Project 1: ACME Client

1 | General Information

- This is the first of two graded course projects. This project will determine **16.67% of your final grade**. Not handing in this project will result in a 1 for this project, but passing the course is still possible. However, if you do not hand in the projects (this one and one coming up later in the semester), you must at least have an exam grade of 5 to pass the course.
- This project must be completed **individually**. You are of course allowed to discuss the ACME protocol and RFC with your colleagues; however, discussions about the implementation and sharing of code is not permitted.
- **Plagiarism checks** will be performed. In addition to manual checks we use an online plagiarism checker hosted by a foreign research institute. Therefore, please ensure there is no sensitive personal information in your code.
- The due date of this project is **2023-11-10 23:59**. This is a hard deadline that will be automatically enforced.
- The instructions in this document are very limited---this is on purpose. Most information needed for this project is very well documented in standards. Familiarizing yourself with these standards is one of the goals of this project.
- In case you run into issues during the project, please consult the FAQ before asking a question on GitLab.

2 | ACME Protocol

Public Key Infrastructures (PKIs) using X.509 certificates are used for many purposes, the most significant of which is the authentication of domain names. Certificate Authorities (CAs) are trusted to verify that an applicant for a certificate legitimately represents the domain name(s) in the certificate. Traditionally, this verification is done through various ad-hoc methods.

The Automatic Certificate Management Environment (ACME) protocol (RFC8555) aims to facilitate the automation of certificate issuance by creating a standardized and machine-friendly protocol for certificate management.

3 | Your Task

Your task is to write an application that implements ACMEv2. However, to make the application self-contained and in order to facilitate testing, your application will need to have more functionality than a bare ACME client. The concrete requirements for your application are described in the remainder of this section.

3.1 | Application Components

Your submitted application must consist of the following components:

- *ACME client:* An ACME client which can interact with a standard-conforming ACME server.
- *DNS server:* A DNS server which resolves the DNS queries of the ACME server.
- *Challenge HTTP server:* An HTTP server to respond to http-01 queries of the ACME server.

- *Certificate HTTPS server:* An HTTPS server which uses a certificate obtained by the ACME client.
- *Shutdown HTTP server:* An HTTP server to receive a shutdown signal.

3.2 | Required Functionality

In order to receive full marks, your application should be able to

- use ACME to request and obtain certificates using the `dns-01` and `http-01` challenge (with fresh keys in every run),
- request and obtain certificates which contain aliases,
- request and obtain certificates with wildcard domain names, and
- revoke certificates after they have been issued by the ACME server.

Note that the automatic tests enforce a timeout of 1 minute. Furthermore, the final grading will be determined by a second offline run of the tests. You are therefore advised to avoid nondeterministic behavior of your implementation, e.g., due to multi-threading.

3.3 | Input and Output

3.3.1 | File layout

We will supply you with a basic skeleton which you should use for submission. Three files in this skeleton are of particular importance:

- `pebble.minica.pem` This is the CA certificate for the private key used to sign the certificate of the HTTPS endpoint of the ACME server itself. Use this as a trust root to check the ACME server's certificate when interacting with this endpoint. You will lose points if your application sends more than one request to an ACME server with an invalid certificate (one request is needed to obtain the certificate and check its validity).
- `compile` This file will be executed by the automated-testing environment before any tests are run. You should modify this file. If your project needs to be compiled, this file should contain the commands needed to compile the project. If no compilation is needed, this file can do nothing (or install dependencies). Note that you are only allowed to install explicitly allowed dependencies, see below.
- `run` This file will be executed by the testing environment when the tests are being run. You should modify this file. It will receive the command-line arguments listed in Section 3.3.2. Your `compile` script may overwrite this file.

Note that all paths in your code should be relative to the root of the repository.

3.3.2 | Command-line arguments

Your application should support the following command-line arguments (passed to the `run` file):

Positional arguments:

- `Challenge type` *(required, `{dns01 | http01}`)* indicates which ACME challenge type the client should perform. Valid options are `dns01` and `http01` for the `dns-01` and `http-01` challenges, respectively.

Keyword arguments:

- `--dir DIR_URL` *(required)* `DIR_URL` is the directory URL of the ACME server that should be used.
- `--record IPv4_ADDRESS` *(required)* `IPv4_ADDRESS` is the IPv4 address which must be returned by your DNS server for all A-record queries.
- `--domain DOMAIN` *(required, multiple)* `DOMAIN` is the domain for which to request the certificate. If multiple `--domain` flags are present, a single certificate for multiple domains should be requested. Wildcard domains have no special flag and are simply denoted by, e.g., `*.example.net`.
- `--revoke` *(optional)* If present, your application should immediately revoke the certificate after obtaining it. In both cases, your application should start its HTTPS server and set it up to use the newly obtained certificate.

Example: Consider the following invocation of `run`:

```
run dns01 --dir https://example.com/dir --record 1.2.3.4 --domain
netsec.ethz.ch --domain syssec.ethz.ch
```

When invoked like this, your application should obtain a single certificate valid for both `netsec.ethz.ch` and `syssec.ethz.ch`. It should use the ACME server at the URL `https://example.com/dir` and perform the `dns-01` challenge. The DNS server of the application should respond with `1.2.3.4` to all requests for `A` records. Once the certificate has been obtained, your application should start its certificate HTTPS server and install the obtained certificate in this server.

3.4 | Server sockets

Your application should be running the following services on the following ports:

- *DNS server:* should run on UDP port 10053. The ACME server will direct all of its DNS queries to this DNS server.
- *Challenge HTTP server:* should run on TCP port 5002. The ACME server will direct all `http-01` challenges to this port. Note that this deviates from RFC8555.
- *Certificate HTTPS server:* should run on TCP port 5001. The testing environment will issue a `GET /` request to this server in order to obtain the certificate served by this server. The server should serve the full certificate chain obtained from the ACME server, i.e., including the intermediate certificate.
- *Shutdown HTTP server:* should run on TCP port 5003. Once testing is complete, the testing environment will issue a `GET /shutdown` request to this server. When this request is received, your application should terminate itself.

4 | Guidelines and Restrictions

- Only programming languages, modules, libraries, software packages, etc. that are explicitly listed on the whitelist may be used. We will happily consider requests to extend this whitelist. In principle, we will allow anything as long as
    - we can run it in our automated testing environment,
    - it does not implement core ACME functionality, and
    - it does not implement JSON Object Signing and Encryption (JOSE) functionality.

You can direct request for extensions of the whitelist by opening a Gitlab issue.

- Your application may spawn other processes. However, the entire application must start solely by executing the `run` file.
- You can use any key type and key size supported by openSSL and Pebble for the requested certificate.

5 | Testing

We recommend the use of Pebble for local testing. Pebble is a lightweight ACME server designed specifically for testing. During the automated testing your application will be tested against a Pebble server.

Note that by default, Pebble is configured to reject 5% of good nonces. The grading environment, on the contrary, does not reject valid nonces. Hence, we do not expect your application to gracefully handle cases of unjustly rejected nonces. To achieve the same Pebble behavior locally as is used in the CI testing, adjust the value of the corresponding environment variable: `export PEBBLE_WFE_NONCEREJECT=0`.

6 | Grading

You should submit your code by pushing it to a personal repository we have set up for you at

```
https://gitlab.inf.ethz.ch/PRV-PERRIG/netsec-course/project-acme/netsec-2022-acme/<YOUR_ETH_USERNAME>-acme-project
```

This repository has continuous integration (CI) set up and will automatically test your submission. Note that a green checkmark in the gitlab CI only indicates that the test runner terminated successfully, **not** that your code actually passed the tests. To obtain your test results, you need to inspect the output of the runner `give-score`.

In grading, tour score for the project will be calculated as follows:

```
final_score = 1 + 5 * nr_passed_tests/nr_total_tests
```

The fraction is the final score reported in the output of the `give-score` runner in the gitlab-CI. We will run your code 10 times and take the median score to rule out irreducible non-determinism. **We will not make adjustments to this median score.** The only exception to this is when we notice that you did not follow the project rules or when we detect plagiarism. The auto grader is highly transparent and you have plenty of time to ensure that your code passes the test cases. If you experience problems with the auto grader, you must address these **before** the deadline.

Note that in the beginning of the project some test cases might be added. If this happens, we will clearly communicate this, and we guarantee not to add any test cases in the last two weeks of the project.

Note also that the CI server does not have unlimited resources. It is likely that testing times will increase when many students start submitting solutions at the same time. We will not accept this as an excuse for

failed tests and we will not extend the deadline because of this. There is plenty of time to submit and most testing can be performed locally with Pebble.