# Project 1: A Library for Arithmetic Primitives

## 1 Introduction

This project implements the fundamental mathematical operations that underpin modern cryptography implementations, and especially zero-knowledge proof implementations. You will implement three key components in Rust:

1. **Modular arithmetic**: Addition, subtraction, multiplication, and division of integers modulo a prime $p$. (This is equivalent to the arithmetic of elements of the field $\mathbb{F}_p$; in the class we will usually use this algebraic viewpoint.)

2. **Elliptic curve arithmetic**: Algebraic operations on elliptic curve groups, including point addition, scalar multiplication, and multi-scalar multiplication.

3. **Polynomial arithmetic**: Operations on univariate and multivariate polynomials over finite fields.

These implementations will serve as building blocks for more advanced cryptographic protocols in subsequent projects. The focus is on correctness and understanding the mathematical foundations rather than optimizing for performance. This project is also designed to introduce you to the Rust programming language.

## 2 Why Rust?

Rust is a general-purpose programming language known for its runtime performance and strong compile time guarantees enforced by its type system – particularly, data-race- and memory-safety. Memory safety bugs in particular are a very common sources of security exploits such as HeartBleed, Stagefright, WannaCry, BlueCry, and many others with unfortunately less catchy names. Indeed, Microsoft found that between 2006 and 2018, around 70% of the security vulnerabilities they fixed via security update were due to memory unsafety. Similarly, Google has cited the use of memory-safe languages (most notably, Rust) as a primary reason the percentage of Android vulnerabilities due to memory unsafety has dropped from 76% in 2019 to 24% in 2024. They additionally note that the "rollback rate" (emergency code reversions) of Rust changes is less than half that of C++.

Of course, no language is a panacea, but these features make Rust a natural choice for new cryptographic implementations, where speed is paramount and the cost of these bugs is dire. As zero-knowledge proofs are a relatively recent technology, it is unsurprising that the vast majority of today's zero-knowledge-proof ecosystem is written in Rust. It is for this reason that we chose to teach this class in Rust, even though we know many of you may be unfamiliar.

## 3 Getting Started with Rust

Installation instructions for Rust can be found here. We highly recommend setting up Rust integration in your editor of choice. Instructions for common editors are available here.

If you're new to Rust, here are some resources and advice to get you started:

- **The Rust Programming Language** (`https://doc.rust-lang.org/book/`): The official Rust book, comprehensive and beginner-friendly
- **The Brown Rust book** (`https://rust-book.cs.brown.edu/`) is a version of the above book from some folks at Brown. It has interactive quizzes to aid understanding and some additional material.
- **Rust by Example** (`https://doc.rust-lang.org/rust-by-example/`): Learn through practical examples
- **Rustlings** (`https://github.com/rust-lang/rustlings`): Interactive exercises to practice Rust concepts

## 4 Getting the starter code

Navigate to `https://github.com/pag-crypto/EECS498598-W26-ZKP` to find the checkout for the starter code. It contains two directories, `src/` and `tests/`. The `tests/` directory contains some tests for the functions you implement, though it does *not* contain the tests we will use to grade your submission. Inside the `src/` directory are the starter code files `zq.rs`, `curve.rs`, and `poly.rs`. You must replace all the `todo!()` blocks with implementations that conform to the behavior described in the comment above the function. The comments also contain some tips and hints. Note that your code will use a multiprecision integer library called `sfs-bigint` that is specific to this course. The code is available at `https://github.com/cmlsharp/sfs-bigint`.

Note that you cannot add any external dependencies that are not present in the starter `Cargo.toml`.

### 4.1 Running tests

To run the local tests, use the `cargo test` command from the root directory of the project. This will run all the tests in the `tests/` directory. To continue even if tests fail, use `cargo test --no-fail-fast`.

To run a specific test, use the `cargo test <test_name>` command. (Note this will run any test that contains the given string as a substring.) Add the `-- --exact` flag to run only tests that exactly match the string. (Note that both pairs of double dashes are needed.)

To see what the tests print, use the `cargo test <test_name> -- --nocapture` command.

For tests that run slowly, you can add the `--release` flag to run them in release mode. This will make some tests run faster, but can cause tests to pass that will fail in the autograder.

### 4.2 Resources

For modular arithmetic, the first four chapters of Shoup [Sho09] will be useful. For elliptic curves, chapter 15 of Boneh-Shoup [BS20] describes point addition and scalar multiplication. For polynomial arithmetic, chapters 2 and 3 of Thaler [Tha22] and chapter 17 of Shoup describe most of the algorithms you will need.

## 5 Submission Instructions and Grading Criteria

To submit your project, perform the following steps from the root directory of the project:

- zip ./submission.zip src/
- submit ./submission.zip to autograder.io

Autograder.io will run our test suite using your submitted code and compute your grade. Your grade will be the percentage of tests that pass. Any modifications you make to the `tests/` directory will not impact how your code is graded. (Feel free to add your own tests locally; this can be a good way to test your own understanding.)

Note that doing something "fishy" to try to get a better grade—for example, overwriting the equality method—will result in failing the assignment.

## References

[BS20] Dan Boneh and Victor Shoup. *A Graduate Course in Applied Cryptography*. Draft version 0.6, 2020.

[Sho09] Victor Shoup. *A Computational Introduction to Number Theory and Algebra*. Cambridge University Press, 2nd edition, 2009.

[Tha22] Justin Thaler. *Proofs, Arguments, and Zero-Knowledge*. Foundations and Trends in Privacy and Security, 2022.