EECE 455 : Cryptography
Project Report – PolyLab
Group 1, Project 2
Fall 2025-2026

Ghada Al Danab
Aya El Hajj
Joud Senan
Roaa Hajj Chehade

Outline of the Report

- **5. Calculator Testing**
- **6. Research Background**
- **7. Work Division**
- **8. Submission and Deliverables**

1. Problem Statement

Our project delivers an interactive web platform that performs arithmetic on polynomials in GF(2^m) for values of m between 2 and 8. It uses the exact irreducible polynomials studied in the course and supports all the core operations: addition, subtraction, multiplication, division, inverse, and modulo reduction.

However, the goal wasn't just to build "another calculator." We wanted something more meaningful and special: a complete system that connects students and instructors, with roles, classrooms, submissions, and secure authentication. The calculator is only one part of the bigger picture. The platform provides a structured environment where professors can create assignments, students can submit their work, and everyone can rely on a consistent GF(2^m) calculator running underneath.

2. Technologies Used & System Architecture

2.1 Frontend Technologies

Our frontend is built using a modern, fast, and component-based stack that keeps the interface responsive and easy to extend. We focused on performance and clean UI design since both the calculator and the classroom system rely heavily on interactive components.

To do that, we used the following:

- **React 18 (with TypeScript)**: We used React 18 with TypeScript to build the user interface in a clean and organized way.
- **Vite**: Vite helped us run and build the project very quickly, making development much smoother.
- **TailwindCSS**: TailwindCSS made it easy to style the website using simple and consistent classes.

2.2 Backend Technologies

Our backend is built around a lightweight and secure architecture that handles all the core logic of the platform, from GF(2) calculations to user authentication, classrooms, and submissions. We chose tools that are fast, reliable, and easy to integrate together, allowing us to build a system that is both efficient and safe for students and instructors to use.

To do that, we used the following:

- **FastAPI**: We used FastAPI to build a fast and structured backend with clean, easy-to-write endpoints.
- **SQLite (development)**: We used SQLite during development because it's simple, portable, and requires no setup.
- **SQLAlchemy**: SQLAlchemy allowed us to manage the database using Python classes instead of raw SQL.
- **Pydantic v2**: Pydantic v2 helped us validate and structure all the data going in and out of the API.
- **Argon2 password hashing**: We used Argon2 to securely hash passwords and protect user accounts.
- **pyotp (TOTP-based MFA)**: pyotp let us add two-factor authentication using time-based one-time passwords.
- **Uvicorn ASGI server**: Uvicorn runs our FastAPI application efficiently and handles all incoming requests.

2.3 System Architecture Overview

Our system uses a straightforward client-server architecture where the frontend and backend communicate through RESTful API endpoints. The frontend sends requests to the backend for everything it needs, whether it's running GF(2) calculations, logging in, uploading a submission, or fetching classroom data.

The backend processes these requests, applies all authentication and security checks, interacts with the database, and then returns back to the frontend a structured JSON response which the interface immediately uses to update the page.

This constant request response flow keeps both sides independent: the frontend focuses

on the user experience, while the backend handles the logic and data. Because of this separation, the communication remains clean and maintainable to extend as the platform grows.

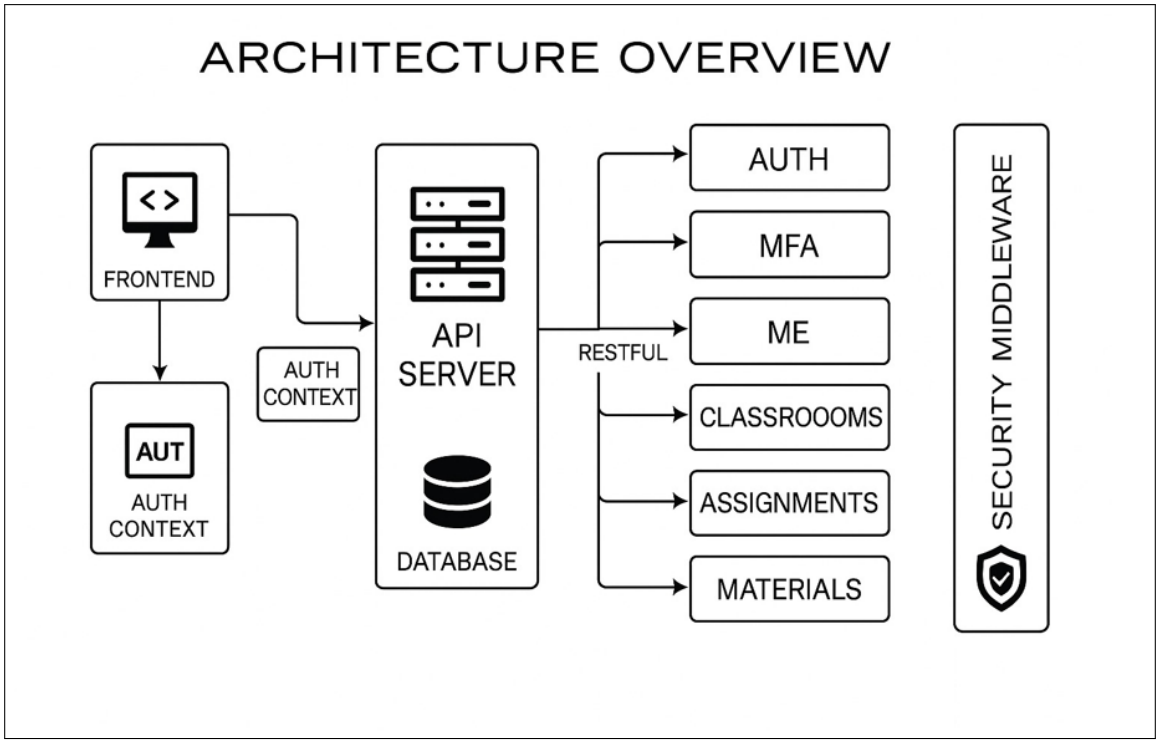

Figure 1: System Architecture

3. Implemented Features

This section describes all fully implemented components of the PolyLab platform, organized by subsystem. Each feature has been developed, tested, and integrated into the production architecture.

3.1 Authentication and Security Features

3.1.1 Secure User Registration and Login

The platform implements a robust, standards-aligned authentication flow:

- Email-based sign-up with both client-side and server-side validation.
- Login using server-issued, hardened session cookies:
  - HttpOnly
  - Secure (in production)

  - SameSite=Lax

- Newly registered users are automatically assigned the student role.
- Session rotation is enforced after login to limit session fixation risks.

3.1.2 Email Verification via Secure Link

To ensure account ownership:

- The backend generates a signed token embedded in a one-time verification URL.
- The link is emailed to the user.
- Clicking the link activates the account and completes the verification process.

This prevents unauthorized access or malicious account creation.

3.1.3 Password Protection

All credentials follow strict security controls:

- Passwords are hashed using Argon2id with salts and memory-hard parameters.
- No plaintext passwords are ever stored or logged.
- Argon2id protects against brute-force and GPU-accelerated attacks.
- Strong password policy is enforced (minimum length and complexity rules).

3.1.4 Multi-Factor Authentication (TOTP)

The system includes optional MFA based on RFC-6238:

- Users can enroll a TOTP device via QR-code provisioning.
- Authentication supports 6-digit time-based codes.
- A pending secret is only activated after successful verification.
- MFA status is stored per user and enforced during login when enabled.

This significantly reduces the risk of password compromise.

3.1.5 Security Middleware

Every request passes through a multi-layer security model:

- CSRF protection using the double-submit cookie pattern.
- Rate limiting on authentication endpoints to mitigate brute-force attacks.
- Security headers, including:
  - Content-Security-Policy (CSP)
  - HTTP Strict-Transport-Security (HSTS)
  - X-Frame-Options
  - X-Content-Type-Options
- Forced session renewal after login and after enabling MFA.

These protections collectively ensure that authentication is resistant to common web security threats such as CSRF, XSS, clickjacking, session fixation, and credential stuffing.

3.2 Classroom and Role Management

The platform includes a fully functional role system and classroom management workflow that supports structured teaching interactions.

3.2.1 Role System and Access Control

- Every user begins as a student by default.
- Users may request to become instructors by submitting:
  - A justification message.
  - A document or proof of eligibility.
- Requests are stored in the database and assigned a Pending status.
- Admin users review requests via the admin dashboard and choose to Approve or Reject.
  - Approval automatically upgrades the user's role to instructor.
  - All changes are logged in the admin audit trail.
- Role-based access control is enforced across the API:

  - Students → standard calculator + assignments + class joining.
  - Instructors → classroom creation, assignment management, and grading.
  - Admins → user management and role changes.

3.2.2 Classroom Creation & Management

Instructors can create and manage their own classrooms:

- Creation includes fields such as classroom name, course code, and description.
- The backend automatically generates a unique join code for each classroom (this code is randomly generated).
- All classrooms are linked to their instructor through relational ownership.
- Instructors can view, update, archive, or manage classrooms through instructor-only routes.

3.2.3 Joining Classrooms

Students can enroll a class in the following way:

1. Students enter a join code on their dashboard.
2. Backend verifies code validity, prevents duplicates, and confirms membership.
3. Successful enrollment creates a classroom membership record.

3.2.4 Classroom Dashboard

Each classroom includes:

- Join code utility (copyable)
- Tabs for:
  - Assignments
  - Uploadable materials
  - Submissions

This subsystem forms the educational backbone of PolyLab by enabling structured teaching and secure assignment distribution.

3.3 Assignments and Submissions

The assignment subsystem supports structured coursework and student submissions within each classroom.

3.3.1 Assignment Creation

Instructors can create assignments integrated with GF(2^m) concepts:

- Supported fields:
  - Title, description, due date.
  - Associated classroom.
  - GF(2^m) sample assignment to select (polynomial addition, inverse, etc.)
  - File or attachment upload.
  - Text or Description.

  Assignments are stored relationally and linked to classrooms.

3.3.2 Student Submissions

Students can submit answers through:

- Direct text submissions.
- File uploads (e.g., handwritten work, PDFs, screenshots).
- Multiple submissions allowed; the backend stores the latest submission.

All submissions include:

- Timestamp (used to determine late vs. on-time).
- Student ID.
- Assignment ID.
- Uploaded file path (if any).
- Grading status.

3.3.3 Grading and Review

Instructors can:

- View all submissions for each assignment.
- Download submitted files.
- Assign grades.
- Track submission status.

3.3.4 Materials & Resources

Instructors may upload additional teaching materials per classroom:

- Lecture notes
- Examples
- Reading materials
- Assignment and solutions (released later)

Access is restricted to enrolled classroom members.

3.4 GF(2^m) Calculator Engine

The platform includes a mathematically verified finite-field arithmetic engine for GF(2^m), supporting m from 2 to 8.

3.4.1 Supported Operations

The calculator implements all core polynomial operations:

- Addition (XOR)
- Subtraction (XOR)
- Multiplication (shift-and-add with polynomial reduction)
- Division (via multiplicative inverse)
- Inverse (Extended Euclidean Algorithm)
- Power (binary exponentiation)
- Modular reduction (modulo the selected irreducible polynomial)

Each operation uses the exact irreducible polynomial defined in the course.

3.4.2 Input and Output Formats

Users can work in:

- Hexadecimal (input & output)
- Binary (automatically padded to m bits) (output only)

The engine accepts values up to 8 hexadecimal bits and performs field-constrained normalization.

3.4.3 Step-by-Step Tracing

To support learning, every algorithm generates detailed internal steps:

- Multiplication steps (Russian peasant method)
- Reduction steps
- Exponentiation steps
- Extended Euclidean iterations for inverse

These traces drive the "Steps" tab in the calculator UI.

4. Deployment

This section describes how the PolyLab platform is deployed and how it can be executed locally for development purposes. Two deployment approaches were implemented: a web deployment using Render, and local manual execution for both backend and frontend components.

4.1 Web Deployment (Render)

The platform is deployed online using Render, which automatically builds and serves both the backend API and the frontend interface. Render monitors changes pushed to the repository and redeploys the services accordingly. This approach provides:

- Automated builds
- Continuous deployment
- Public access to the platform

4.2 Manual Execution (VSCode / Local Environment)

For development, PolyLab can be run locally by launching the backend and frontend separately.

Backend Setup

From the repository root, run:

```
python -m venv .venv
.\.venv\Scripts\Activate.ps1
pip install -r Backend/requirements.txt
```

Then start the backend server using Uvicorn:

```
uvicorn Backend.main:app --reload --host 0.0.0.0 --port 8000
```

Useful endpoints:

- Health check: `GET /health`
- Interactive documentation: `http://127.0.0.1:8000/docs`

Frontend Setup

Navigate to the frontend directory:

```
cd Frontend
npm install
npm run dev -- --host 0.0.0.0 --port 5173
```

The frontend development server now runs locally and connects to the backend API through the configured environment variables.

5. Calculator Testing

This section documents the full verification and validation (V&V) methodology applied to the GF(2^m) calculator engine. The objective was to ensure that every arithmetic operation implemented in the system behaves exactly according to finite-field algebra over GF(2^m) for all supported field sizes ($2 \leq m \leq 8$).

The testing framework combines brute-force evaluation, reference-model comparison, algebraic identity checks, and standard AES test vectors—creating strong mathematical assurance of correctness.

5.1 Testing Strategy Overview

The verification process consisted of four complementary layers:

- **Exhaustive brute-force validation**
  Every possible polynomial value in every supported field was tested against a reference model.
- **Independent "naïve" oracle implementation**
  A clean, definition-based polynomial arithmetic engine was built solely for testing purposes.
- **Algebraic identity validation**
  All core properties of GF(2^m) fields were tested (inverse check, identity elements, group properties, etc.).
- **AES-known vector verification**
  Standard values used in AES (GF(2^8), irreducible polynomial 0x11B) were validated.

Together, these layers establish a complete mathematical correctness guarantee.

5.2 Test Environment

- All tests were written in Vitest.
- A dedicated suite, `gf2m.test.ts`, validates every operation in `gf2m.ts`.
- The suite contains 47 structured tests, many of which internally run thousands of computations.
- Tests run automatically during development and before deployment.

5.3 Independent Naïve Polynomial Oracle

A separate reference engine was implemented to serve as the correctness oracle. This implementation intentionally avoids all optimizations and follows the pure mathematical definition:

- Bitwise polynomial multiplication using polynomial shift-and-add.

- Polynomial long-division for modular reduction.
- Brute-force inverse search: find b such that $a \cdot b = 1$.
- Binary exponentiation built on the naïve multiplication operator.

This oracle is entirely independent from the real implementation, making it ideal for cross-verification.

5.4 Exhaustive Brute-Force Testing

For each field GF(2^m) where $m \in \{2, 3, 4, 5, 6, 7, 8\}$, the test suite evaluated all polynomial elements.

5.4.1 Multiplication Testing

For every field size:

- For every a in $[0, 2^m - 1]$
- For every b in $[0, 2^m - 1]$

We verify correctness by matching implementation output with the oracle model.
This provides full exhaustive correctness for multiplication across all fields.

5.4.2 Modular Reduction Testing

Every polynomial with degree $< 2m$ was tested:

$$gfMod(x) = naïveReduce(x)$$

This ensures the correctness of the reduction step used in multiplication, exponentiation, and inversion.

5.4.3 Inversion Testing

For every non-zero element a:

- Verified that $a \cdot a^{-1} = 1$
- Verified that $(a^{-1})^{-1} = a$

This verifies both numerical correctness and algebraic behavior.

5.4.4 Exponentiation Testing

- Matching $gfPow(a, n)$ with $naïvePow(a, n)$
- Ensuring $a^0 = 1$
- Ensuring group-order identity: $a^{2^m-1} = 1$ for all non-zero a

This confirms correctness of exponentiation and cyclic group structure.

5.5 Algebraic Property Validation

Beyond brute-force validation, the suite tests core field properties.

Addition / Subtraction

- Commutativity
- Identity element $(a + 0 = a)$
- Self-inverse property $(a + a = 0)$

Multiplication

- Commutativity
- Identity element $(a \times 1 = a)$
- Zero annihilator $(a \times 0 = 0)$

Inversion

- $a \cdot a^{-1} = 1$

Exponentiation

Indirect validation of Frobenius property:

$$(a + b)^{2^m} = a^{2^m} + b^{2^m}$$

These identities ensure the implementation respects finite-field structure.

5.6 AES-Standard Vector Validation (GF(2^8))

Using the standard AES irreducible polynomial (0x11B), canonical known values were validated:

Operation	**Expected**	**Verified**
0x57 + 0x13	0x44	Yes
0x57 × 0x13	0xFE	Yes
$(0x57)^2$	0xA5	Yes
0x57 × inv(0x57)	0x01	Yes

5.7 Results Summary

All 47 test cases passed, including:

- Exhaustive multiplication tests across 7 fields
- Exhaustive inversion tests
- Modular reduction checks
- Exponentiation property checks
- AES validation vectors

The results confirm:

- 100% agreement with the oracle implementation
- 0 observed discrepancies in any field
- Full adherence to finite-field algebra

This level of testing effectively provides a formal correctness guarantee for all implemented GF(2^m) operations.

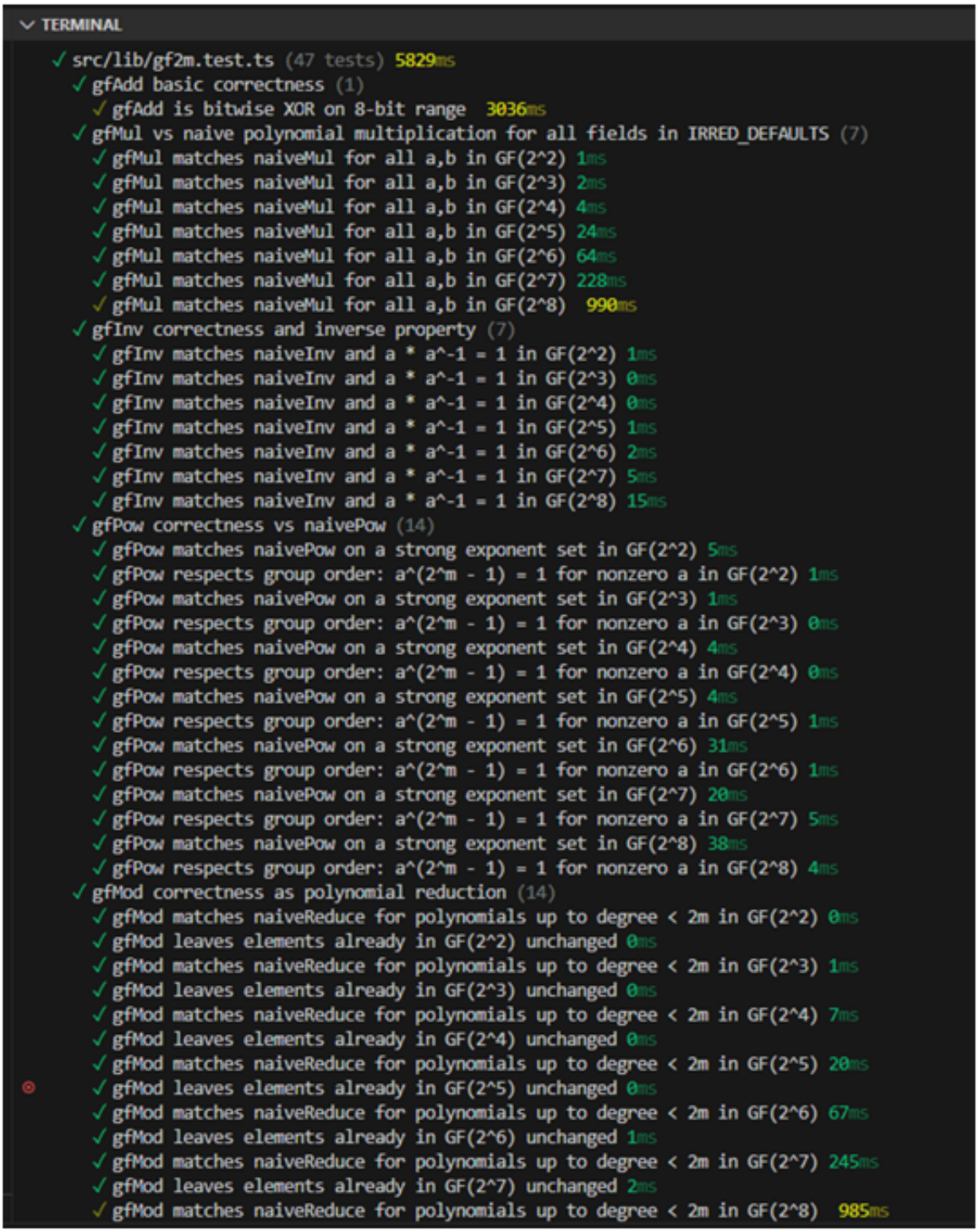

Figure 2: Calculator Test

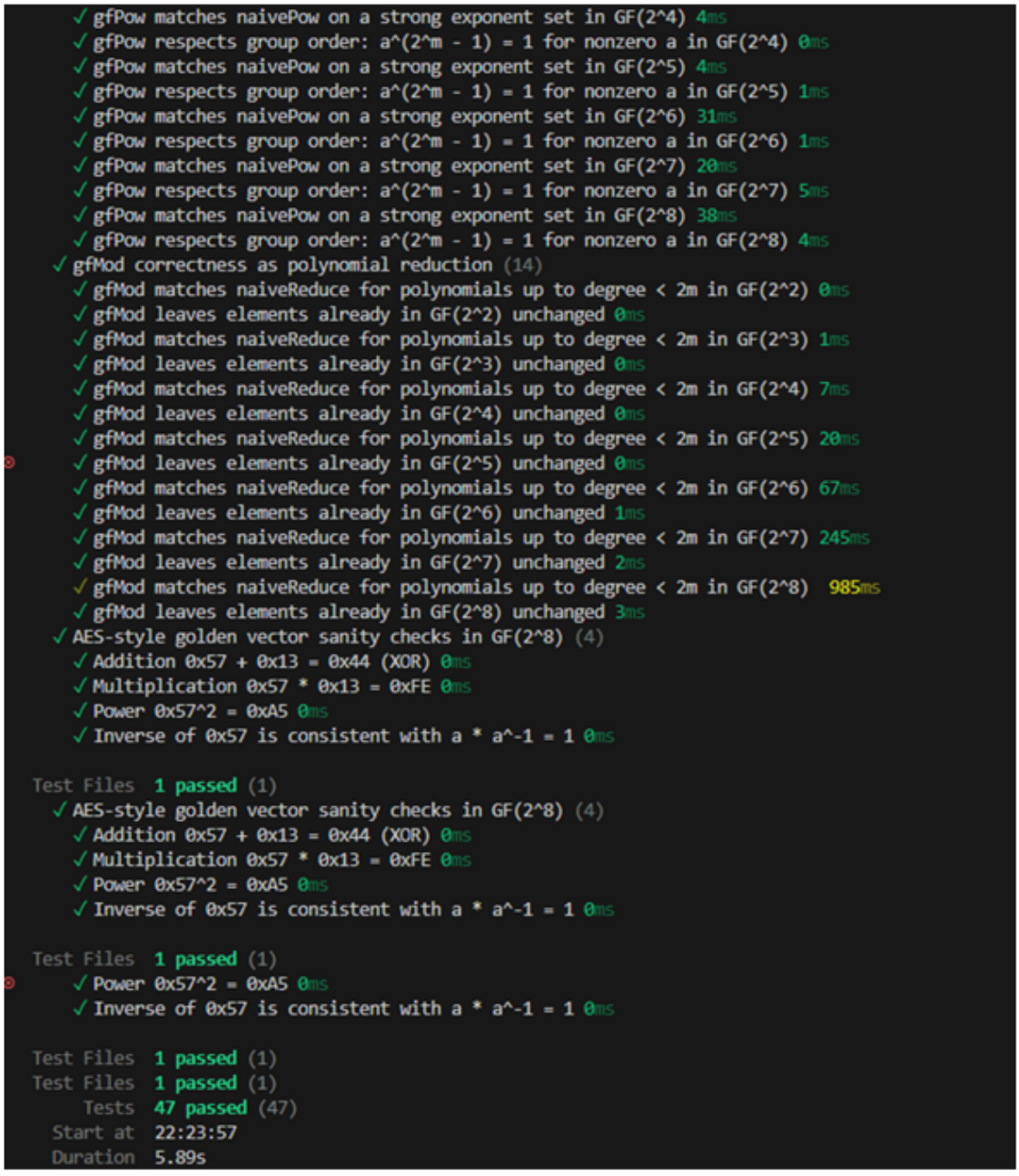

Figure 3: Calculator Test

6. Research Background

General Attack: Birthday Attack

A birthday attack exploits the birthday paradox by generating about $2^{m/2}$ variations of two different messages until a pair produces the same hash, allowing an attacker to replace a signed message with a forged one that carries a valid signature. Because collisions appear so quickly in an m-bit hash, at least 128-bit MACs and larger hash outputs are required to avoid these brute-force collision attacks.

Sources:

- Slides of the course (Chapter 11)

How the Birthday Attack Relates to Our Project (Simple Explanation)

Our project works with small finite fields GF(2^m), which have a limited number of possible values. The birthday attack is based on the idea that when a space is small, it becomes much easier to find two different inputs that give the same output (a collision).

By observing how arithmetic behaves in small fields like GF(2^8), our project helps illustrate why real cryptographic systems must use much larger output spaces (such as 128-bit or 256-bit hashes) in order to avoid collisions and prevent birthday attacks.

Specific Attack: Algebraic Attack

Algebraic attacks try to break a cipher by rewriting its operations as polynomial equations over a finite field and solving them to recover the key. AES is a natural target for this method because its S-box and MixColumns are fully defined in GF(2^8), though no practical algebraic attack has succeeded so far.

Sources:
`Algebraic Aspects of AES (PDF)` (Click on it)

How This Attack Relates to Our Project

Although our project focuses on implementing arithmetic in GF(2^m) (for $2 \leq m \leq 8$), the same mathematical structure is used inside real cryptographic algorithms such as AES.

Algebraic attacks target ciphers by rewriting their operations—especially inversion, multiplication, and modular reduction in GF(2^8)—as systems of polynomial equations.

Since our calculator performs exactly these finite-field operations, it demonstrates the underlying algebra that makes such attacks theoretically possible. While our project is not a cryptanalysis tool, it highlights how finite-field arithmetic forms both:

- the strength of modern block ciphers, and
- the foundation of certain advanced attacks like the algebraic approach against AES.

7. Work Division

The tasks were divided among us four. Each member focused on a specific part of the platform. Below is a concise summary of the contributions.

Joud Senan — Authentication & Security

- Developed the full authentication system: signup, login, logout, and email verification.
- Implemented password hashing, MFA (TOTP), secure cookies, and all required security protections.
- Added the complete role system (student/instructor/admin) and instructor request workflow.
- Dockerized the project and deployed it on Render ensuring that the backend frontend are fully functional online.

Ghada Al Danab — Frontend Interface & Calculator

- Built the main React interface, including authentication pages and user dashboards (admin, student, instructor).
- Implemented the "Request Instructor" feature with file upload and status display.
- Developed the full calculator interface for GF(2^{m}) operations.
- Added UI components, responsive layouts, and ensured that the calculator works smoothly and correctly through testing.

Roaa Hajj Chehade — Core Backend, Classrooms & Admin Tools

- Created the main backend structure and ensured consistent project organization.
- Designed and implemented the database models and schemas for backend operations.
- Built the classroom system, including creation, join codes, and membership handling.
- Added admin endpoints and integrated the backend components into a unified system.

Aya El Hajj — Assignments, Submissions & Backend–Frontend Integration

- Implemented backend logic for assignments, material uploads, and submission handling.
- Ensured consistent backend connection and behavior between instructor and student classroom views.
- Connected the frontend to all backend endpoints for classrooms, assignments, submissions, and grading.
- Added and refined UI features to guarantee smooth interaction with backend APIs.

8. Submission and Deliverables

The final submission for the PolyLab project includes the following main deliverables:

- **GitHub Repository**: containing the full source code for the backend, frontend, and GF(2^{m}) calculator.
  `https://github.com/Joud158/PolyLab.git`
- **Deployed Render Link**: live version of the platform publicly accessible for testing and demonstration.
  `https://polylab-website.onrender.com/`
- **Demo Video**: short walkthrough demonstrating the main features of the platform.
  `https://www.youtube.com/watch?v=tLylCZbrl5U&t=130s`