dilithium-cortexm

Denisa O. C. Greconici, Matthias J. Kannwischer, Daan Sprenkels

25 January 2021

This repository contains the supplementary code for our efforts to write an optimized implementation of Dilithium for the ARM Cortex M4 and Cortex M3 architectures. We are targeting the STM32F407 Discovery and the Arduino Due development boards, which respectively hold an STMicroelectronics STM32F407 and Atmel SAM3X8E microcontroller.

The corresponding paper is available at https://eprint.iacr.org/2020/1278.

Table of contents

Repository structure

- `dilithium` holds our optimized implementation of the Dilithium signature scheme for the Cortex-M4 and Cortex-M3.
- `kyber` holds the optimized NTTs for Kyber.
- `newhope` holds the optimized NTTs for NewHope.
- `libopencm3` contains a version of the `libopencm3` library, which is used for abstracting the hardware from the STM32F407 board.
- `vendor` contains some (boilerplate) build tools, primarily the tooling for building binaries to the Arduino Due, without needing the Arduino IDE.

In each code directory, the `m3` will contain the code that is specific to the Cortex-M3. If applicable, the `m4` directory contains the code specific for the Cortex-M4. Typically, these are the directories that contain the handwritten assembly code. Furthermore, `common/` holds some vendored building blocks (mainly SHAKE256).

Getting started

We have tried to make this README file as clear as possible. However as always the case with embedded programming, problems may arise that we have not foreseen. Of course, it will not possible to exactly match your setup as well.

Cloning the repository

If you do not have this package locally yet, clone the repository using the `--recursive` option:

```
git clone --recursive https://github.com/dilithium-cortexm/dilithium-cortexm.git
```

In case you have a downloaded CHES artifact, you should already have everything you need, including the submodules. We have kept the `.git/` directories intact.

Required software

First, install the following software: - GCC for bare-metal ARM (`arm-none-eabi-gcc`). If you are installing this using your operating system's package manager, you might need to install other packages aside from GCC (like Newlib/Binutils). - `pyserial` - `stlink` version `>=1.6.0` (we have had some issues with other versions during development, including `1.6.1`) (only for M4) - The Arduino IDE (only for M3) - `bossa` `>=1.9.0`. Some older versions, particularly those in the Ubuntu repositories do not work (only for M3)

```
# On Arch Linux, use:
yay -S arm-none-eabi-gcc arduino arduino-cli bossa python-pyserial stlink
```

User-access to serial devices

Depending on your operating system, your user may or may not be able to directly access the serial devices in `/dev`. If you know that you are not affected by this, you can skip this step.

Fix via udev One way to fix this is to whitelist the device in udev, i.e., add a rule to your udev configuration that allows user access to the USB device.

```
# Whitelist the Arduino SA device for user access in udev.
echo 'ATTRS{idVendor}=="2341", ATTRS{idProduct}=="003d", TAG+="uaccess" # Arduino Due' |
sudo tee /etc/udev/rules.d/10-arduino-due.rules >/dev/null

# Reload udev rules.
sudo udevadm control --reload-rules && sudo udevadm trigger
```

You can use `lsusb` to find out what the vendor and product ID is of a USB device. For example, `lsusb` reports, for the ST-LINK/V2 device:

```
Bus 003 Device 021: ID 0483:3748 STMicroelectronics ST-LINK/V2
```

Here, `0483` is the vendor ID, and `3748` is the product ID. Then you will construct a udev rule like this:

```
# STM32F3DISCOVERY rev A/B - ST-LINK/V2
ATTRS{idVendor}=="0483", ATTRS{idProduct}=="3748", TAG+="uaccess"
```

Fix via dialout group There is another way to fix this, should the regular udev method does not work for you, you can allow blanket access to the usb devices for your user. To enable this, add your user to the `dialout` group. Logout and login again after applying this change. Beware that on some systems, this group may exist under a different name.

STM32F4 Discovery setup

The STM32F407 device is connected with USB to the host computer. For serial communication we use a separate serial device. To connect the serial, communication device, connect its `rx` and `tx` pins to the `PA2` and `PA3` pins of the STM32F407 discovery board. `GND` should be connected through the USB cable, but you might also need to connect the serial-device `GND` to the `GND` of the STM32F407 board.

To interact with the hardware, we use the `libopencm3` library, which provides full support for the STM32F407.

Flashing the code

STM32F407 binaries are flashed using the `st-flash` command, which is part of the `stlink` package. To flash a binary, use:

```
# Flash a dilithium test binary
st-flash write m4/bin/dilithium3_test.bin 0x8000000
```

Serial console

The PySerial package provides the `miniterm` tool, which is used to start a serial monitor to some device.

To start up the console for the STM32F407, use:

```
# Start a serial console to the /dev/ttyUSB0 device with a baud rate of 115200.
miniterm /dev/ttyUSB0 115200

# Use Ctrl+] to exit the console
```

Arduino Due setup

The Arduino Due is connected with USB to the host computer. The USB driver has serial communication built-in, so it is not necessary to attach a separate serial-communication device. Beware though that the Arduino has two micro-USB sockets. For programming, use the one closest to the power jack.

For interaction with the hardware, we have tried to use the Arduino library wherever we can. However it turns out that the Arduino library is not very complete at all. Therefore, we sometimes fall back to using `libsam` (which exposes some of the ATSAM3X8E's peripherals) or programming in bare-metal.

Flashing the code

For flashing code to an Arduino Due, we use the Bossa tool.

Serial console

The PySerial package provides the `miniterm` tool, which is used to start a serial monitor to some device.

To start up the console for the STM32F407, use:

```
# Start a serial console to the /dev/ttyUSB0 device with a baud rate of 115200.
miniterm /dev/ttyUSB0 115200

# Use Ctrl+] to exit the console
```

Building and running `dilithium/`

First, `cd` into the `dilithium/` directory. In this directory, `m3/` contains the files specific to the Cortex-M3 and `m4/` contains the files specific to the Cortex-M4. `dilithium/` contains the platform-independent implementation of dilithium, and `common/` holds some vendored building blocks (mainly SHAKE256).

To build and test the software for the Cortex-M4, run:

```
# Build the software, this will put a binary file at `m4/bin/dilithium3_test.bin`.
make -C m4

# In a separate terminal, start up miniterm.
miniterm /dev/ttyUSB0 115200

# Flash the firmware to the device
st-flash write m4/bin/dilithium3_test.bin 0x8000000

# Trigger a reset
st-flash reset

# Now the testing will start, and the test results should be printed to the serial console.
```

To build and test the software for the Cortex-M3, run:

```
# Build the software, this will put a binary file at `m3/build-arduino_due_x/m3_.bin`.
make -C m3


# Flash the firmware to the Arduino
bossac -a
bossac --erase --write --verify --boot=1 --port=/dev/ttyACM0 m3/build-arduino_due_x/m3_.bin

# In a separate terminal, start up miniterm (Arduinos use a baud rate of 9600).
miniterm /dev/ttyACM0 9600

# You may need to manually reset the device to start the program.
# The test results should be printed to the serial console.
```

At this point, feel free to take a look at the `m3/Makefile` and `m4/Makefile`. There you will find all the customizations that are available for our implementations. For example, `DILITHIUM_MODE` describes which parameter set of Dilithium should be used; and `SIGN_STACKSTRATEGY` specifies which of the stack-space scenarios should be compiled. Also there are binaries available for testing, benchmarking, profiling, and generating testvectors.

Building `kyber/` and `newhope/`

Building the `kyber/` and `newhope/` code for the Cortex-M3 is similar to building Dilithium for that platform, except all the code is already in the top-level directory, i.e., you `cd` to that directory and do `make` immediately without `-C m3`.

The other settings remain the same. To switch to the different parameter set, change the `CRYPTO_PATH` in `kyber/Makefile` or `newhope/Makefile`. Available parameter sets are

- `kyber/kyber512/`
- `kyber/kyber768/`
- `kyber/kyber1024/`
- `newhope/newhope1024cca/`
- `newhope/newhope1024cpa/`

Reproducing results

Running benchmarks

During the writing of our paper, we have automated most of our benchmarking pipeline. The scripts that we used are included in the `dilithium` directory. The `m4benchmarks.py` script instruments the discovery board to generate benchmarks, and `m3benchmarks.py` instruments the Arduino Due to generate its benchmarks.

Please excuse us for the quality of that code. It was not initially written for being published. The benchmarking scripts do not expose a command-line interface; you have to edit any settings in the source.

These scripts generate the literal latex files that are used in the paper. Note that, because of the variable runtime of the Dilithium signing algorithm, you'll need to run about 10000 iterations. Be advised that the Arduino Due is pretty slow, and that our benchmarking is not really optimized; these benchmarks can take a while to complete (a couple of days).

Version info

The benchmarking scripts have been confirmed to run from a host using the following software:

```
python --version
# Python 3.9.1

arm-none-eabi-gcc --version
```

```
# arm-none-eabi-gcc (Arch Repository) 10.2.0
# Copyright (C) 2020 Free Software Foundation, Inc.
# This is free software; see the source for copying conditions.  There is NO
# warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

st-flash --version
# v1.6.1

bossac --help | head -n2 | tail -n1
# Basic Open Source SAM-BA Application (BOSSA) Version 1.9.1

python -c 'import serial; print(serial.VERSION)'
# 3.5

make --version | head -n1
# GNU Make 4.3

# libopencm3 is included in the libopencm3/ directory.
(cd libopencm3 && git rev-parse HEAD)
2ce5cc58ce5a66b6a5026357bc8a2716f8839033

# Arduino-Makefile is included in the vendor/ directory.
(cd vendor/Arduino-Makefile/ && git rev-parse HEAD)
# e870443f4824cbbcd560f500c6072012dd668f62

# ArduinoCore-sam is included in the vendor/ directory.
(cd vendor/ArduinoCore-sam/ && git rev-parse HEAD)
# eed66e7977b4037f49bbcd832437d4bcf72c0f3f

# Pandoc is used to convert markdown text files to PDFs.
# pandoc --version | head -n 3
pandoc 2.11.3
Compiled with pandoc-types 1.22, texmath 0.12.1, skylighting 0.10.2,
citeproc 0.3.0.2, ipynb 0.1.0.1
```

Troubleshooting (Cortex-M3)

`No device found on /dev/ttyACM0`

This error occurs when executing `make raw_upload`. It is caused by `bossac` not being able to reach the Arduino Due through the `/dev/ttyACM0` serial port.

```
# Check if you can access it at all.
touch /dev/ttyACM0
```

`touch: cannot access '/dev/ttyACM0': No such file or directory` Check your connections, and check if the serial port is exposed on a different device. If the serial port is exposed on a different device, use the `ISP_PORT` setting to override it.

`touch: cannot touch '/dev/ttyACM0': Permission denied` Make sure your user has the correct permissions for writing to the serial port. Refer to the udev configuration listed in the intro.

No error (option 1)

```
# Check if the device is in use by some other process:
sudo lsof | grep /dev/ttyACM0
```

If another process is currently using the device, make sure to kill it. In particular, I have bad experiences with the `modem-manager` service on Debian-based systems in the past.

No error (option 2) The Arduino Due may be in an uninterruptable state. Reset it using the RESET button on the board.

`Device unsupported`

The Arduino Due is probably in an uninterruptable state. Reset it using the RESET button on the board.

If this does not unstuck the device, erase the chip by holding the ERASE button for 5 seconds and/or waiting for 5 seconds more or replug the device.

Troubleshooting (Cortex-M4)

`[!] send_recv send request failed: LIBUSB_ERROR_TIMEOUT`

The STM32F407 is unresponsive. This happens (for example), when a flashing operation is interrupted. Replug the device to unstuck it.

`miniterm` comand not found

`miniterm` may not be installed in your `$PATH`. Perhaps, on your installation it is called `miniterm.py`. Otherwise, put this tool in your path; or alternatively call it using `python -m serial.tools.miniterm`.