Final Year Project

AES Adventure

Logan Czernel

Student ID: 19304973

A thesis submitted in part fulfilment of the degree of

BSc. (Hons.) in Computer Science

Supervisor: Dr Félix Balado

UCD School of Computer Science
University College Dublin

May 20, 2023

Table of Contents

1 **Project Specification** . . . 3

1.1 Original Project Description . . . 3

1.2 Full Project Outline . . . 3

1.3 Advanced Goals . . . 4

2 **Introduction** . . . 5

3 **Related Work and Ideas** . . . 6

3.1 Algorithm Visualization . . . 6

3.2 Existing Tools . . . 7

3.3 Summary . . . 11

4 **Outline of Approach** . . . 14

4.1 UI Design . . . 14

4.2 Technologies . . . 14

5 **Project Workplan** . . . 18

5.1 Original . . . 18

5.2 Reality . . . 18

6 **Implementation** . . . 20

6.1 Application Description . . . 20

6.2 General Technical Description . . . 23

6.3 Specific Technical Challenges . . . 25

6.4 Distribution . . . 33

7 **Evaluation** . . . 35

7.1 Lighthouse Audits . . . 35

7.2 User Testing . . . 36

7.3 Comparison with Existing Tools . . . 39

8 **Future Work** . . . 41

9 **Conclusion** . . . 43

Abstract

This paper presents a new educational simulator named *AES Adventure* that guides users through the Advanced Encryption Standard (AES) cipher. The tool assists users in gaining an understanding of the inner-workings and reasoning behind the 3 major components of AES: encryption, decryption, and key expansion. It aims to bring together all the best features present in existing solutions into a single modern and intuitive interface. The tool facilitates users with varying levels of prerequisite knowledge, languages, and additional assistance requirements. External resources are linked where necessary - to cover related concepts that are not taught by this tool. From a small series of user tests, *AES Adventure* scores highly on standard quality in use measures, and issues identified by these tests have been addressed.

Access *AES Adventure* at https://aes-adventure.web.app and the GitLab project at https://csgitlab.ucd.ie/lczernel/aes-educational-simulator.

Chapter 1: Project Specification

1.1 Original Project Description

"The Advanced Encryption Standard (AES) implementing the Rijndael algorithm is the current standard symmetric-key encryption algorithm. It is near universally found in many modern security protocols and applications relying on symmetric-key cryptography. In this project the student will develop a visual and interactive educational simulator of AES encryption and decryption. The tool should allow the user to acquire an in-depth understanding of what goes on at every stage of the algorithm. The student must propose a full project outline, and is free to choose the programming tool(s) to be used."

1.2 Full Project Outline

The project entails designing, implementing, testing and evaluating a new educational tool that assists its users in learning about AES.

- Design: a wire frame of the user interface (UI) must be created that reflects the latest in algorithm visualization research.
- Implementation: the tool itself must be built with the following major components/features:
  - Visual: users can see the input, output, and how the input *transforms* into the output at every stage of the algorithm down to the byte level. Users can view self-explanatory graphs with metrics that describe various aspects of the encryption and decryption processes.
  - Interactive: users can input their own data and interact with the tool every stage of the algorithm. Users should not feel as though they are watching a fixed animation.
  - Comprehensive: users can use the tool to learn about encryption and decryption of all AES key sizes.
  - Accessible: high contrast mode and use of a screen reader is supported, users can access the tool through all modern web browsers (no installation), and use the tool in the language of their choice.
  - Extensible: users can easily contribute language files to extend the tool to support additional languages.
  - Robust: users should not be able to break the tool in any way. Any errors that occur are handled and tracked.
- Tests: the tool should be well tested with "happy path" integration tests at a minimum. The AES implementation should be tested against NIST's Cryptographic Algorithm Validation Program known answer test vectors for AES ECB mode [1].
- Evaluation: the tool should be put in the hands of users and get real feedback from external sources.

The target audience for this tool is Computer Science Undergraduate students, but it should facilitate those that do not have the prerequisite knowledge needed to use the tool in a productive way.

1.3 Advanced Goals

- Students are able to use the tool with their own modified version of AES to further their understanding of the algorithm.
- A dashboard is made available that displays user events to help understand how the tool is most commonly used but also to get insights into what parts of AES users struggle with.

Chapter 2: **Introduction**

Standardized by NIST in 2001 [2], the Advanced Encryption Standard (AES) algorithm is the most popular symmetric block cipher today. It is a highly trusted cipher that is used by both commercial and government organisations to protect data, notably including the US National Security Agency which uses AES to encrypt up to TOP SECRET level classified documents [3]. Due to its ubiquitous nature, many computer scientists from around the world try to learn about AES every year. This project aims to assist those learners to build a solid understanding of the basics, inner-workings, and design motivations of AES.

As a block cipher, AES operates on fixed-size blocks of 128 bits in length. AES supports three different key sizes: 128, 192, and 256 bits. When AES is referenced, they key size is often included, for example "AES-192" refers to AES with 192 bit keys. At a very high level, AES encryption involves performing a series of operations on the input multiple times (in rounds), and decryption is simply encryption but in reverse order (and with inverted operations). Listed below are the aforementioned operations, which act on the *state matrix*; a 4x4 matrix of bytes originally created from the input:

- *SubstituteBytes*: Each byte of the input is substituted with another using a substitution box.
- *ShiftRows*: Bytes are cyclically shifted row-wise.
- *MixColumns*: Bytes are combined column-wise.
- *AddKey*: Bytes are XORed with a round key (see below).

For each round of encryption and decryption, a round key is needed. Key expansion is the critical process that takes place during both encryption and decryption that involves taking the input key and expanding it to create all of the round keys. Key expansion is slightly different for each of the key sizes [4], but it is always a recursive process that uses XOR operations to combine previously created 32-bit words, starting with the input key.

Chapter 3: Related Work and Ideas

3.1 Algorithm Visualization

Algorithm Visualization (AV) technology refers to software that graphically illustrates how algorithms work [5], typically with educational purpose. A brief review of algorithm visualization literature is critical for this project because the best design for the tool is one that is informed by evidence instead of speculation.

3.1.1 Interactivity and Engagement

The strongest trend found in this brief review is that AV technology that engages users yields the most successful educational use [6–8]. A possible quantitative guideline for engagement is measuring roughly how long users are expected to spend between interactions - users should have active participation at least every 45 seconds [9]. The most effective AV tools do not just serve as a medium for knowledge transfer; rather they serve as a catalyst for learning [5].

This interactivity can be achieved in many ways, but one method that came up in multiple studies was allowing users to experiment in "what if" scenarios [5, 8, 10]. In the context of this project, one such scenario could be: "what if we exclude the ShiftRows layer in AES? What effects does that have on the output?".

Amongst the literature there are other suggested methods of encouraging engagement:

- Prediction exercises - users are prompted to guess at the output of a step of the algorithm [5].
- Users can input their own plaintext/ciphertext and key to the tool to see custom outputs [6, 7]. To be discussed further in 3.1.2.
- User is able to interact with cipher analysis tools and metrics [7].

3.1.2 Data Input and Default Values

There are some mixed opinions on the subject of data input ability - some studies suggest that users should be able to input their own data [6, 7], but one older study included in this review concluded that data input ability is not necessary for significant pedagogical value [11].

In relation to providing default values for inputs, research seems to agree that they should be available [6, 11]. Furthermore, more experienced users care more about the default values provided [6].

3.1.3 Execution / Animation Speed

There is convergent evidence that users must be able to control the speed at which the algorithm executes / animations play to massively increase pedagogical value [6, 11]. This includes intermittent breaks in execution for algorithms and playback controls (speed, play, pause, seek) for animations. The aforementioned capabilities also increase the overall interactivity of the tool.

3.1.4 Usage Guidelines

A somewhat surprising focus of a couple of papers is *how* the AV tool is used, but they differ slightly:

Firstly, what users see is not as important as how the tool used for educational effectiveness [5]. This is referring to the learning exercise in which the tool is used. Another paper aligns with this by suggesting that sample exercises, lecture materials, and quizzes should be provided for widespread usage [12].

Secondly, users should be well supported in learning how to use the tool itself [12] to enable users to get the most out of the tool while avoiding user frustration.

3.1.5 Multiple Complexity Levels

Another recurring claim is that AV technologies should support showing/hiding complexity [6, 13]. There are many benefits to this approach, including but not limited to: avoiding overwhelming users with too much information at one time and supporting a wider user base (with greater variation in prerequisite knowledge).

3.1.6 Color and Sound

Lastly, it is worth carefully considering the use of color and sound in AV technology. It may seem almost irrelevant, but one study showed that semantically-chosen colors improve speed on chart reading tasks [14]. While intelligent usage of color and sound is beneficial, one must also be weary of relying on color and/or sound too much to convey key information: many users may be incapable of detecting this information. For example, color vision deficiency (color blindness) affects approximately 1 in 12 men and 1 in 200 women [15].

3.2 Existing Tools

There is a small handful of AES educational simulators available today. A comprehensive review of existing tools is necessary because the best design for the tool is one that is informed by what came before; replicating the best qualities and iterating on or discontinuing the worst qualities.

3.2.1 AESvisual

AESvisual is an AV tool that helps students learn the Advanced Encryption Standard, while also helping instructors teach it [16]. It is available to download on Windows, MacOS and Linux. There are two modes: demo mode and practice mode.

Demo Mode

According to the authors, the purpose of demo mode is for presentations in the classroom. In this mode, there are four sub pages: overview, encryption, decryption and key expansion.

Users can get a high level overview of AES in the overview tab (see Fig. 3.1).

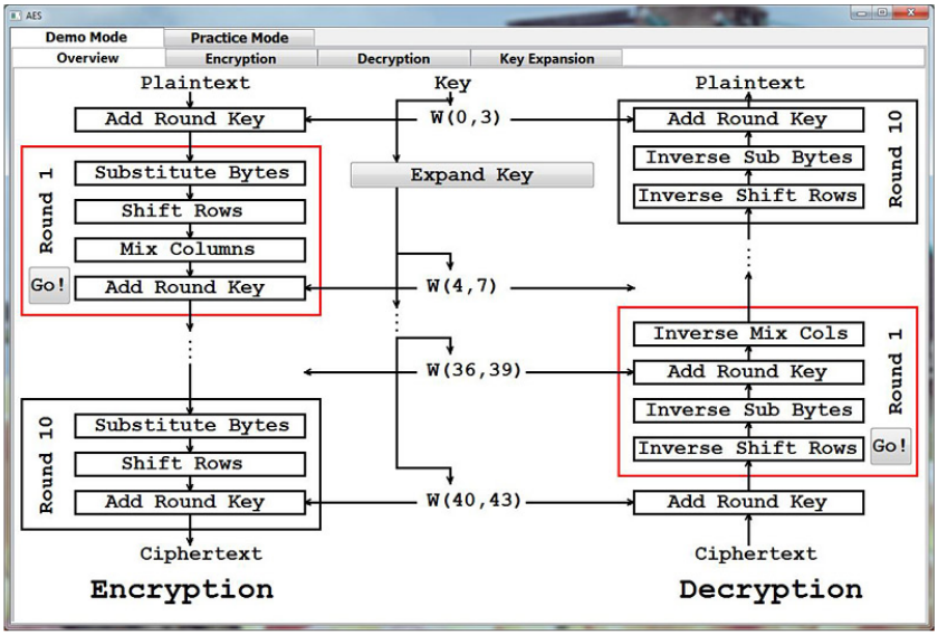

Figure 3.1: AESvisual Demo mode

The encryption tab divides further into sub tabs for each of the layers of AES (see Fig. 3.2). In each of these layer sub tabs, users can inspect how the layer works in isolation. Data input is not supported. The decryption tab is very similar to the encryption tab.

Lastly, the key expansion tab teaches users about how the 128-bit key is expanded to get the 10 round keys required (see Fig. 3.3). This is the only key size supported.

Practice Mode

The intention for practice mode is for the users to be able to test themselves; in this mode, results are hidden and a correct answer is required to continue (see Fig. 3.4). Users are able to skip failed questions. After the encryption is finished and the user has answered all of the questions, a report of the results is made available.

3.2.2 CryptTool-Online: AES

CryptTool-Online offers applications for testing, learning and discovering old and modern cryptography [17]. One such application is a program that walks users through AES step by step.

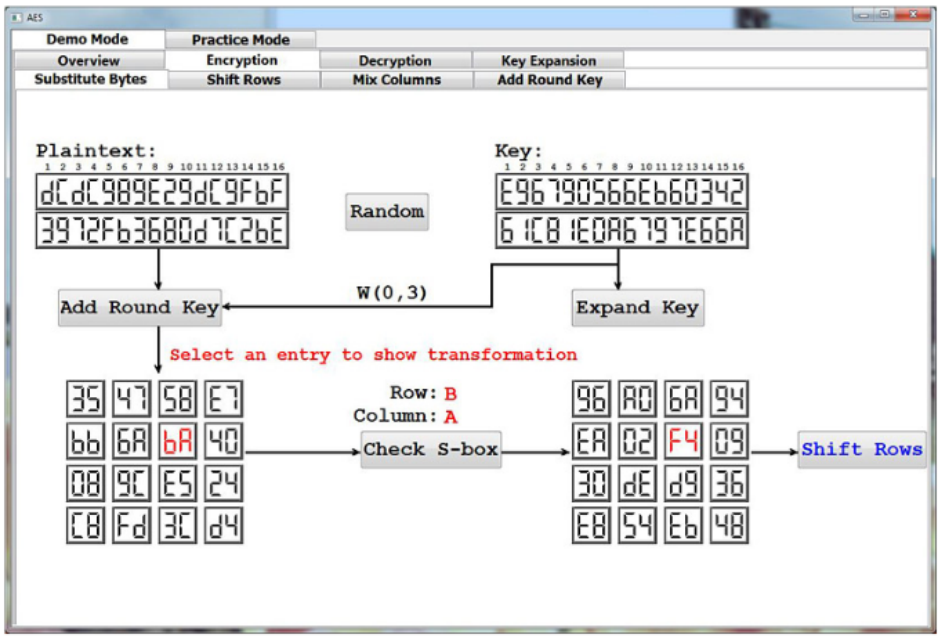

Figure 3.2: AESvisual Demo mode - Encryption

Encryption and decryption of all three key sizes is supported. In addition, it supports encryption and decryption of multiple blocks via chaining with either CBC or ECB modes. The entire tool is contained in one long stack of accordions (see Fig. 3.5).

Available Configuration

This tool is highly configurable, the following is a list of ways that users can experiment with AES:

- Input and key.
- Number of rounds.
- Change the values of the S-Box.
- The IV (CBC only).
- Remove layers for any round(s).

Byte Marking

The tool has a very unique and powerful ability: users can mark a byte and see exactly which bytes it depends on, and how those bytes were combined to form the marked byte (see Fig. 3.6).

Internationalization

The AES section of CryptTool-Online has support for German and English. Since the code is all open source, users are able to submit a pull request with the required translations to support their language.

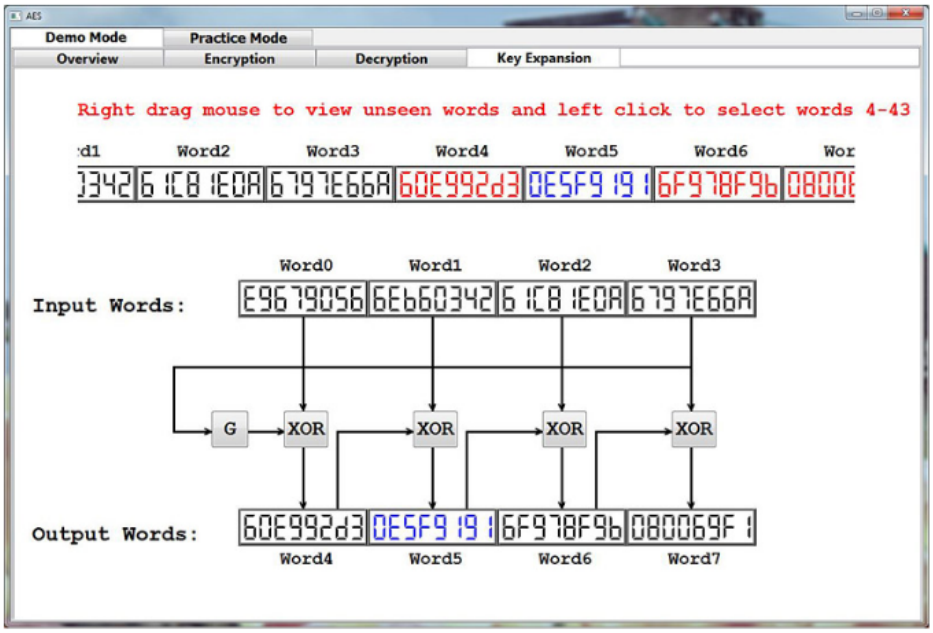

Figure 3.3: AESvisual Demo mode - Key Expansion

3.2.3 AES Visualization

AES Visualization is a tool that allows users to input their own values and watch a 3-dimensional visualization of the algorithm [18].

While I was not able to get the tool running, it is worth mentioning here for completeness (see Fig. 3.7).

3.2.4 AES Visualizer

The step-through visualizer of AES on advancedcomputing.org [19] allows users to input their own plaintext and 128-bit key before expanding the key and showing all intermediate states during encryption (see Fig. 3.8). Decryption and other key sizes are not supported. I could not find the authors of this tool, but it seems that its intended usage is as a calculator rather than a tool for learning [20].

3.2.5 Rijndael Inspector

The Rijndael animation tool shows users how AES works by animating the encryption process of a static input state and key [21]. Every change throughout the process is animated, and users are able to jump to certain steps of the algorithm via a progress bar at the bottom (see Fig. 3.9).

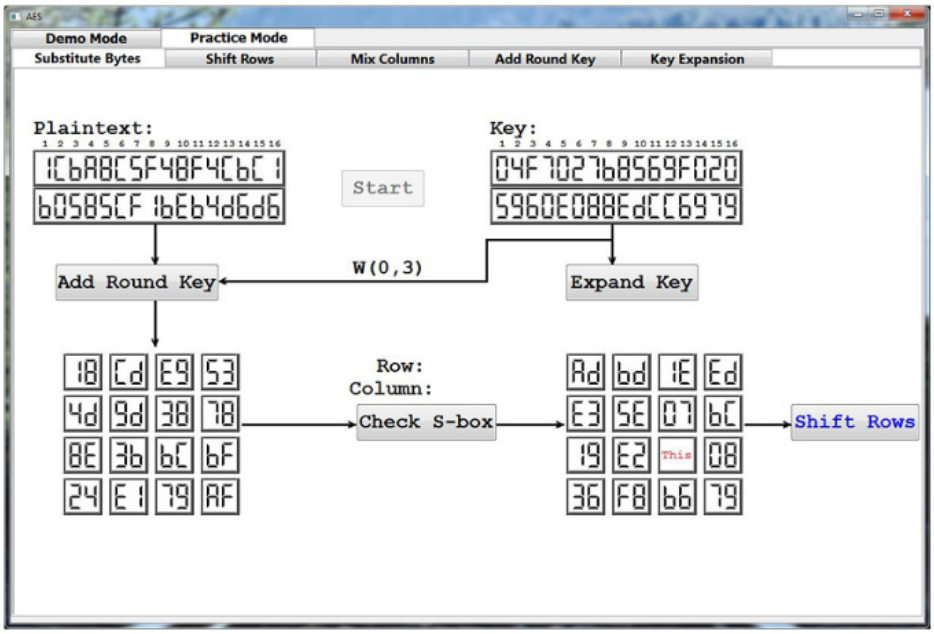

Figure 3.4: AESvisual Practice mode - Substitute Bytes

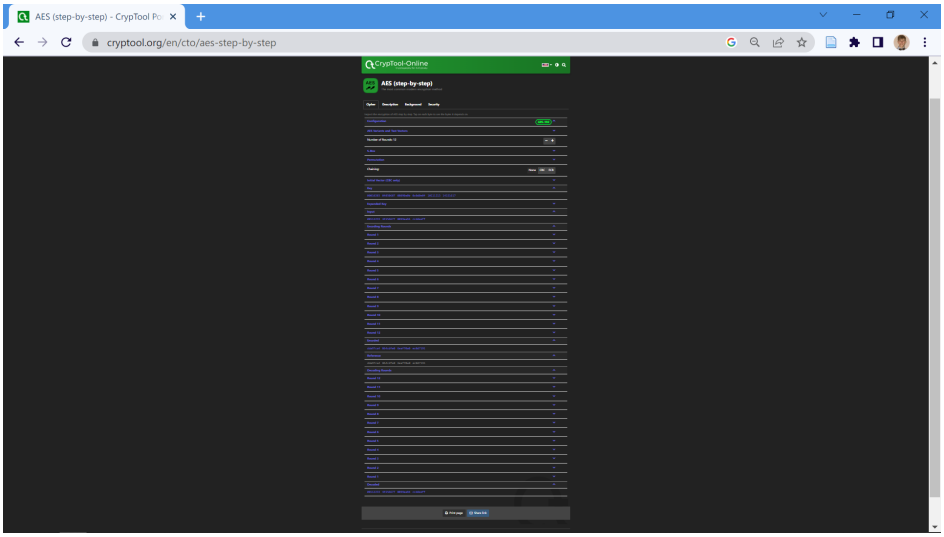

Figure 3.5: CryptTool-Online - 25% zoom showing the layout of AES tool

3.3 Summary

I intend to include both the suggestions from AV researchers and the best features in existing tools in *AES Adventure*. In situations where the research is not in unanimous agreement, my own personal judgement will make the decisions. The existing solutions are all very different; some with various features stand out as desirable, while others are severely lacking. There is no single tool that has all of the good qualities, and many of them have flaws.

Positive qualities that the tool should include:

- Auto-fill input data (with default values).
- Complete control over any animations.
- Supporting website with content that shows users how to use the tool.

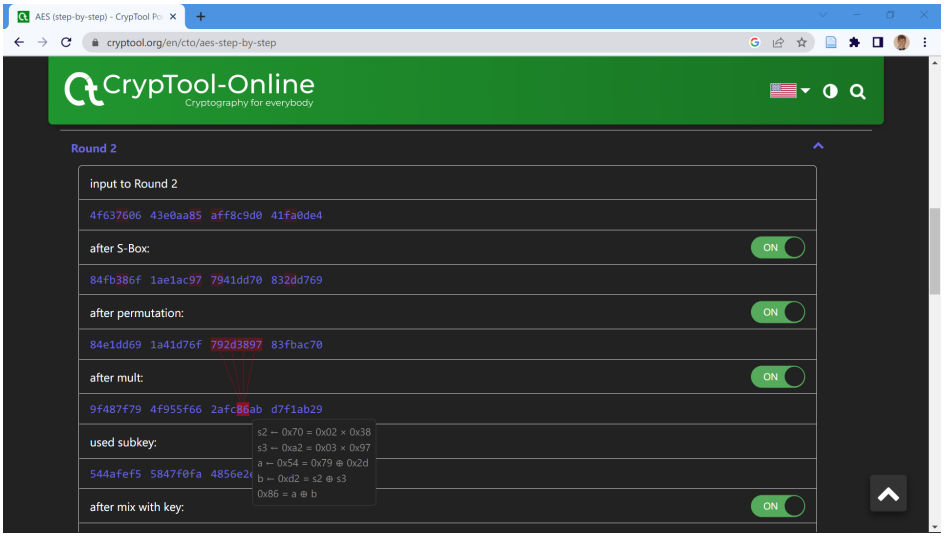

Figure 3.6: CryptTool-Online - Marked byte after MixColumns step

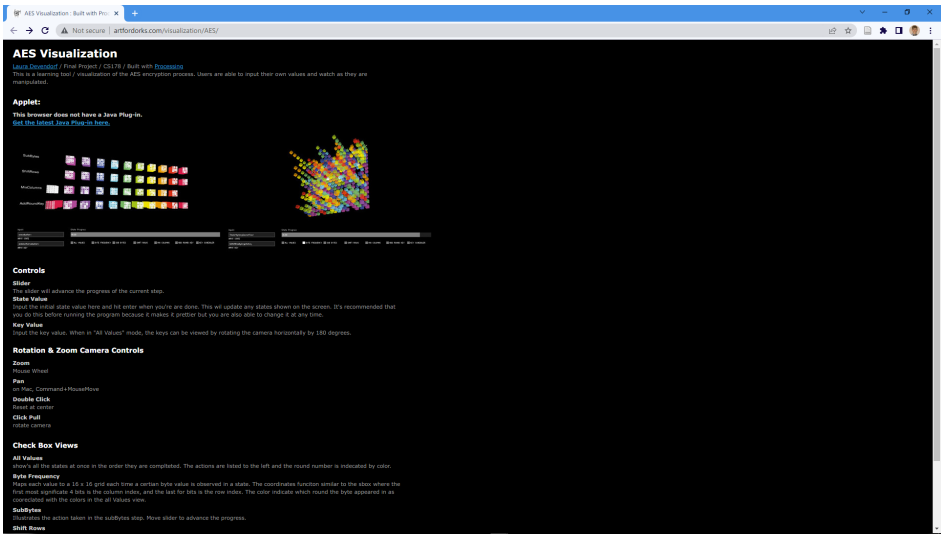

Figure 3.7: AES Visualization - Layout

- Hide complexity by default, but users can reveal it to learn the details.
- Custom data input for plaintext, ciphertext and key.
- Animation: users are shown the input state, output state, and an animation showing how the input state transforms into the output state.
- Configure AES and experiment with their own version of the algorithm, with metrics to assist comprehension.
- Internationalization and accessibility.

Negative qualities that the tool should **not** include:

- Over-reliance on color and sound.
- Poor/outdated design.
- Lacking support for decryption and all key sizes.

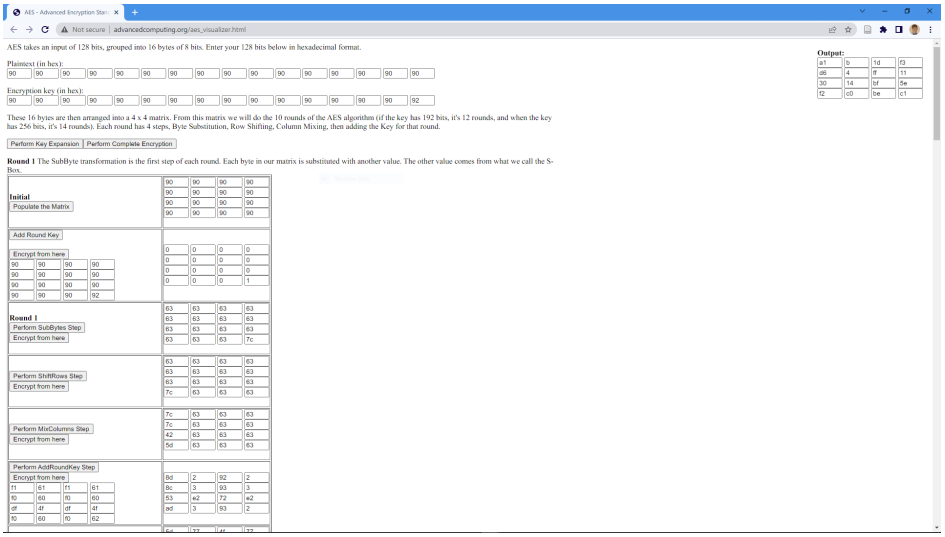

Figure 3.8: AES Visualizer - Encryption

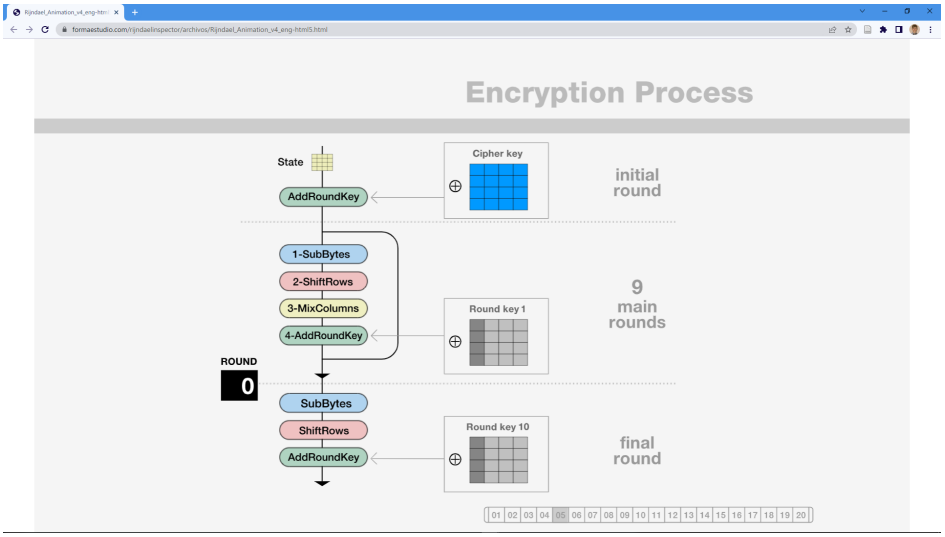

Figure 3.9: Rijndael Inspector - Encryption Overview

- Missing explanations (simply gives input and output).
- No configuration.
- Redundant animations that feel sluggish and overkill.

Chapter 4: Outline of Approach

4.1 UI Design

With all of the findings from Chapter 3 in mind, a UI design for the tool has been created using Whimsical. Whimsical is a powerful tool that enables the creation of low-resolution designs and fast iteration. However, this was a long process - the current design is the 10th version (see Fig. 4.1)! Each time one or more major flaws are discovered, another version is created to address those flaws. Here are some examples of the flaws found throughout the versions:

- Version 1: too difficult to change key size, inconsistent and incomplete UI. *This much was expected from the first version!*
- Version 3: no room for descriptive metrics.
- Version 4: terribly ugly...
- Version 7: missing supporting website, missing initial and final state transformations, no room for a complete explanation of the complex key expansion process (for all key sizes).
- Version 9: inconsistent UI, poor use of space, users need to switch context to get explanations of the algorithm.

You can view a read-only copy of the latest design here.

4.2 Technologies

There are a number of technologies that the tool has been built upon. Given the time constraint, not relying on these technologies was simply not an option.

4.2.1 UI Framework

Ryan Bates once said:

> I'm starting to wonder if there are more client-side JavaScript frameworks than there are apps that use them.

Needless to say, there are *many* frameworks that could be used for this tool. Almost all of them would yield a very similar result for the user that is acceptably fast, however there is one functional requirement, namely, search engine optimization (SEO). The tool should be easily find-able and in the best case scenario, near the top of AES related search results.

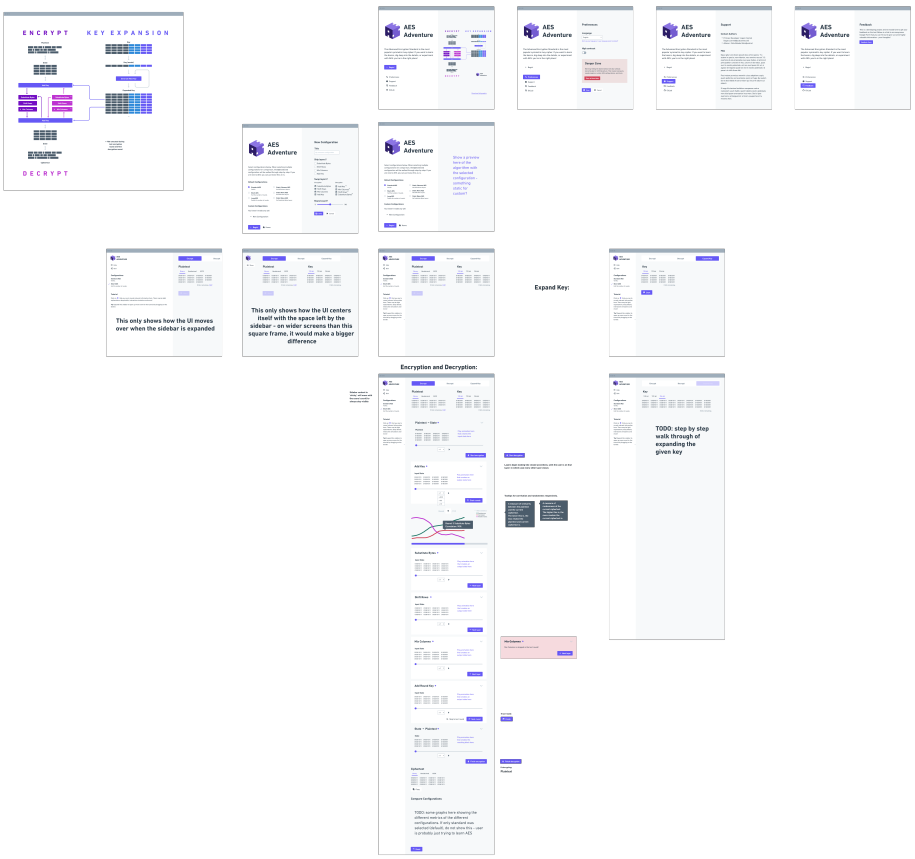

Figure 4.1: Whimsical Design - Version 10 Overview

Therefore, the potential frameworks list was reduced to only the ones that offer good SEO *and* that I was interested in using. The latter requirement removed more candidates than the former because many single page application (SPA) frameworks offer ways to improve SEO, often by facilitating server side rendering (SSR).

Astro was strongly considered:

1. Excellent SEO out-of-the-box: Astro renders as much as possible on the server.
2. Application context: Astro's intended use is for content-focused websites, not applications. This tool will not behave like an application (no login, dynamic data) and will contain lots of static content that can take advantage of Astro's optimizations.
3. Framework agnostic: I can use my favourite UI framework (Vue 3) to build the highly dynamic components of the tool.
4. I want to work with it!

However, there was a lot of friction with packages in Vue 3 ecosystem. It certainly did not feel as advertised, i.e. 'bring your own framework'. Therefore, a fail-fast approach was taken and the project shifted to use Nuxt 3. With many of the same benefits as Astro plus perfect interoperability with Vue, Nuxt 3 proved to be the framework for the job.

4.2.2 Component Library

A strong component library will boost development speed massively. There are many Vue 3 component libraries, all of which fulfill the most common component needs.

Vuestic and Naive UI were considered:

1. Free! Many component libraries have licenses that protect part of or the entire library from non-paying users.
2. TypeScript support.
3. Supports internationalization and accessibility.
4. Unique components for an effortless visual boost.

However, the downfall of these component libraries is in their SSR support, which was lacking. In fairness, at the time of writing, SSR adoption in general is still on the initial rise. In light of this, the project shifted to use Vuetify 3. Vuetify is a massively popular Vue 3 component library and with that came stability and SSR support. While I certainly have my complaints about Vuetify, it was completely functional when used with Nuxt 3's Universal Rendering.

4.2.3 Animation/visualization Library

Following the designs, there are a lot of animations, so it is worth discussing the decision process for selecting an animation library. It started by reading a recent blog post outlining the most popular JavaScript animation libraries [22]. Many of them had faults such as lack of functionality, poor API, or freemium. However, two libraries stood out from the rest: anime.js and popmotion.

Between the two, I settled on anime.js. They both had some great features, however popmotion was *missing* an absolutely critical one: the ability to play/pause animations. Older versions of the library had this feature, but it was removed in the latest major version.

4.2.4 AES Implementation

To build a tool animating AES, an implementation of AES is essential. There are various implementations available on node package manager (NPM), but none of these are suitable because the UI needs more than the input and output: at a bare minimum, it needs the state matrix at every step of the algorithm! Therefore, I created my own implementation, referred to as AESi ('i' for interactive).

Inspiration

AESi is a heavily refactored version of aes-cross, a TypeScript implementation of AES. I also took inspiration from aes-js and Dani Huerta's C implementation.

Testing

It is vital that the implementation is battle-tested and meets NIST's standard. In order to achieve this confidence, AESi is tested against every Known Answer Test (KAT) vector for AES ECB mode

provided by NIST's Cryptographic Algorithm Validation Program [1] (see Fig. 4.2).

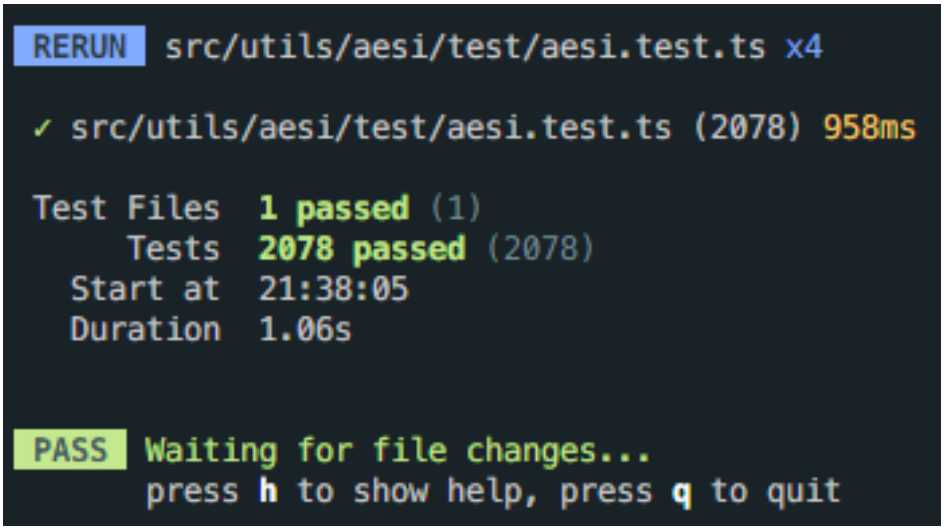

Figure 4.2: Unit tests output

Chapter 5: Project Workplan

5.1 Original

The plan for the project in the coming semester is shown in Fig. 5.1. It was created using onlinegantt.com. This is a very granular plan but there is wiggle room for error - weekends are generally avoided and there are 2 weeks of leeway that can be used to catch up (or get ahead!). As time progresses through the project, I expect development productivity to also increase as I grow further accustomed to the technologies in use.

While development will not stop over the spring break, the work load is lightened considerably for the duration of the break.

Evaluation will likely take the form of contacting test users to convince them to use the tool and fill out the feedback form that can be accessed through the feedback page (see designs in section 4.1). 2 weeks are allocated to evaluation, but due to the nature of the evaluation method for this project, this can go on until the last minute.

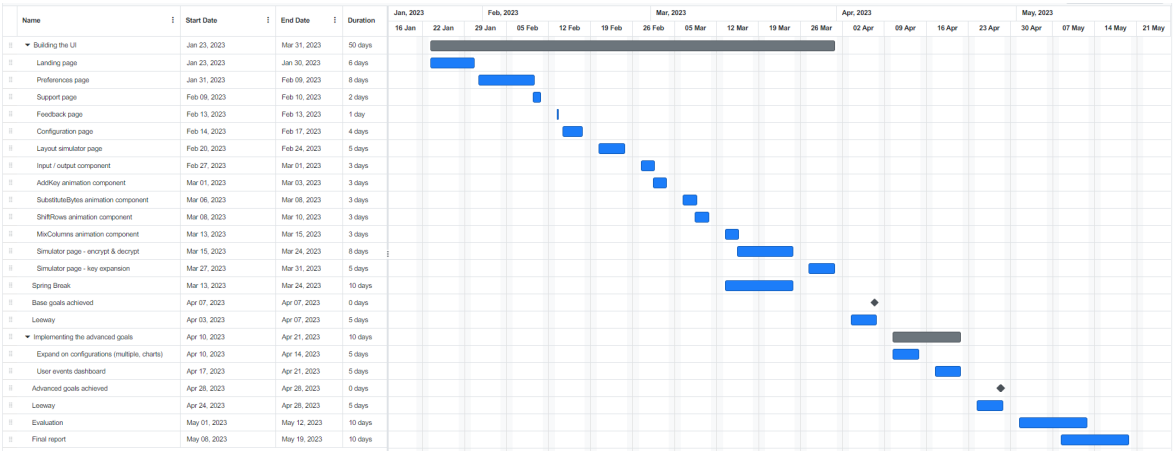

Figure 5.1: Detailed project plan

5.2 Reality

The above plan was surprisingly accurate; the initial pages and time spent building animation components (large portion of the workload) were accurate to within a few days. However, there was not enough time allocated to building 'Key Expansion', so the time that should have been spent on the advanced goals was spent building the third simulator. The leeway weeks proved very helpful, but the idea that no work would be done on weekends was completely wrong. The project was deployed right on schedule which allowed adequate time for the evaluation process.

New code was written almost every week from the end of January through to the start of May (see Fig. 5.2). In addition, a weekly email was sent to the project advisor that detailed the

changes made, provided a brief video demonstrating any visual changes and new features, and often asked questions. In response, the project advisor gave valuable answers, constructive criticism and feedback on the latest changes.

The actual evaluation process used is described in detail in Chapter 7.

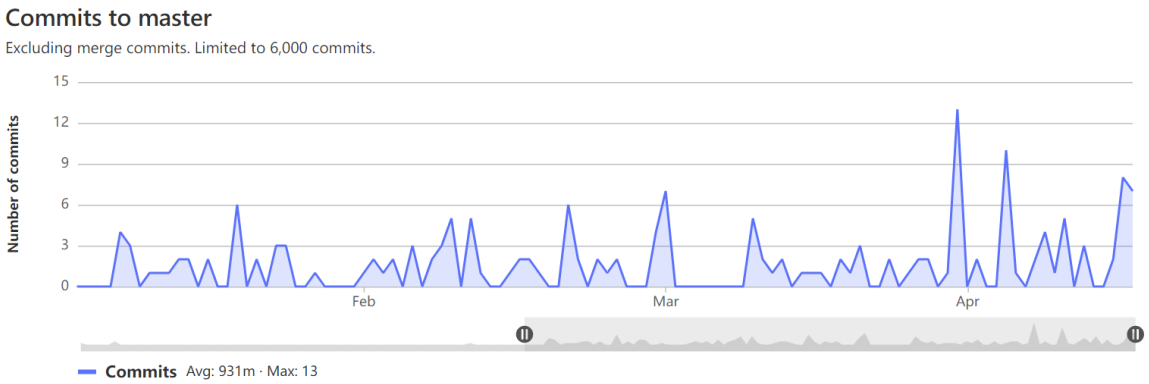

Figure 5.2: GitLab contributions vs time graph

Chapter 6: Implementation

6.1 Application Description

6.1.1 Landing Pages

The landing page is expected to be the first page that users visit. Before entering the heart of *AES Adventure*, they are presented with sub-pages Preferences, Guide, and Feedback, each of which are described in the following subsections.

In addition to these sub-pages, there is a modal that can be used to configure the simulator pages. The modal contains a radio select component in which users can select one of the preset *AES Adventure* configurations.

Guide

The guide page is a great starting-point to new *AES Adventure* users. It outlines what capabilities they can expect from the app as well as tips to help them get the most out of it.

Preferences

The preferences page is a dedicated place for app-wide UI preferences. There are two preferences: app theme and display language. Many languages are supported, such as English, Spanish, Japanese, and Hindi. The app theme can be changed from light mode to dark mode. The default is light mode, but dark mode offers higher contrast in animations.

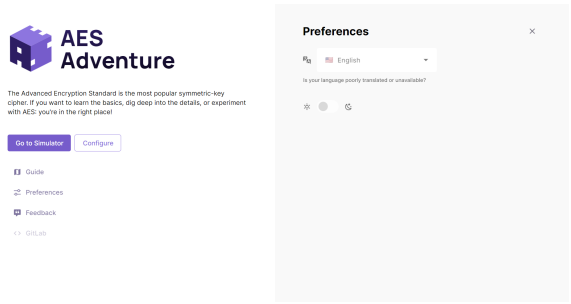

Figure 6.1: AES Adventure - Preferences Page

Feedback

The feedback page helps users express any comments, criticisms, requests, etc. to the creators of the app (myself and the project advisor) through 2 mediums: a structured Google Form (more on this later) and contact information.

6.1.2 Simulator Pages

The three main pages of *AES Adventure* implement three simulators of the primary AES processes outlined in Chapter 2. These pages are described below.

Encryption

The encryption page contains a simulator of AES encryption. Users input a block of plaintext and a key (any of the 3 key sizes), and hit 'Start'. From there, each step of the encryption process is animated and users are guided through the process in the proper order of the algorithm. Each step contains an animation which converts the step input to output. The steps can be opened/closed as they are accordions, but the current step cannot be closed - this is to prevent users from losing track of where they are in the process. After the entire encryption process has completed, the resulting ciphertext is shown. Users can click a single button to copy this ciphertext output into the input field of the 'Decryption' input.

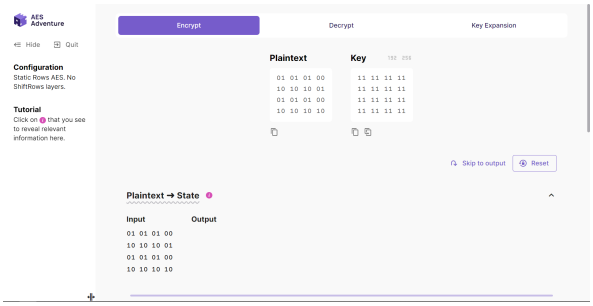

Figure 6.2: AES Adventure - Encryption Page

Decryption

The decryption page contains a simulator of AES decryption. Users input a block of ciphertext and a key (any of the 3 key sizes), and hit 'Start'. Much like encryption, every step of the decryption process is animated and guides users through the algorithm. After the decryption process has completed, the resulting plaintext is shown. Users can easily copy this to the 'Encryption' input with the click of a button.

Key Expansion

The key expansion page contains a simulator of AES key expansion. Users input a key (any of the 3 key sizes), and hit 'Start'. Much like the other two simulators, the key expansion process is broken down into animated steps. Once again, *AES Adventure* tries to reuse the experience that users may have from using the other simulators. The concept of *rounds* (and all interactive abilities when navigating between them) and a *state* is seen in all three simulators.

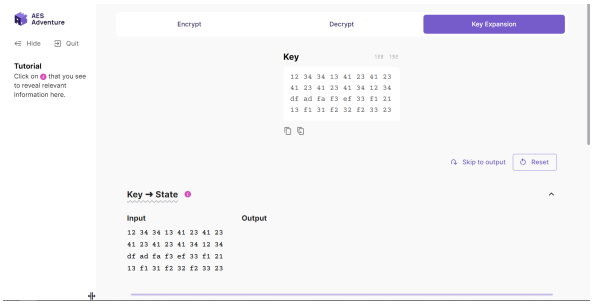

Figure 6.3: AES Adventure - Key Expansion Page

6.1.3 Usage

AES Adventure accommodates users of varying levels of prerequisite knowledge. Information can be hidden and revealed for both beginners and more advanced users.

A note on learnability: *AES Adventure* has a focus on consistency such that once users learn how to do certain actions in one part of the app, that knowledge is reusable in other parts of the app. The symmetry between the encryption and decryption simulators is a glaring example of this; nearly every ability learned in encryption is reusable in decryption. This is also true (but slightly less so as the processes are *very* different) with key expansion.

Beginner

Users that are new to AES are expected to move slowly through the simulator. There are various features in place to facilitate these users:

- Tutorials: throughout the simulator are small tutorial icon buttons. When one is clicked, a tutorial is shown in the sidebar corresponding to the concept that the tutorial icon button is beside. For example, the Substitute Bytes step has a tutorial icon button beside the title of the step. When clicked, a tutorial appears in the sidebar that explains what Substitute Bytes is doing, why it does that, and links to external resources to learn more.
- Playback speed controls: beginners can half the speed at which animations play if needed.
- Dynamic guidance: when viewing an animation, there are buttons that can be clicked to move on to the next step. The style of these buttons is dynamic in that it depends on the corresponding animation. They are clickable, but not highlighted, while the animation is playing something that **beginners should watch**, and they should not skip. For example, Substitute Bytes is quite repetitive and long, so the buttons become highlighted after the first few bytes have been operated on - the next minute plus of animation follows the same process. On the other hand, the Round G-Fn in key expansion should be viewed until the very end by beginners, and the buttons reflect this!

Intermediate

Users that are roughly familiar with AES are expected to rely less on tutorials and start exploring the statistics that are made available: the confusion and diffusion graph (see Fig. 6.4 and 6.3.6). They may use extreme key values (e.g. all 0s) or compare 128 and 256 bit key sizes.

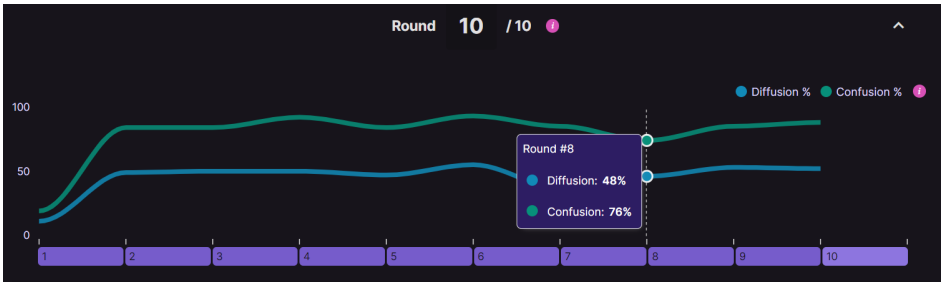

Figure 6.4: Encryption - Confusion and Diffusion graph (dark mode)

Advanced

Advanced users are expected to utilize everything that intermediate users do plus the available configurations. *AES Adventure* has 6 configurations that affect the encryption and decryption simulators (see Fig. 6.5). Only one configuration can be selected at a time. A great example of an expected advanced user path is:

1. For a given plaintext p and 128-bit key k, discover the change of diffusion as the rounds of encryption are completed.
2. Configure the simulator to remove one of the major diffusive elements of AES, e.g. ShiftRows.
3. Using p and k again, discover the impact that removing a diffusive layer from the encryption process had on the diffusion as the rounds of encryption are completed.

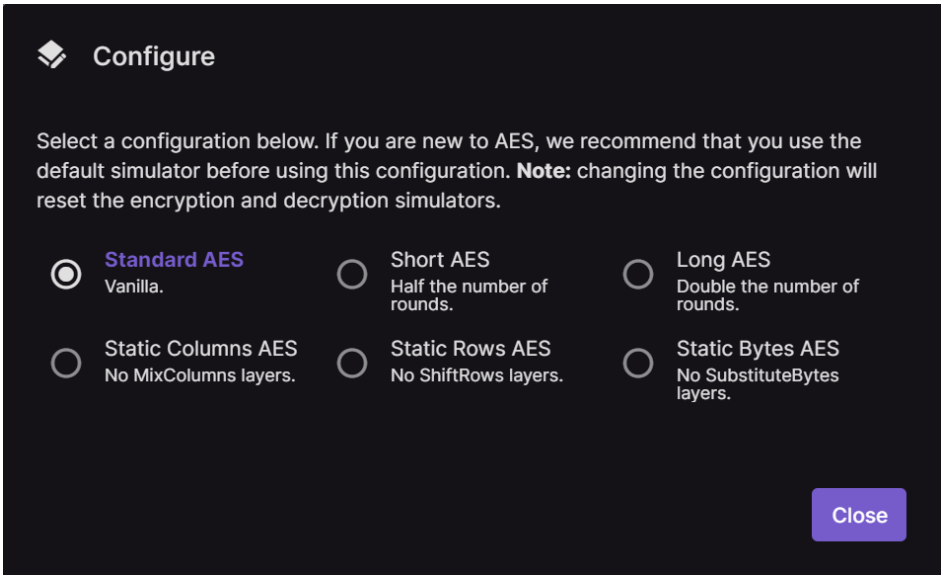

Figure 6.5: Available configurations (dark mode)

6.1.4 Limitations

The minimum display width for *AES Adventure* is its main limitation. Users can expect a poor experience on screen sizes smaller than standard 13" laptop screens. There are workarounds for screens that are slightly smaller:

1. Zoom out using browser zoom capabilities present in all major browsers.
2. All *AES Adventure* animations can scroll when their minimum widths are broken, and the simulator sidebar can be hidden entirely to make more space available.

On mobile devices with small screens such as smartphones, there is even less support. Sections of the app are generally scroll-able, but the power of *AES Adventure* as a learning tool suffers greatly. By rotating the device, users can maximise the width which improves the experience. While this certainly is a limitation, it is worth noting that *AES Adventure* is designed to be used during study; which is most commonly done through a laptop [23].

AES Adventure has great browser support, working flawlessly on all modern browsers. However, this does not include Internet Explorer.

The final limitation is robustness. None of the UI components are tested through unit tests, nor is any user path (e.g. happy path) through integration tests.

6.2 General Technical Description

6.2.1 Dependencies

AES Adventure is written in TypeScript for its obvious advantage over JavaScript: the type safety. Instead of Cascading Style Sheets (CSS), all style-related code is written in Syntactically Awesome Style Sheets (SASS) for all of its rich features that it provides on top of CSS. As outlined earlier,

Nuxt 3 is the framework of choice. Other notable dependencies that have not been discussed already include:

- Pinia and pinia-plugin-persistedstate: global state stores that are persisted in `localStorage`.
- ApexCharts.js: charting library.

6.2.2 Layouts

Nuxt 3 has a concept of page layouts - these are not served at any route, rather they are foundations that pages are built upon. *AES Adventure* has 2 page layouts: `LandingPage` and `SimulatorPage` . The former is used to build the index, guide, preferences, and feedback pages. The latter is used to build the encryption, decryption and key expansion pages. Despite being completely different HTML files (*AES Adventure* is a Multi-Page Application or MPA!), Nuxt allows you to define custom Vue 3 page transition animations between pages in the same layout (e.g. guide to preferences) and between different layouts. This gives the smooth routing feel of a Single-Page Application (SPA) but retains all the SEO benefits of a MPA.

6.2.3 Pages

Each *page* of *AES Adventure* has its own HTML file. All pages except for the error page are wrapped by one of the two layouts. The error page (see Fig. 6.6) is shown when an app-wide error occurs in order to show something presentable and on-brand for the user, as well as an easy method of returning to the index page.

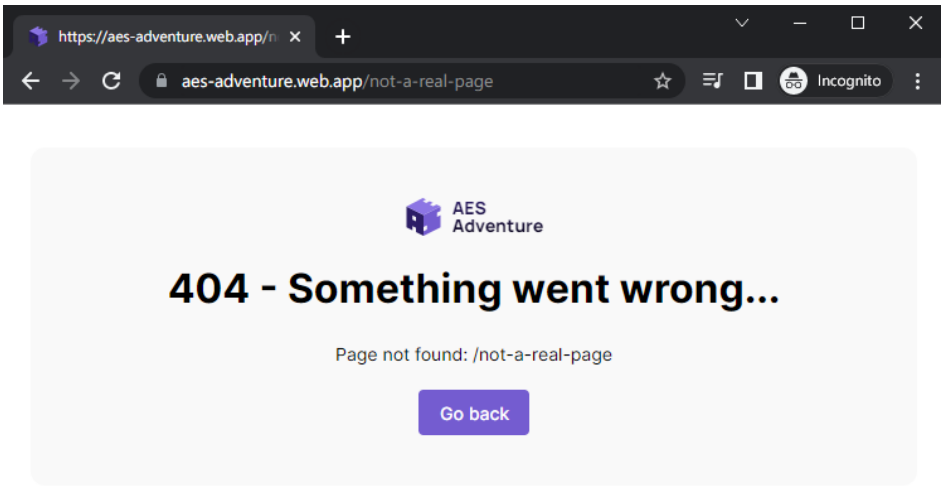

Figure 6.6: Error page

6.2.4 Components

Each component is a Vue single-file component, meaning the component file contains the template, style (SCSS) and JavaScript (TypeScript) that are needed for the component. Components that must be rendered on the client side are wrapped in Nuxt 3's builtin `ClientOnly` component. On the server, a fallback is rendered: a skeleton version of the component. This can be seen in the `LanguageSelect` component.

6.2.5 Composables

Composables holds functions that can be used to hook into reactive shared state. For example, `useEncryptState` holds the state of the encryption simulator. In addition to the state variables, composables expose methods to mutate the state. For example, `useSidebar` holds the state of the sidebar component and exposes the method `toggle` which can be used to toggle the visibility of the sidebar (it is 'closable').

This is the layer at which the dependencies Pinia and pinia-plugin-persistedstate act. Pinia is used to create these Vue 3 reactive global state stores, while the plugin automatically persists the state to `localStorage`.

6.2.6 Stylesheets

The `assets` folder contains all of the global style code which is imported into `app.vue`. These stylesheets provide heavily reused classes as well as overrides, some examples of these files are:

- `vuetify-overrides.scss`: overrides many different Vuetify styles. In my opinion, Vuetify has many senseless defaults and looks awful.
- `animation-grid.scss`: defines classes such as `animationGrid`, `animationGridSbox`, and `animationGridKey`. These classes are reused *many* times throughout all three simulators to arrange the text nodes into grids of various dimensions.
- `math-font.scss`: defines `math` class which can be used for special math related glyphs.

6.3 Specific Technical Challenges

6.3.1 Animation

Animation is the centrepiece of every algorithm 'step' presented in *AES Adventure*. There were a number of challenges in creating these animations and updating them efficiently.

Animation Frame

It all starts with the `AesAnimationFrame` (see Lst. 6.1) - a very powerful component that manages the anime.js Timeline instance for each animation. This Timeline instance is used to control the position and playback of the animation. The animation frame also provides common functionality such as the timeline and playback speed controls. All animations are wrapped in an animation frame.

```
<AesAnimationFrame>
  <template #animation="{ timeline }">
    <!-- Animation component -->
  </template>
  <template
    #appendControls="{ timeline, restartAndPause, play, pause }"
  >
    <!-- Bottom right control -->
  </template>
</AesAnimationFrame>
```

Listing 6.1: AesAnimationFrame API

Animation Components

All animations require the Timeline instance created by the wrapping `AesAnimationFrame` as a prop, in addition to any animation specific props. For example, `AesSubstituteBytes` animation also needs the substitution box, input and output numbers. Once mounted in the Document Object Model (DOM), an animation will *add* animation steps to the Timeline instance. Animation with anime.js involves targeting elements (by class, ID, or other methods) and specifying what to do with those elements. For example, in plain English: all `<div>` elements with the class "test-123" should be translated right by 100 pixels over the course of 2 seconds. It is the responsibility of each animation component to mount their input to the DOM with scoped classes - class names that won't interfere with other instances of the same animation on the same page - and *build* the animation as it was just described.

Animation Utilities

The animation components rely on a number of utility functions, the most notable of which are `hexToDivs`, `addAnimationClasses`, and `updateDivs`.

- `hexToDivs`: converts a list of numbers represented in hex to a list of `<div>`s. Each `<div>` contains two child `<div>`s, one for each nibble of the hex value. Animations convert their hex values into DOM elements using this method.

```
export const hexToDivs = (hex: Array<number> | Uint8Array)
   => Array<HTMLDivElement>
```

- `addAnimationClasses`: adds special animation classes to each input `<div>`, as well as each input `<div>`'s children, *and* each input `<div>`'s grandchildren. Animations add the target classes to their DOM elements in order to build their anime.js animations. `addAnimationClasses` returns a series of helper functions that can be used to target groups of elements or individual elements. It assumes that the elements it is adding animation classes to are arranged in a 2D grid, therefore some example helper functions it returns are `targetRowClass`, `targetColumnClass`, and `targetCoordsClass`. A simple and clear example of how these would be used is in the `AesMixColumns` animation component - which uses `targetColumnClass` in order to translate entire columns of the input state matrix at a time.

  It is also worth noting that, although `addAnimationClasses` and `hexToDivs` are often used together for each group of elements that must be animated, every animation calls `hexToDivs` more than `addAnimationClasses`. The inputs to every animation are rendered twice - one copy has the animation classes added to them, while the other remains static. This is the

case because whenever the input is manipulated in an animation, the **original** input is always still visible - the static copy of the input elements.

```
export const addAnimationClasses = (divs: Array<
    HTMLDivElement>, id: string, rowCount = 4) => ...
```

- `updateDivs`: similar to `hexToDivs`, but sets the `textContent` of each child `<div>` (modifies in-place), replacing the value with the corresponding value in the input hex. This method is **critical** to a smooth experience when moving between AES rounds because the animations update their `<div>`s rather than unmounting and recreating everything from scratch. More on this later.

```
export const updateDivs = (divs: Array<HTMLDivElement>,
    newHex: Array<number> | Uint8Array) => void
```

There are also helper functions for dealing with words of hex values rather than bytes, but at the core of each of these functions is a call to one of the functions listed above. The key schedule of AES is word oriented, so the related animations rely on the word oriented animation utilities.

Efficiently Updating Animations

It was noted early on that rebuilding animations entirely when moving between rounds of AES was too slow. There was visible jank with long blocking times (around 1.5 seconds). The root cause was all of the DOM operations that had to take place when unmounting four animation components (one for each step of the round) and then immediately mounting four updated animation components. Note that the DOM structures between these updates were identical: the only difference is the `textContent` of the leaf nodes. To alleviate this issue, the following refactor was made:

1. `AesAnimationFrame` accepts a `timelineKey` prop - when this value changes, the animation frame scraps the current Timeline instance and creates a fresh instance. This new instance no longer has the previous animation steps, so the animation components must also be aware of this change.
2. Animation components watch for the Timeline instance to change, and when it does, they invoke `updateDivs` and re-add their animations to the new Timeline instance. This effectively updates the animation to new values **without** all of the expensive DOM operations.

These changes almost halved the blocking time, but there is still room for improvement.

Lazily Rendering Animations

After the encryption page was built, it was noticed that navigating to this completed simulator resulted in visible jank. All of the animations are wrapped in a `StepDropdown`: a custom dropdown component built on Vuetify's dropdown component, which has an `eager` prop. If this is `true`, then the content of the dropdown will be rendered while the dropdown is closed. Each animation takes a small amount of time to render, so they were previously all using `eager = true` to avoid jank when moving between animations. As a consequence, when navigating to the encryption page, they all rendered at once, causing the jank. Another large refactor had to take place in order to mitigate this issue:

1. `StepDropdown` defaults `eager = false`, but watches for `mouseenter` events on the dropdown title, setting `eager = true` when this event is fired. Users trying to open a dropdown will click the dropdown title, and the time between users mouse hovering over the title and users clicking the title is enough for the dropdown content (the animation) to be rendered. This works well for users manually opening the dropdowns, but there is one other scenario that had to be handled.
2. As users move through a simulator's steps, pressing the 'Next step' button closes the current step dropdown and opens the next step dropdown. Therefore, each step overrides the default `eager = true` only when it is the **next** step in the simulator.

This refactor retained the smooth dropdown openings as if they were not rendered lazily, while reducing the visible jank when moving between simulators.

6.3.2 Tutorials

Tutorials are found throughout the entire app - they can be hidden and ignored, or relied on in conjunction with the animations to learn about AES. The primary technical challenge in the implementation of the tutorials was to create a type-safe way for any component in the app to render a desired tutorial component in the `SimulatorPage` side panel.

Tutorial Keys

An `emum` stores all possible tutorial keys - special strings that are associated with one tutorial component. When a component anywhere in the app wants to render a tutorial component in the sidebar, they refer to that component by its corresponding tutorial key.

Tutorial Composable

The `useTutorial` composable can be invoked in any component. It exposes methods such as `open` and `close`. The composable has access to a 1-1 mapping between tutorial keys and the corresponding components. `open` requires a `TutorialKey` parameter and sets the `currentTutorialComponent` value to the corresponding component. In order to mount the tutorial to the DOM, we leverage the builtin Vue dynamic component:

```
1 <component :is="tutorial.currentTutorialComponent" />
```

6.3.3 Serialisation

Almost the entire app state is serialised and stored in `localStorage`. Any change to app state results in a write to `localStorage`. On render, this state is read from `localStorage` and deserialised back to the JS state objects. All of this takes place at the composables layer, which hold the app state.

There was a bug that was difficult to fix: the `useSidebar` composable would throw errors due to an access of a `not defined` variable. Readers more familiar with JavaScript will know that is **not** the same as a variable being `undefined`... It was difficult to identify exactly why this was happening because, even with the guards of TypeScript, that variable *was* both declared and

defined. In addition, it would only happen when reloading a simulator page, but not by navigating onto a simulator page. The pinia-plugin-persistedstate library automatically serialises Vue 3 refs, and the variable in question was *not* a ref... however, this strange behaviour was a issue caused by the plugin, and the discovered workaround was to use a singleton access pattern to this variable (see Lst. 6.2).

```
let _resizeObserver: ResizeObserver
const getResizeObserver = () => {
  _resizeObserver ??= new ResizeObserver(onManualResize)

  return _resizeObserver
}
```

Listing 6.2: Begone with this terrible bug!

There was one additional challenge when serialising the app state, which was converting `Uint8Array` objects to JSON. This is a very common object used in *AES Adventure* - it is an array of hex values! By default, when `JSON.stringify` is invoked on a `Uint8Array`, it creates a mapping between index and value before converting this to a string. When it came time to parse this variable from `localStorage`, pinia-plugin-persistedstate was unable to re-create the `Uint8Array` and there was a type mismatch between compile time and run time. This means that, while programming, it was dealt with as a `Uint8Array`, but after deserialisation (while code is running), it was actually of type `object`. To fix this, the `Uint8Array.prototype.toJSON` method was implemented to return just the array of values instead of an object (see Lst. 6.3). This implementation is automatically used by `JSON.stringify` once defined.

```
function toJSON(this: Uint8Array) {
  return Object.values(this)
}

; (Uint8Array.prototype as any).toJSON = toJSON
```

Listing 6.3: Overriding Uint8Array serialisation

6.3.4 Internationalization

One of the primary goals of this project was to make this tool accessible to far more people than just those that can speak English. That has been achieved; *AES Adventure* supports 30 languages. Disclaimer: these translations are machine translations and the app will not look exactly the same in all languages because of the differences in length of certain phrases.

One of the earliest commits to this project was setting up a Nuxt 3 internationalization module: @nuxtjs/i18n. From there, every user facing string in the entire app is placed into the `en.json` file and read dynamically by a function provided by the module called `t`. At the very end of development, it came time to convert `en.json` into other languages. This proved to be the hardest part: this is very expensive! Immediately, I learned that human translations were out of the question. There are also challenges for machine translation: it must deal with the nested HTML tags in certain strings, unique glyphs, and placeholder strings.

The first attempt was done with https://translate.i18next.com. It was one of the rare machine translation services that was free. Everything looked great at first, until I inspected the translated files further. Many of the resulting strings were cut after the first or second sentence.

The next attempt was with jsontt, another free machine translation tool. This tool would inconsistently fail at translating many keys, to the point of it being unusable.

The final attempt was with Lokalise (see Fig. 6.7). This is a paid service with their starter plan costing $120 per month and that does not even include the translations! However, their 14-day free trial was exploited and a project for *AES Adventure* was setup. This turned out to be an extremely powerful and sleek tool for managing and ordering translations at scale. After uploading `en.json` and a test conversion to Spanish, everything looked perfect and it even caught some grammar and spelling errors that were previously missed. There are thousands of words in `en.json`, so converting one language via Google Translate costed $2.25. *AES Adventure* supports 30 languages in total, therefore 29 translations were required which in total cost $65.25. Any changes in the future will have to be done manually (using translation software and individually updating translation keys).

The end result is a language select component in the preferences page and the app route is prefixed with the locale code (great for SEO, more on that in the next section).

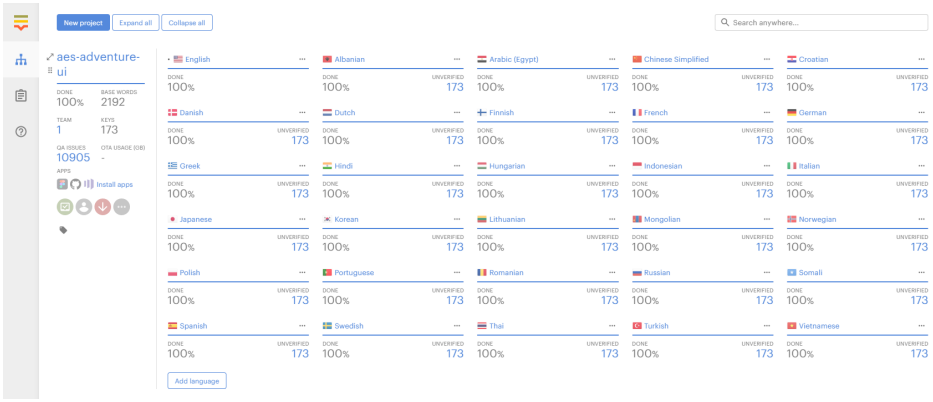

Figure 6.7: aes-adventure-ui Lokalise project

6.3.5 Search Engine Optimization

SEO is critical to *AES Adventure* for students and teachers to easily discover the tool. A number of steps were taken to maximise SEO (see 7.1.2).

Firstly, Nuxt 3 was selected as the framework upon which the app was built. Since Nuxt renders as much as possible on the server, there are large swaths of the application that are immediately available to web crawlers. In addition, Nuxt is used to build an MPA, so there are multiple HTML files that link to each other (crawlers can go from one to the next).

Next, the internationalisation (i18n) methods were adjusted. Initially, changing the language was done with JavaScript and had no impact on the page routes or generated HTML files. This did not follow i18n best practices, so it was refactored so that all application routes can be prefixed with a language code to access the target resource in the corresponding locale. In addition, a number of i18n elements were added to the `<head>` tag of every page, including `og:locale` and alternate links (hrefLangs). The final i18n related SEO work done was adding the `lang` attribute to the root `<html>` tag so Google can recognize what language each page is in.

Lastly, all the SEO chores were completed. This includes the following:

- Adding a `rel` attribute to all `<a>` tags.
- Adding a localised description to each page (using a `<meta>` tag).
- Adding other og attributes for embedding previews. This includes `og:image`, meaning that when an *AES Adventure* link is sent through a service like Discord, the resulting embed

preview will be custom with the *AES Adventure* title, banner image, and more (see Fig. 6.8).

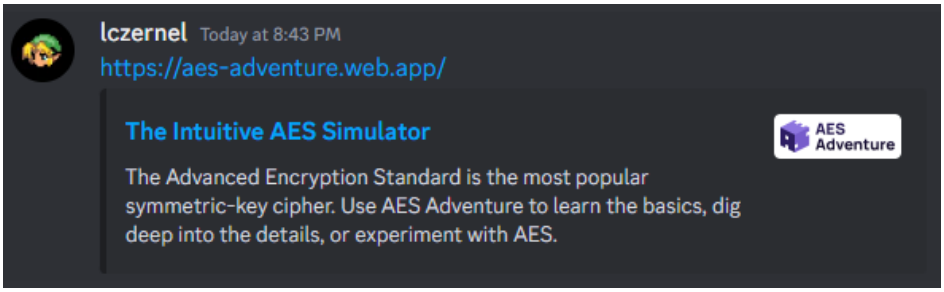

Figure 6.8: Custom *AES Adventure* embed preview

6.3.6 Statistics

AES Adventure does not expect its users to look at blocks of hex values and compare them manually to draw conclusions about the cipher. In addition, the metrics that the tool does provide are formatted in a way that anyone can understand: a percentage. The two metrics presented to users represent the confusion and diffusion levels at each round of the encryption and decryption processes. Confusion and diffusion were first defined by Claude Shannon as the fundamental properties of secure cipher operation [24].

Diffusion

Diffusion is the act of spreading the influence of a single plaintext bit over many ciphertext bits [3]. This is represented in *AES Adventure* by measuring the avalanche effect at each round of the algorithm. This means that given plaintext p, every variation of p where 1 bit is flipped $p_0, p_1, ..., p_n$ and ciphertext at the current round c, the avalanche effect at the current round is calculated by:

$$\frac{1}{n}\sum_{i=0}^{n} dist(p_i, c)$$

Where $dist(a, b)$ is the sum of the different bits at each position between two binary vectors a and b.

This value is then converted into a percentage of the maximum possible avalanche effect wherein all ciphertext bits are flipped for each plaintext variation. The final percentage is presented as the diffusion level for a given round. This works quite well in representing diffusion as it can be used in conjunction with the configuration options to verify the diffusive layers of AES; by configuring the simulator to exclude one of these layers, diffusion occurs much slower through the rounds or never hits the desired 50% mark.

Confusion

Confusion is the act of obscuring the relationship between the ciphertext and key [3]. This is represented in *AES Adventure* by the normalised circular cross-correlation between the key and ciphertext at the current round. This means that given the n-bit key k and ciphertext at the current round c, the confusion is calculated by:

$$\rho(k) = \frac{1}{\|k\|\|c\|} \sum_{j=0}^{n} c_j k_{((j+k) \bmod n)+1} \tag{6.1}$$

$$\max(\rho(0), \rho(1), ..., \rho(n-1)) \tag{6.2}$$

Where all the 0s in c and k are mapped to -1s and $a_i \in \{-1, 1\}$ is the value at the ith index within the vector a. The ciphertext is padded with 0s if needed.

This value is then converted into a percentage and presented as the confusion level for a given round. This is an okay-at-best representation of confusion; for most inputs and keys, the confusion level jumps up to high 80% after the first round. When the algorithm is configured to turn off the source of confusion (substitute bytes), only with a few input and key pairs, the confusion level takes 2 or 3 rounds to reach its normal value. An example of one such 'nice' plaintext and key pair is $p = 01010101001010101001010101001010$ and $k = 11111111111111111111111111111111$ (see Fig. 6.9).

(a) Standard AES

(b) No Substitute Bytes

Figure 6.9: Comparing confusion levels with versus without Substitute Bytes with a 'nice' key.

6.3.7 AESi

The biggest challenge behind AESi was gathering the intermediate encryption/decryption/key expansion data. It had to be done with the UI in mind; what is going to be shown to users drove the decision making. This is evident in the return type of AESi:

```
export type AesiOutput = {
  // Plaintext or ciphertext
  block: Uint8Array

  // Input transformed into state matrix
  initialState: Uint8Array

  // Initial or final key addition
  symmetryKeyAddition: AesiRoundStep

  // Includes short round
  rounds: Array<AesiRound>
}
```

Listing 6.4: Encryption return type

Carefully *deep* copying the arrays of hex values in each step was required - but even if a mistake was made and a shallow copy was made (that later caused an issue), this would be caught by one of the unit tests that were previously discussed.

6.4 Distribution

AES Adventure is accessible via any modern browser at https://aes-adventure.web.app. The app cannot be pre-rendered and deployed via a static hosting service alone because of the large amounts of application data that are serialised and stored in `localStorage`.

6.4.1 Services

The deployment is broken down into two parts, both of which are part of Google's Firebase suite:

- Firebase Hosting: serves the server-side rendered app route HTML files through Google's global CDN for fast response times, with the SSL certified domain provided by Firebase. To be clear, large sections of the app are only rendered client-side as they require data in `localStorage`. Firebase Hosting also serves the static files in the `public` directory such as the app logo SVG.
- Firebase Functions: serves additional files that are lazily requested for client-side rendering. Such files include minified and chunked JavaScript, and CSS files.

These two services' logs are collected by Google Cloud Logging. These logs can be inspected and queried in the logs explorer (see Fig. 6.10).

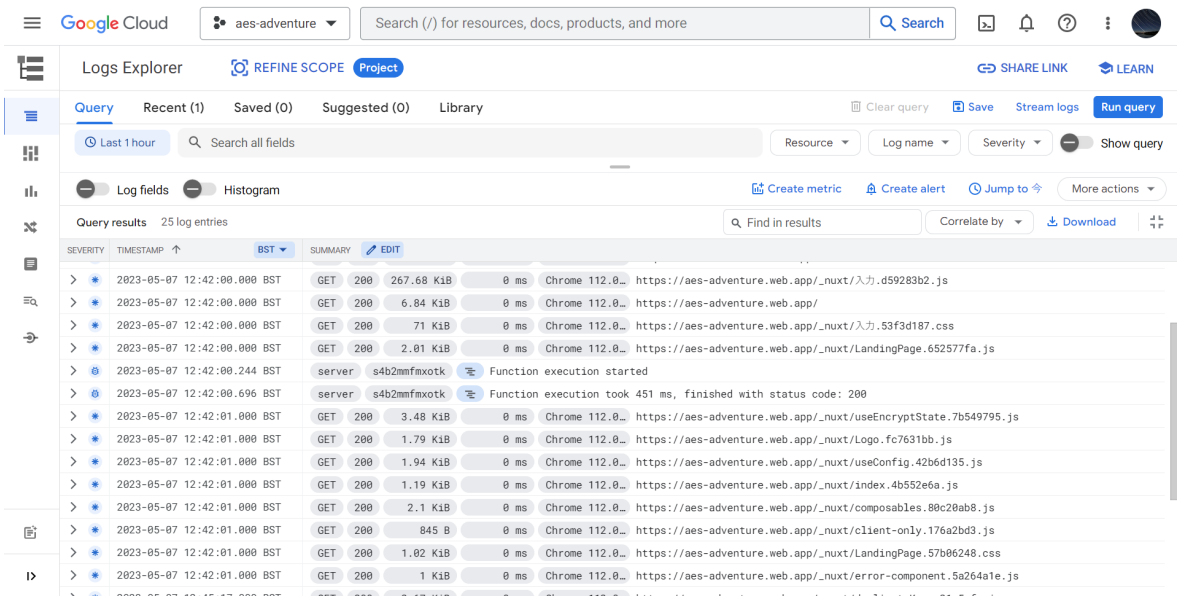

Figure 6.10: Logs Explorer

6.4.2 Error Monitoring

When any error occurs, it is picked up by *AES Adventures*' integration with Sentry. These errors can be inspected in Sentry's web app (see Fig. 6.11) for 30 days before they are deleted. A longer data retention policy will require an upgraded plan (above free-tier).

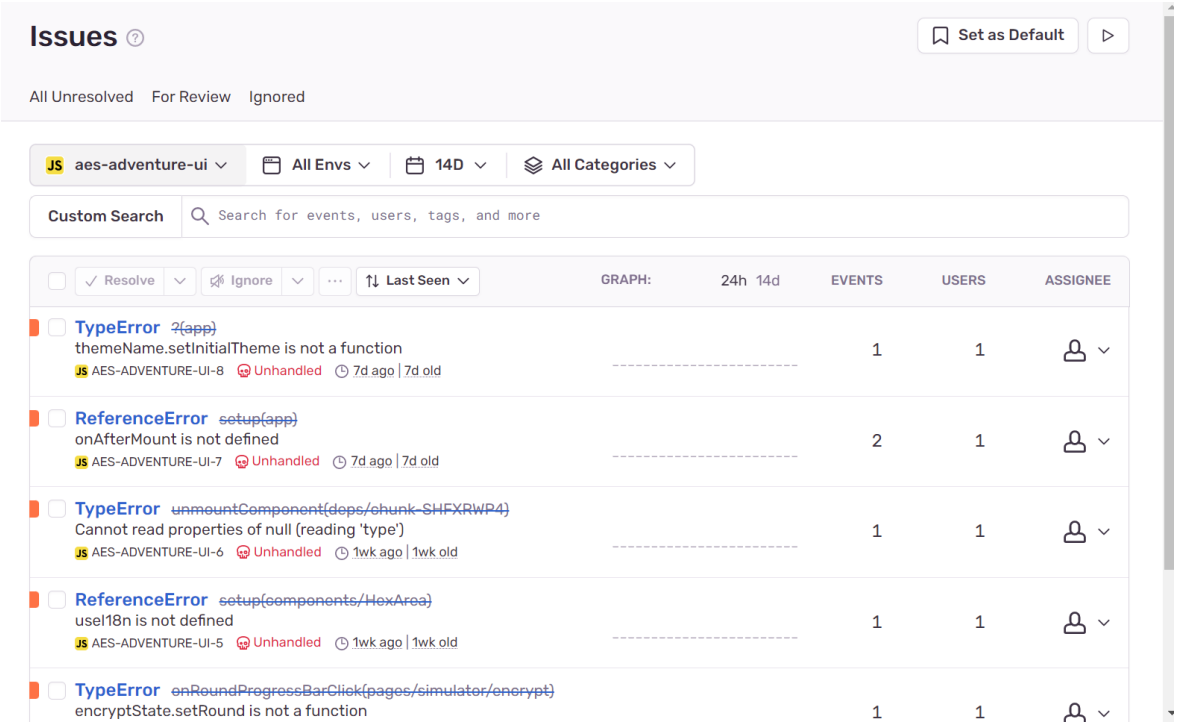

Figure 6.11: Sentry Issues dashboard

6.4.3 Cost

Google is quite generous with their Firebase free-tier limits. Cloud Functions are only accessible on their pay-as-you-go Blaze plan, but the already high free-tier limits still apply, meaning the upgrade is essentially free.

Cloud Functions allows for 2,000,000 requests per month for free, beyond that will start to cost cents. The average user session will make no more than about 20 invocations, meaning 100,000 monthly active users (MAU) will cost nothing to serve the cloud functions to.

The metric to watch for pricing is the downloads from Cloud Hosting, which allows for 360MB per day for free. An upper bound on the average amount of data downloaded per user is 3MB, meaning 120 daily active users (DAU) will cost nothing to serve the static files from Cloud Hosting.

Therefore, the amount of usage at which the distribution of *AES Adventure* will begin to cost is roughly 3,600 MAU.

Chapter 7: Evaluation

After the allocated development time had ended and *AES Adventure* was deployed, it was evaluated using Chrome Lighthouse audits and user tests. Each of these methods and results are described and discussed below.

7.1 Lighthouse Audits

7.1.1 Method

A Chrome Lighthouse audit was executed on various pages of *AES Adventure* in different states. These audits calculate a single score for each of the following categories: performance, accessibility, best practices, and SEO. While they are not perfect (e.g. some items must be manually checked), they are a great way to quickly get some summary measurements for a site.

7.1.2 Results

The following table shows the results of the audits:

Page	Performance	Accessibility	Best Practices	SEO
Landing	87	100	100	100
Guide	97	100	100	100
Preferences	99	90	100	100
Feedback	92	100	100	100
Encrypt	91	98	100	100
Decrypt (mid-simulation)	86	89	100	100
Key Expansion (mid-simulation)	82	96	100	100

7.1.3 Discussion

Overall, the results above are a positive indication that some of the original goals of this project were indeed hit, with high accessibility scores and full marks for SEO on all pages.

Performance never hits 100 - the primary issue identified by the audit is slow server response time. A large contributing factor is likely that the Firebase Functions (server) are deployed in `us-central1`. This region is *not* ideal as these audits were executed by a client in Europe.

The landing page has a relatively poor performance score because the infographic image being served is not optimized; it much larger than it needs to be (image size vs rendered size). Reducing the size of the image used would boost this performance score.

Key expansion (mid-simulation) is lower than encryption and decryption because of the much larger

quantity of DOM elements - almost 3 times more. This is the case for two reasons:

1. Two of the steps in key expansion involve substitution boxes which each have large DOM trees. Each box is a 16 by 16 grid where each element has 2 children elements, one for each nibble.
2. Key expansion is word oriented; and so are its animations. This means that not only do the bytes have 2 children elements, but every 4 bytes is further encased by a single `<div>` that groups them as a single word.

Another refactor that would boost performance score by reducing the number of DOM elements is to optimize `hexToDivs`. Currently, it creates a DOM structure that allows for the animation of each individual nibble. Some animations need this, but others don't (only need to animate bytes or words at a time)! Each animation should be able to specify the granularity at which they must animate when invoking `hexToDivs`. This would easily half the number of DOM elements being created for animations because one of the biggest perpetrators of creating too many nodes is SubstituteBytes - an animation that does not need to animate each individual nibble of the substitution box! Another example is the key expansion state to round keys step - for 256-bits, it has a massive 1566 child elements, while the animation only needs to animate each word at a time!

Accessibility generally stays high but dips for mid-simulation pages. The most common accessbility issues on these pages are:

- The components for each simulator step uses Vuetify's expansion panel component, which has a mismatch of aria attributes to roles.
- Heading elements are not in sequentially-descending order. Specifically, the tutorial heading uses `<h4>` tags when they should be `<h2>` in order to pass this check. That would look awful, so the solution may be to use the correct tag but override the default CSS properties of the heading.

AES Adventure scores full marks in the other two categories on all pages.

7.2 User Testing

7.2.1 Method

The target user group for *AES Adventure* is Computer Science undergraduates. Therefore, 5 students that fit this criteria partook in the same test over the course of a week. The students had varying levels of prerequisite knowledge, but all were proficient enough to be at least at the 'Beginner' level previously discussed. Tested browsers include Chrome, Safari, Brave, and Firefox.

This test consisted of a series of tasks for the user to complete roughly in order. The tasks started in the landing page, then moved over to each simulator, and finally back to some of the landing pages. The phrasing of the tasks is important; care was taken to avoid revealing *how* to complete the task in the task itself. For example, users were never asked to encrypt something, as this would hint that the button they must press has 'encrypt' written on it. Instead, they were asked to

obtain a block of ciphertext. The test users shared their screen to the test moderator as they were working through them. The test moderator remained quiet, avoiding answering any questions and taking notes. After each task, the test users rated how easy the task was on a System Usability Scale (SUS) [25]. Test users were told that it is up to them whether they 'think aloud' as they progress, but either way, after they completed the tasks, a retrospective interview was held for the moderator to query the test user's thought processes as they displayed various behaviours. In addition to the post-interview, after the tasks were completed the test users were asked to rate the usability of *AES Adventure* in terms of 3 key quality in use measures: effectiveness, efficiency, and satisfaction [26]. Lastly, participants were reminded that it was the interface being tested, and not them.

7.2.2 Results

The maximum, minimum and average minutes spent completing the tasks were 27, 18 and 23.4 respectively. All participants responded 'Strongly Agree' to the following tasks:

- *Find some information to help you get started with AES Adventure.*
- *Control the position and playback speed of the animations.*
- *Finish this process* and retrieve the result.* *refers to encryption.
- *Get additional information for parts* that you don't understand.* *refers to steps of decryption.
- *Discover the total number of round keys created from a 256-bit key.*
- *Change the display language of the app to French.*
- *Voice your thoughts about AES Adventure.*
- *Change the theme to dark mode.*

Of the remaining tasks, the following received mostly 'Strongly Agree' and at least one 'Agree' (still overwhelmingly positive results):

- *Get a block of ciphertext using repeated '01's as the plaintext and all '1's for the key.*
- *Get additional information for parts* that you don't understand.* *refers to steps of encryption.
- *Return to the start of round 2** *after decryption.

The rest of the tasks received mixed results; the following plots show the distribution of responses for each of these tasks (see Fig. 7.1, 7.2, 7.3).

The following means are calculated from the test user ratings for the quality in use measurements (100% is the best, 50% is neutral, 0% is the worst):

- *Effectiveness*: 84%
- *Efficiency*: 96%
- *Satisfaction*: 64%

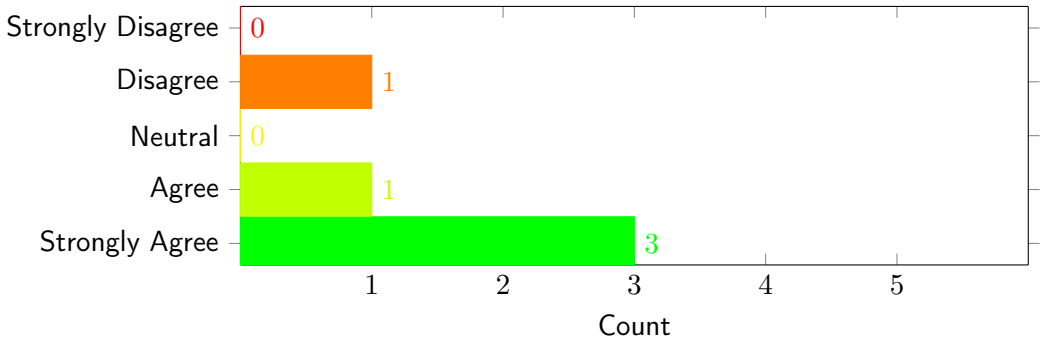

Figure 7.1: *Use the same key as before* and the previous output to begin the opposite AES process.* *from encryption task.

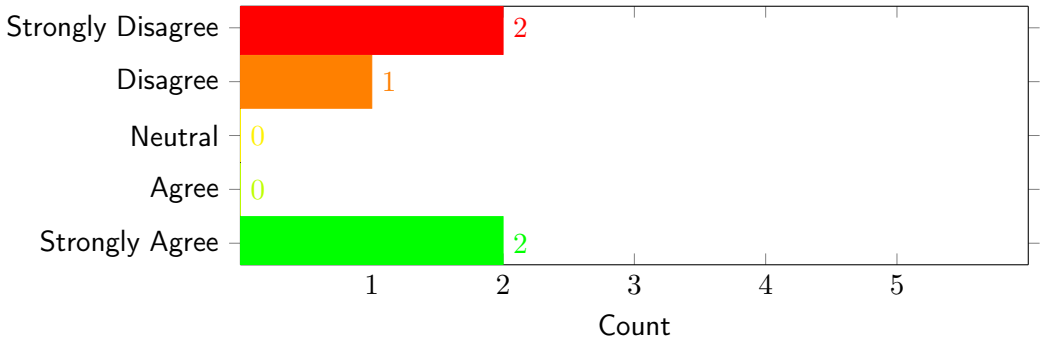

Figure 7.2: *Discover the confusion and diffusion levels at the start of round 8.*

7.2.3 Discussion

Overall, these user tests were extremely beneficial. The results above do not capture all of the findings; notes from user behaviour (moderator was observing the users throughout the duration of the test) informed some powerful decisions that improved *AES Adventure*. More on that in the following subsection.

It is important to note that the testing methods had many flaws, so the results should be treated accordingly. The first of these flaws is the sample size - only 5 users participated. These fresh, external perspectives were enough to identify glaring issues in the interface that I couldn't see as the developer. However, this small test user group is not nearly enough to be able to draw empirically sound conclusions. Another flaw with the testing was that no attempt was made to normalize the mentality of each test user in their interactions with the tool. For example, from post-interviews, it was discovered that some test users felt their job was to test the interface and *not* to learn about AES using the tool. This would then affect their focus and attention to particular components of the app. To name one more flaw: diversity of the user group in terms of gender and prerequisite knowledge was almost non-existent.

However, even in this small sample, we can make some observations.

1. Landing pages scored perfectly, which suggests that the interface on those pages is acceptable.
2. Related tasks between encryption and decryption got higher scores in decryption and key expansion, which suggests that users can learn patterns in one simulator and later apply those patterns in the other simulators.
3. Most users struggled to find the statistics dropdown that contains the confusion and diffusion, which suggests that there are not enough affordances to indicate its presence.

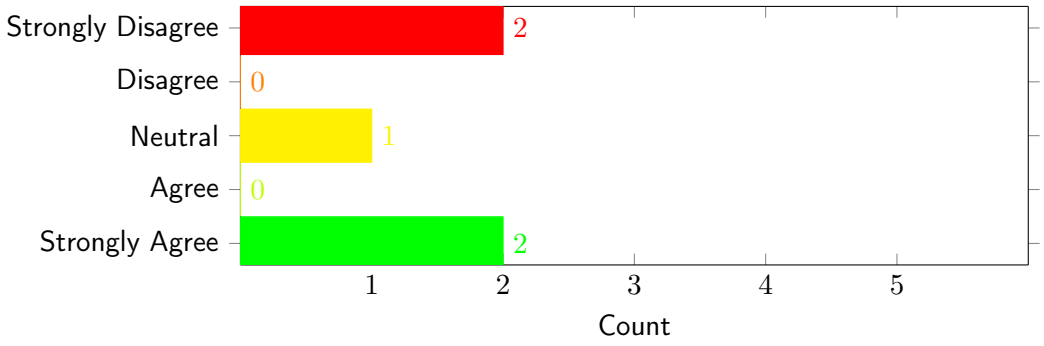

Figure 7.3: *Turn off the ShiftRows layers.*

4. Most users struggled to configure the simulators, which suggests that the simulator is not clear that it cannot be configured in the simulator itself.

7.2.4 Consequences

After the testing period had ended, as a result of the above findings, many minor changes were made to *AES Adventure*. There was not enough time left in this project's timeline for more extensive changes. The key changes are discussed here.

Tooltips were added to the copy buttons that explain what each one will do on-click. This is expected to improve users ability to complete tasks like that seen in Fig. 7.1.

'View Stats' text was added to the statistics dropdown to give a clear indication that clicking it will reveal statistics. This is expected to improve users ability to complete tasks like that seen in Fig. 7.2.

The simulator sidebar behaviour was changed to reopen when a tutorial icon button is pressed. Multiple users encountered this and in post-interviews they conveyed how they thought the corresponding tutorial was either missing or something was broken!

A layer on/off indication icon was added to encryption and decryption steps that can be configured to be turned off. It was observed that users went to step X looking to try and turn step X off - now there is an element there that guides them to the configure modal. This is expected to improve users ability to complete tasks like that seen in Fig. 7.3.

7.3 Comparison with Existing Tools

In my admittedly subjective opinion, *AES Adventure* has the most modern, easy-to-use, and polished interface of any of the existing tools that were discussed in section 3.2. The table below provides a more objective comparison by indicating the presence of features found in the existing tools versus *AES Adventure*. The rows were chosen by inspecting each tool and identifying features that have not already been listed. Minor *AES Adventure* features were excluded for brevity. ✓ indicates a strong implementation of the corresponding feature. ✗ indicates no implementation of the corresponding feature. 'Meh' indicates a somewhat-lacking implementation of the corresponding feature.

Feature	AES Adventure	AESvisual	CryptTool-Online: AES	AES Visualizer	Rijndael Animation
Encryption: AES-128	✓	✓	✓	✓	✓
Encryption: AES-192	✓	✗	✓	✗	✗
Encryption: AES-256	✓	✗	✓	✗	✗
Decryption: AES-128	✓	✓	✓	✗	✗
Decryption: AES-192	✓	✗	✓	✗	✗
Decryption: AES-256	✓	✗	✓	✗	✗
Expansion: AES-128	✓	✓	✓	Meh	✓
Expansion: AES-192	✓	✗	✓	✗	✗
Expansion: AES-256	✓	✗	✓	✗	✗
Custom input	✓	✗	✓	✓	✗
Practice mode	✗	✓	✗	✗	✗
Isolated layers	Meh	✓	✗	Meh	✗
Multiple blocks	✗	✗	Meh	✗	✗
Configuration	Meh	✗	✓	✗	✗
Byte marking	✗	✗	✓	✗	✗
Internationalization	✓	✗	Meh	✗	✗
Animation	✓	✗	✗	✗	✓
Theme	✓	✗	✓	✗	✗
Persistence	✓	✗	✗	✗	✗
Available in browser	✓	✗	✓	✓	Meh
Statistics visualisation	Meh	✗	✗	✗	✗

Chapter 8: Future Work

AES Adventure can be considered a great start to more perfect AES educational tool. There are many opportunities for improvement, some of which have already been discussed in this report, such as the minimum display width limitation (6.1.4) and optimising animation related performance (6.3.1 and 7.1.3). However, there are many more possible improvements, which I will describe next.

Implementing missed base and advanced project goals would be an excellent start. This includes creating a robust suite of unit and integration tests, and tracking user events to create a publicly available dashboard which could yield powerful insights to inform future development of AES educational tools.

Next, the usage of the Canvas API or WebGL APIs to create animations should be investigated and potentially implemented. Currently, some animation DOM trees are huge because CSS transforms are used to animate. `<canvas>` can handle large numbers of 2D elements with ease. On the other hand, WebGL would be far more challenging but could be far more performant as it draws hardware-accelerated graphics. Not only could it solve current performance concerns, but it opens the door to more complex animations (including 3D) at high frame rates. It is also worth noting that at the time of writing, the WebGPU API, the successor to WebGL, now ships in Chrome as an experimental technology. It is expected to become the new standard API for accelerated graphics.

Originally, it was planned that users could create their own configuration (see design). More expansive configuration options would be a great avenue of features to develop. Exportable and shareable configurations may also be of great use to educators seeking to distribute custom configurations to a group of students.

Another valuable feature that would have been implemented with more development time is encrypting and decrypting multiple inputs at a time. This would allow users to learn about the difference that just one flipped input bit would have on each step of the process. Comparing up to 3 inputs fits in the current design by stacking the inputs and outputs, reusing the headers (e.g. "Input", "Output") and any constants being used (e.g. s-boxes, constant matrices).

One feature that I was really looking forward to building but did not have enough time was the so-called byte marking (see 3.2.2) ability. It should be noted that the tool that this idea came from did not animate between steps, so while it was vital and insightful in that tool, bringing that feature to *AES Adventure* would be more of a nice-to-have.

Improving the statistics available to users (targeted towards advanced users) could assist them in drawing conclusions from their experiments with AES. This could come in the form of taking more measurements (e.g. before rounds, in between steps, and after all steps) and/or adding more metrics.

The tutorial components have tonnes of room for improvement. Currently, only simple transposition steps have a 'sandbox' environment in the corresponding tutorial where users can interact with the step in isolation - this should be possible for all AES steps! In addition, less reliance on external resources could improve the user experience, further reinforcing *AES Adventure* as a one-stop-shop for AES education. Another potential change is to make tutorial components dynamic with respect to the current simulator step. This becomes more important once the base functionality of tutorials is improved. For example, if tutorials had sample calculations, make those calculations use the numbers from the current step! Another example that could be implemented with the current tutorial components is to show a different tutorial based on whether encryption

or decryption is being used (for steps that are found in both encryption and decryption).

One final suggestion comes in light of the recent boom in AI research surging into the public eye - large language models like GPT-3. Integrating with a model could take the form of a prompt in the tutorial sidebar that allows users to ask highly context specific questions, with the model adopting the persona of a skilled computer science educator.

Chapter 9: **Conclusion**

An educational simulator of AES was successfully created, named *AES Adventure*. It has achieved many of the original goals set out (see 1.2), which include a research-informed design, highly visual and interactive, accessible, and comprehensive. It is capable of guiding users through the 3 major components of AES with any of the 3 key sizes in 30 different languages. *AES Adventure* also provides a guide in how to use the simulators as well as multiple themes. The project work plan was executed without issue, but it was a lot more involved than initially anticipated, which resulted in the robustness and usage tracking dashboard objectives being missed. Google's Firebase Functions and Hosting were used to deploy the application, which is expected to be free up to roughly 3,600 monthly active users. The tool performed well in both the Lighthouse audits and user testing, but there is room for improvement. There are many features left to-be-desired as well as performance enhancements.

Acknowledgements

I am extremely grateful for the project advisor, Dr Félix Balado. *AES Adventure* would not exist in its current form without his guidance and suggestions - I can say without any doubt that his contributions to the tool resulted in a less cluttered, easy to use and polished product. In addition, many thanks to the test users that gave their time to interact with the tool, think critically about it, and give feedback.

Bibliography

1. Computer Security Division, I. T. L. *Block Ciphers - Cryptographic Algorithm Validation Program | CSRC | CSRC* EN-US. Oct. 2016. https://csrc.nist.gov/projects/cryptographic-algorithm-validation-program/block-ciphers.
2. FIPS 197, Advanced Encryption Standard (AES). en, 51.
3. Paar, C. & Pelzl, J. *Understanding Cryptography: A Textbook for Students and Practitioners* en. http://link.springer.com/10.1007/978-3-642-04101-3 (Springer Berlin Heidelberg, Berlin, Heidelberg, 2010).
4. Nechvatal, J. *et al.* Report on the development of the Advanced Encryption Standard (AES). *Journal of Research of the National Institute of Standards and Technology* **106.** https://nvlpubs.nist.gov/nistpubs/jres/106/3/j63nec.pdf (May 2001).
5. Hundhausen, C., Douglas, S. & Stasko, J. *A Meta-Study of Algorithm Visualization Effectiveness | Elsevier Enhanced Reader* en. June 2002. https://reader.elsevier.com/reader/sd/pii/S1045926X02902375?token=26FBCE10AE92AD5906BECC53477FDE6FC8AD1B6120EFB10EFB3A7D2
originRegion=eu-west-1&originCreation=20221117125321.
6. Lazaridis, V., Samaras, N. & Sifaleras, A. An empirical study on factors influencing the effectiveness of algorithm visualization. en. *Computer Applications in Engineering Education* **21.** _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cae.20485, 410–420. https://onlinelibrary.wiley.com/doi/abs/10.1002/cae.20485 (2013).
7. Schweitzer & Baird. *The Design and Use of Interactive Visualization Applets for Teaching Ciphers* in *2006 IEEE Information Assurance Workshop* (June 2006), 69–75.
8. Schweitzer, D. & Brown, W. USING VISUALIZATION TO TEACH SECURITY. en, 8 (2009).
9. Stasko, J., Stasko, J. T., Brown, M. H., Domingue, J. B. & Price, B. A. *Software Visualization: Programming as a Multimedia Experience* en (MIT Press, 1998).
10. Eisenstadt, M., Price, B. A. & Domingue, J. Software visualization as a pedagogical tool. en. *Instructional Science* **21,** 335–364. http://link.springer.com/10.1007/BF00121202 (1993).
11. Saraiya, P. Effective Features of Algorithm Visualizations. en, 133 (July 2002).
12. Naps, T. & Anderson, J. Evaluating the Educational Impact of Visualization. en, 13.
13. Bostock, M. *Visualizing Algorithms* June 2014. https://bost.ocks.org/mike/algorithms/.
14. Lin, S., Fortuna, J., Kulkarni, C., Stone, M. & Heer, J. Selecting Semantically-Resonant Colors for Data Visualization. en. *Computer Graphics Forum* **32.** _eprint: https://onlinelibrary.wiley.com/doi/pdf/10
401–410. https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.12127 (2013).
15. *About Colour Blindness* en-GB. https://www.colourblindawareness.org/colour-blindness/ (2022).
16. Ma, J., Tao, J., Keranen, M., Mayo, J. & Wang, C. AESvisual: A Visualization Tool for the AES Cipher. en, 6.
17. *AES (step-by-step) - CrypTool Portal* https://www.cryptool.org/en/cto/aes-step-by-step (2022).
18. Devendorf, L. *AES Visualization : Built with Processing by Laura Devendorf* http://www.artfordorks.com/visualization/AES/ (2022).
19. *AES - Advanced Encryption Standard* http://advancedcomputing.org/aes_visualizer.html (2022).

20. *AES Visualizer Step by Step in Javascript* http://advancedcomputing.org/aes.html (2022).
21. Zabala, E. & Esslinger, B. *Rijndael_Animation_v4_eng-html5* https://formaestudio.com/rijndaelinspector/archivos/Rijndael_Animation_v4_eng-html5.html (2022).
22. Ivanovs, A. *JavaScript Animation Libraries: 10 Popular Choices* en-US. Apr. 2022. https://stackdiary.com/javascript-animation-libraries/.
23. Darius, P. S. H., Gundabattini, E. & Solomon, D. G. A Survey on the Effectiveness of Online Teaching–Learning Methods for University and College Students. en. *Journal of The Institution of Engineers (India): Series B* **102,** 1325–1334. https://doi.org/10.1007/s40031-021-00581-x (Dec. 2021).
24. Shannon, C. E. Communication Theory of Secrecy Systems*. en. *Bell System Technical Journal* **28,** 656–715. https://ieeexplore.ieee.org/document/6769090 (Oct. 1949).
25. Brooke, J. SUS: A quick and dirty usability scale. *Usability Eval. Ind.* **189** (Nov. 1995).
26. ISO/IEC JTC 1/SC 7 Software and systems engineering. *ISO/IEC 25022:2016* en. June 2016. https://www.iso.org/standard/35746.html.