.12453 [cs.CR]. July 2020.

[8] Reaper Hulk. *Cryptography 41.0.7*. https://pypi.org/project/cryptography/. Nov. 2023.

[9] Patrick Nohe. *How to convert a certificate to the correct format*. https://www.thesslstore.com/blog/how-to-convert-a-certificate-to-the-correct-format/. Aug. 2018.

[10] Docspython. *Subprocess–Subprocess management*. https://docs.python.org/3/library/subprocess.html. Dec. 2023.