SCIFEST@COLLEGE

2024

Enhancing Data Security in AI Training using Zero-Knowledge Proofs for Secure Methodology Development.

Addison Carey

Stand Number:

Project Report Book

Contents

Abstract (250 words)

Training AI models securely has been a huge problem, facing challenges like handling large volumes of sensitive data and implementing strong security measures like encryption, access control, and compliance with regulations. Adversarial attacks, where malicious input can manipulate AI systems, are also a significant concern. These factors make ensuring the privacy and security of AI training data complex and crucial.

During the lockdown I started learning about Artificial Intelligence (AI), later on I learned about Zero-Knowledge Proofs (ZKPs). AI refers to the development of computer systems that can perform tasks that normally require human intelligence, such as understanding natural language, recognizing patterns and making decisions.

ZKPs are cryptographic protocols that enable one party (the prover) to prove to another (the verifier) that they possess certain knowledge without revealing the actual knowledge itself. This concept allows for authentication and verification without disclosing sensitive information.

I started my project in **February 2024**. My Project involves developing a method to implement ZKPs into AI to ensure a secure way of training sensitive data. The goal is to develop an algorithm to encrypt the data utilising zero-knowledge protocols, which will allow AI models access to the data without disclosure of the raw data.

Implementing ZKPs into AI in real world scenarios involve extremely advanced encryption and mathematical procedures. For my project, I've decided to train a simple Linear Regression model based on the data encrypted with a basic ZKP. This will be able to demonstrate a secure training on data, at a basic level.

Introduction

My interest in Computer Science and Coding goes back to early Primary school. I started using Scratch and Micro:Bit but soon transitioned onto HTML/CSS and JS. Python was my introduction to the power of computer code to manipulate data.

Training AI models on sensitive data in a secure way is a significant challenge due to several reasons. First, handling sensitive data, such as personal or financial information, requires strict adherence to privacy regulations like GDPR or HIPAA, which adds complexity and legal considerations. Ensuring data security throughout its lifecycle - from collection and storage to processing and analysis, demands robust encryption, access controls, and secure data handling practices. The sheer volume of data needed for AI training means there's a higher risk of exposure or breaches, necessitating continuous monitoring and mitigation strategies. Lastly, the potential for adversarial attacks, where AI models can be manipulated or misled, underscores the importance of rigorous security measures in AI training pipelines. Balancing the benefits of AI with data protection and security remains a key challenge for organisations across various sectors.

With the rapid advancement of artificial intelligence technology, the usage of machine learning models is gradually becoming part of our daily lives. High-quality models rely not only on efficient optimisation algorithms but also on the training and learning processes built upon vast amounts of data and computational power. However, in practice, due to various challenges such as limited computational resources and data privacy concerns, users in need of models often cannot train machine learning models locally (Xing, et al. 2023).

My project involves the development of a ZKP algorithm in Python, to help ensure AI models can train on data securely, without access to the raw data through zero-knowledge. ZKPs are proof systems with two parties: a prover and a verifier,

along with a challenge that gives users the ability to publicly share a proof of knowledge or ownership without revealing the details of it. In essence, they demonstrate knowledge of a fact without disclosing the details of that knowledge. They have shown that, it is feasible to demonstrate that some theorems are true without providing the slightest indication of why they are true. For example, where a first party or a prover try to persuade the second party or verifier that the statement is true without sharing any hints or information to the verifier. An analogy for this is to imagine you have a magic trick where you can prove to your friend that you've solved a puzzle without revealing the solution. You show them the puzzle, cover it, and then reveal it solved without ever showing how you did it. ZKPs work similarly in cryptography: they allow one party to prove to another that they know a secret without revealing the secret itself.

My Hypothesis is that using ZKPs to allow secure training of AI models on sensitive data can prevent the disclosure of raw data to AI models. Using this method with advanced encryption and mathematical aspects could have to potential to revolutionise the world of AI, especially in areas like healthcare or finance where dealing with sensitive data is very common. My ZKP will be able to encrypt data ensuring zero-knowledge properties, and a simple Linear Regression model will be able to train on that encrypted data, with no disclosure or access to the original data.

Explanations of Components and Terms

In this section of my report book, you will find explanations of each primary component of the Report Book. These Primary Components include Zero-Knowledge Proofs, Artificial Intelligence, Linear Regression Models and the mathematical aspects of ZKPs in AI.

Zero-Knowledge Proofs

A Zero-Knowledge Proof (ZKP) is a method of proving the validity of a statement without revealing anything other than the validity of the statement itself. It is a proof system with a prover, a verifier, and a challenge that gives users the ability to publicly share a proof of knowledge or ownership without revealing the details of it. In essence, they demonstrate knowledge of a fact without disclosing the details of that knowledge. As their name suggests, ZKPs are cryptographic protocols that do not disclose the data or secrecy to any eavesdropper during the protocol. This concept was first introduced by MIT scientists Shafi Goldwasser, Silvio Micali, and Charles Rackoff in the 1980s. ZKPs have since found applications in various fields, including cryptography, privacy, and blockchain technology (Hasan 2019).

ZKPs must satisfy three characteristics to have evidence of zero-knowledge:

- **Completeness:** If the statement is true, an honest verifier will be convinced by an honest prover.
- **Soundness:** If the statement is false, no dishonest prover can convince the honest verifier. The proof systems are truthful and do not allow cheating.
- **Zero-Knowledge:** if the statement is true, no verifier learns anything other than the fact that the statement is true.

There are several types of ZKPs, and they can be categorised based on the nature of the statement being proven. Some of the more common types include:

1. **Interactive Zero-Knowledge Proofs (iZKPs):** In an iZKP, the prover and verifier engage in a series of communication rounds to establish the validity of the statement. The prover convinces the verifier without revealing the actual information.
2. **Non-Interactive Zero-Knowledge Proofs (NIZKPs):** Unlike iZKPs, NIZKPs require only a single round of communication. The prover can provide a proof that can be verified by the verifier without any further interaction.
3. **Statistical Zero-Knowledge Proofs (SZKPs):** These proofs allow a small, negligible probability of error. The verifier is convinced with high probability that the statement is true.
4. **Perfect Zero-Knowledge Proofs (PZKPs):** In a PZKP, the verifier learns absolutely nothing about the statement being proven.

In a real cryptographic ZKP, complex mathematical algorithms and protocols are used to ensure that the proof is sound and secure. ZKPs are used in various applications such as authentication systems, digital signatures, and privacy-preserving protocols in blockchain technology (such as Zcash's zk-SNARKs). They provide a way to establish trust and verify information without revealing sensitive data, making them a powerful tool for enhancing security and privacy in digital interactions.

ZKPs can be used in authentication systems where a user can prove their identity without revealing sensitive information such as passwords or biometric data. This is particularly useful in scenarios where privacy is paramount, such as healthcare, finance, and voting systems. They enable selective disclosure of information. This means that a party can prove they possess certain data or meet specific criteria without revealing the actual data itself. This is valuable for

complying with regulations (such as GDPR) while still allowing for data analysis and verification (Hasan 2019).

ZKPs play a crucial role in blockchain technology to ensure transaction privacy and data confidentiality. For example, cryptocurrencies like Zcash and Monero use zero-knowledge proofs (zk-SNARKs and Ring Signatures, respectively) to provide anonymous and private transactions while still maintaining a public ledger. In decentralized and trustless systems like decentralized finance (DeFi), ZKPs enable users to verify transactions and smart contract executions without relying on a central authority. This promotes transparency and reduces the need for blind trust in intermediaries.

Zk-SNARKs (zero-knowledge Succinct Non-Interactive Arguments of Knowledge) are ZKPs that is a way to prove some computational facts about the data without disclosure of the data. They are the cryptographic instrument that underlies Zcash and Hawk, both building ZKP blockchains. These SNARKS are used in the case of Zcash to verify transactions, and they are also used in the case of Hawk to verify smart contracts.

Overall, ZKPs are a powerful cryptographic tool with broad implications for enhancing digital security and privacy. Their ability to prove knowledge without revealing sensitive information has revolutionized authentication, blockchain technology, data sharing, and cybersecurity. ZKPs provide a secure and transparent way to verify information, making them a key component in building trust and privacy in digital interactions across various domains (Hasan 2019).

Artificial Intelligence

Artificial Intelligence (AI) refers to the development of computer systems or machines that perform tasks that would typically require human intelligence. It creates intelligent systems that can perceive, learn, reason, and make decisions similar to how humans do. It involves developing algorithms and computer programs that learn from and make predictions or decisions based on data. These algorithms can be trained on large datasets, enabling them to recognise patterns and make predictions or decisions that are accurate and consistent.

At its core, AI is about enabling machines to mimic certain aspects of human intelligence, such as:

- Perception: AI systems perceive and interpret information from their environment. This includes tasks like image recognition, speech recognition, natural language processing, and understanding sensor inputs.
- Learning: AI learns from data and experiences to improve their performance. They can identify patterns, extract insights, and adapt their behaviour based on feedback.
- Natural Language Processing: AI understands and generates human language. This includes tasks like language translation, chatbots, sentiment analysis, and speech synthesis.
- Robotics and Automation: AI can be applied to robotics, enabling machines to perform physical tasks autonomously or assist humans in various domains. This includes industrial automation, autonomous vehicles, and robot assistants.

There are several types of AI, including rule-based systems, machine learning, and deep learning. Rule-based systems rely on a set of predefined rules to make

decisions or predictions. Machine Learning algorithms use statistical models to learn from data and improve their performance over time. Deep learning is a subset of machine learning that uses artificial neural networks to learn from data and make decisions or predictions.

Machine learning is a branch of computer science that broadly aims to enable computers to learn without being directly programmed. It has origins in the artificial intelligence movement of the 1950s and emphasises practical objectives and applications, particularly prediction and optimisation. Computers learn in machine learning by improving their performance at tasks through experience (Bi, et al. 2019).

AI is used in a wide range of applications, from virtual assistants like Siri and Alexa to self-driving cars, medical diagnosis, fraud detection, and more. As AI continues to advance, it has the potential to revolutionise many aspects of our lives and transform industries across the board.

Linear Regression Models

Linear regression is a statistical method used to model the relationship between a dependent variable and one or more independent variables. It's called "linear" because it assumes a linear relationship between the independent variable(s) and the dependent variable.

The goal of linear regression is to find the best-fitting linear relationship that explains the variation in the dependent variable based on the independent variable(s). This relationship is typically represented by the equation of a straight line:

$$y = mx + b$$

Where:

- y is the dependent variable (the one we are trying to predict or explain),

- x is the independent variable,
- m is the slope of the line, which represents the change in y for a one-unit change in x,
- b is the y-intercept, which is the value of y when $x = 0$.

In practice, Linear regression is a modelling technique for analysing data to make predictions. In simple linear regression, a bivariate model is built to predict a response variable (y) from an explanatory variable (x)[1] . In multiple linear regression the model is extended to include more than one explanatory variable (x_1,x_2,….,x_p) producing a multivariate model.

Linear regression models have several advantages that make them widely used in various fields:

- Interpretability: The linear regression equation ($y = mx + b$) is easy to interpret. The slope (m) tells us the change in the dependent variable (y) for a one-unit change in the independent variable (x). This makes it straightforward to understand the relationship between variables.
- Simple to implement: Linear regression is relatively simple to implement compared to more complex models. The calculations involved in fitting a linear regression model are well-established and computationally efficient, making it a practical choice for large datasets.
- Assumptions are clear: Linear regression has assumptions such as linearity, homoscedasticity (constant variance of residuals), independence of observations, and normality of residuals. These assumptions are relatively easy to understand and can be checked using diagnostic plots and statistical tests.

- Feature importance: In linear regression, the coefficients (slopes) of the independent variables provide information about the importance and direction of their influence on the dependent variable. Positive coefficients indicate a positive relationship, while negative coefficients indicate a negative relationship.
- Prediction: Linear regression can be used for prediction purposes, where you can input new values of independent variables into the model to predict the corresponding dependent variable value.

Despite these advantages, linear regression also has limitations. For example, it assumes a linear relationship between variables, which may not always be the case in real-world data. It's also sensitive to outliers and multicollinearity (high correlation between independent variables), which can affect the model's performance. In such cases, more advanced techniques like polynomial regression, ridge regression, or machine learning algorithms may be more appropriate.

In summary, Linear regression is a foundational statistical technique that serves as a fundamental building block in data analysis and predictive modelling. Its simplicity, interpretability, and versatility make it an excellent choice for many applications, from understanding relationships between variables to making predictions based on historical data. However, it's essential to recognise that linear regression relies on the assumption of a linear relationship between variables, which may not always hold true in complex real-world scenarios. Therefore, while linear regression is a powerful tool, it's crucial to consider its assumptions and limitations and explore more advanced techniques when dealing with non-linear relationships or complex data structures.

The Mathematical Aspects of ZKPs in AI

ZKPs are a fascinating tool in cryptography that can be applied to enhance privacy and security in various applications, including AI and machine learning. When using ZKPs in linear regression AI, we are typically concerned with proving certain computations or properties without revealing the underlying data or intermediate values.

After some initial research, I discovered that it can be extremely difficult to implement zero-knowledge into AI for multiple reasons. Considering my knowledge on these proofs, that makes it a lot harder to grasp the concept and create a fully working ZKP that completely prevents disclosure of data.

Below I've demonstrated a few examples of zero-knowledge being used in mathematics as well as the basic ZKP algorithm in my project.

Example 1:

```
# Define the secret value and a related public value
secret_value = 5.67
public_value = secret_value * secret_value
```

- Let *secret_value = 5.67* be the secret value that has to be proven. This can be any integer or float (decimal)
- The related public value is calculated as $public\_value = secret\_value^2$

```
# Define symbols for the zero-knowledge proof
x = symbols('x')
```

- Here, *symbols('x')* creates a symbolic variable x that is used in the equation

```
# Define the equation for the zero-knowledge proof
eq = Eq(x * x, public_value)
```

- This equation represents:

$$x^2 = public\_value$$

```
# Solve the equation to find possible secret values
possible_secret_values = solve(eq, x)
```

- This solves the equation $x^2 = public\_value$ for x, giving us possible values of x (which correspond to the possible secret values)

```
# Verify the zero-knowledge proof
verified = False
for val in possible_secret_values:
    if val == secret_value:
        verified = True
        break
```

- Here, it checks if any of the possible values obtained from solving the equation match the actual secret value. If a match is found, the verification is successful.
- Overall the code is essentially solving the equation $x^2 = public\_value$ to find possible values of x, and then checking if any of these values match the secret value.

Example 2:

The following example of a ZKP in maths uses modular arithmetic maths to prove knowledge of prime factors of a composite number.

Setup:

- $p = 7$ (prime number)
- $q = 11$ (prime number)
- $N = p \times q$

  $\rightarrow N = 7 \times 11 = 77$
- Generate values coprime to N (r_1, r_2)

  $r_1 = 5, \qquad r_2 = 6$

  $\text{hcf}(5,77) = 1 \ \therefore$ coprime

  $\text{hcf}(6,77) = 11 \therefore$ coprime

Prover (Commitments C_1 and C_2):

- $C_1 = r_1^2 \bmod N$

  $\rightarrow C_1 = 5^2 \bmod 77$

  $\rightarrow C_1 = 25 \bmod 77 = 25$

- $C_2 = r_2^2 \bmod N$

  $\rightarrow C_2 = 6^2 \bmod 77$

  $\rightarrow C_2 = 36 \bmod 77 = 36$

- $C_1 = 25, C_2 = 36$ are sent to the verifier

Verifier:

- Randomly chooses $b = \{0,1\}$

1. $b = 0$:

   Receives value $r_1 = 5$

   Verifies if $r_1^2 \equiv C_1 \bmod N$

   $\rightarrow r_1^2 = 5^2 \bmod 77$

   $\rightarrow r_1^2 = 25 \bmod 77 = 25$

   $C_1 = 25$

- Since $r_1^2 \equiv C_1$, the proof is verified and successful.

2. $b = 1$:

   Receives value $r_2 = 6$

   Verifies if $r_2^2 \equiv C_2 \bmod N$

   $\rightarrow r_2^2 = 6^2 \bmod 77$

   $\rightarrow r_2^2 = 36 \bmod 77 = 36$

   $C_2 = 36$

- Since $r_2^2 \equiv C_1$, the proof is verified and successful.

Linear Regression in Maths:

Let's say we have a set of data points (x_i,y_i) where x_i represents the independent variable (such as years of experience) and y_i represents the dependent variable (such as salary).

Hypothetical data points:

$$\text{Years of Experience } (x_i): [1, 2, 3, 4, 5]$$

$$\text{Salary } (y_i): [30000, 35000, 40000, 45000, 50000]$$

The goal of linear regression is to find the best-fit line that represents the relationship between *x* and *y*. We assume a linear relationship of the form:

$$y = mx + b$$

where *m* is the slope of the line and *b* is the y-intercept.

Linear regression calculates the values of *m* and *b* that minimize the sum of squared differences between the actual *y* values and the predicted *y* values on the line. Using simple mathematical formulas, the slope *m* and y-intercept *b* can be calculated as:

$$m = \frac{n(\sum xy) - (\sum x)(\sum y)}{n(\sum x^2) - (\sum x)^2}$$

$$b = \frac{\sum y - m(\sum x)}{n}$$

where *n* is the number of data points, $\sum xy$ is the sum of the products of *x* and *y*, $\sum x$ is the sum of *x* values, $\sum y$ is the sum of *y* values, and $\sum x^2$ is the sum of squared *x* values.

Calculate necessary sums:

- $\sum x = 1 + 2 + 3 + 4 + 5 = 15$
- $\sum y = 30000 + 35000 + 40000 + 45000 + 50000 = 200000$
- $\sum xy = (1 \times 30000) + (2 \times 35000) + (3 \times 40000) + (4 \times 45000) + (5 \times 50000)$

- $\sum x^2 = (1^2) + (2^2) + (3^2) + (4^2) + (5^2) = 55$
- $n = 5$ (number of data points)

Calculate the slope (*m*) and y-intercept (*y*):

$$m = \frac{(5 \times 965000) - (15 \times 200000)}{(5 \times 55) - (15^2)} = 5000$$

$$b = \frac{200000 - (5000 \times 15)}{5} = 25000$$

Use Linear regression to predict:

Now that we have the values of $m = 5000$ and $b = 25000$, we can predict the salary for someone with e.g. 6 years of experience:

$$y = mx + b = (5000 \times 6) + 25000 = 30000 + 25000 = 55000$$

Therefore, according to this linear regression model, a person with 6 years of experience is predicted to have a salary of €55,000.

Concluding from this, the overall aim of my project is to see if both of these can be integrated together into an algorithm using zero-knowledge that can prevent data disclosure to a linear regression model. Whether I will or won't be able to do this, I believe that in the future this could be a big step in the world of AI, and the prevention of data disclosure to AI models should really be considered especially in areas where sensitive data is commonly used in AI models.

My Code / Experimental Methods

In this section, you will find a breakdown of the algorithms used in my code to create the AI model with ZKP algorithm integration, and mathematic aspects of integrating Zero-Knowledge protocols into a Linear Regression Model.

I decided to create my project using the principles of User-centred design (UCD). This is a design approach that focuses on creating products, services, and experiences that are centred around the needs and desires of the people who will use them. To achieve this, UCD relies on a set of principles that guide the design process. These principles include:

- Empathy: Understanding the needs, goals, and motivations of the users and putting oneself in their shoes.
- Collaboration: Involving users in the design process and collaborating with them to co-create solutions.
- Iteration: Prototyping and testing ideas quickly and repeatedly to refine and improve the design.
- Inclusivity: Designing for the diverse needs and abilities of all potential users.
- Accessibility: Creating products and services that are accessible and usable by people with disabilities.
- User testing: Gathering feedback from users and using it to inform and improve the design.

I decided to apply UCD using the Agile Framework. Agile methodology is a project management approach that enables flexibility and empowers practitioners to engage in rapid iteration. It is based on the Agile Manifesto (Fowler 2001), a set of values and principles for software development that prioritize the needs of the customer, the ability to respond to change, and the importance of delivering working software regularly. Agile methodologies, such as Scrum and Lean, are designed to help teams deliver high-quality products in a fast-paced and dynamic environment

by breaking down complex projects into smaller, more manageable chunks and continually reassessing and adjusting the project plan as needed. Agile teams are typically self-organizing and cross-functional and rely on regular communication and collaboration to achieve their goals.

While agile is best used in a team setting, it can be adapted to use by individual developers. I managed my Agile system using Kanban (Fig.1)

Fig.1

Kanban is a project management method that originated in Japan in the 1950s and is based on the principles of lean manufacturing. It is designed to help teams visualize and optimize their workflows by breaking down tasks into smaller, more manageable chunks and tracking their progress through a series of stages. Kanban uses a visual board to represent the distinct stages of a project, with cards representing individual tasks and columns representing the various stages of the workflow. The goal of Kanban is to increase efficiency and transparency by making it easier for team members to see where work is in the process, identify bottlenecks and inefficiencies, and adjust as needed. Kanban is often used in conjunction with other agile methodologies, such as Scrum, to help teams manage their work more effectively. (Corona 2013). Please scan here for my full Kanban journey.

Iteration 1:

Planning Phase

I started my project by creating a mind map (Fig. 2) of all the potential ideas that I could pursue through the lens of ZKPs. I looked towards a more topical issue that could enable the use of zero-knowledge to ensure privacy.

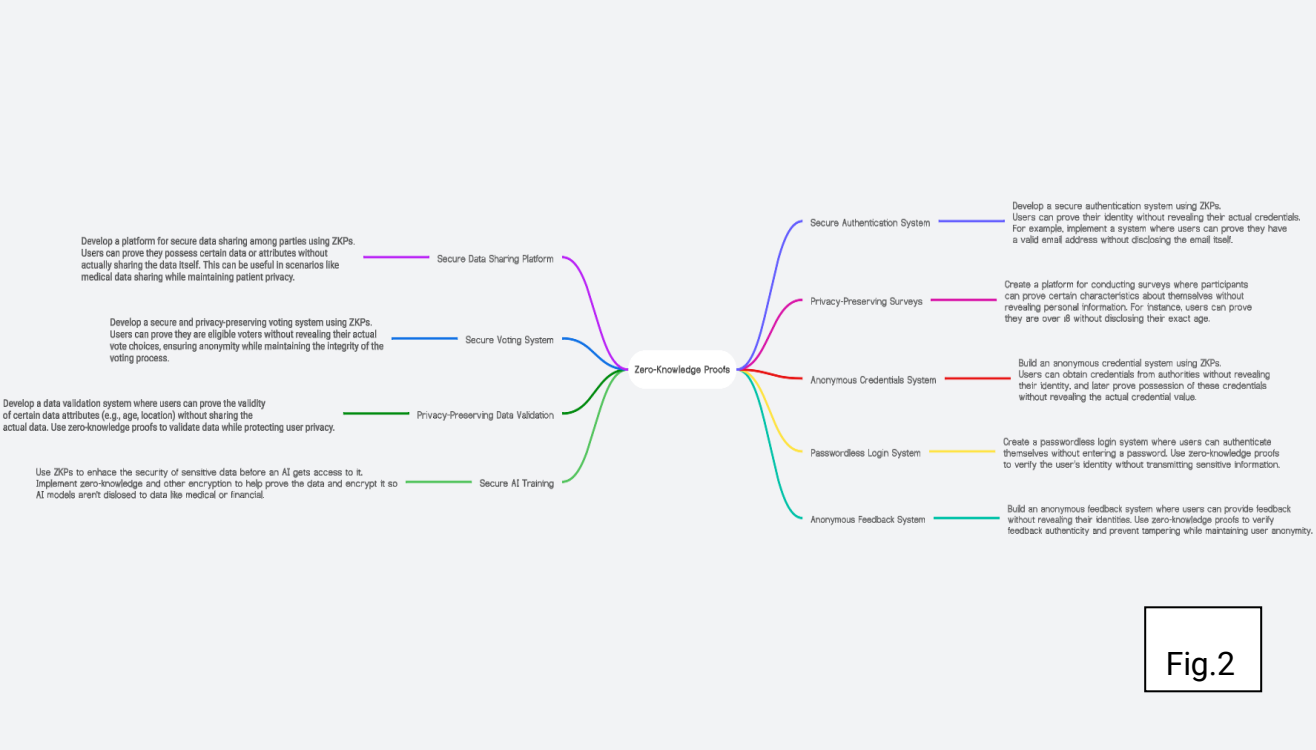


Fig.2

I settled (Fig 3.) on the problem of AI being trained on sensitive data and the creation of an algorithm implementing zero-knowledge to encrypt the data and ensure there is no disclosure of the raw data to the AI model. My Hypothesis is that using ZKPs to allow secure training of AI models on sensitive data can have the ability prevent the disclosure of raw data to AI models. Using this method with advanced encryption and mathematical aspects could have to potential to revolutionise the world of AI, especially in areas like healthcare or finance where dealing with sensitive data is very common.

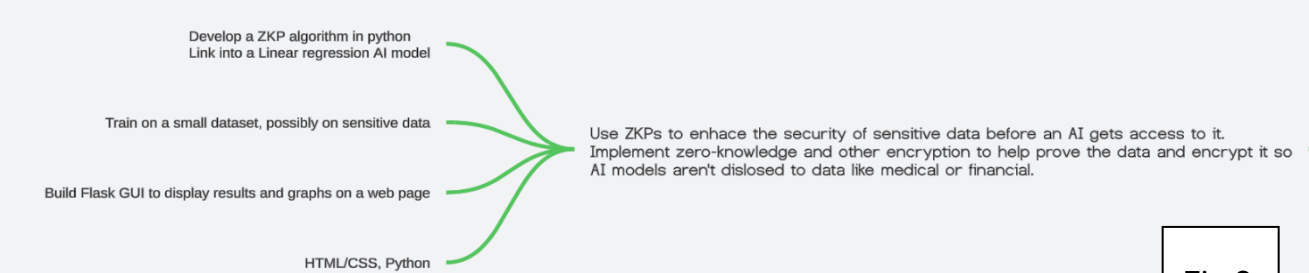


Fig.3

Design Phase

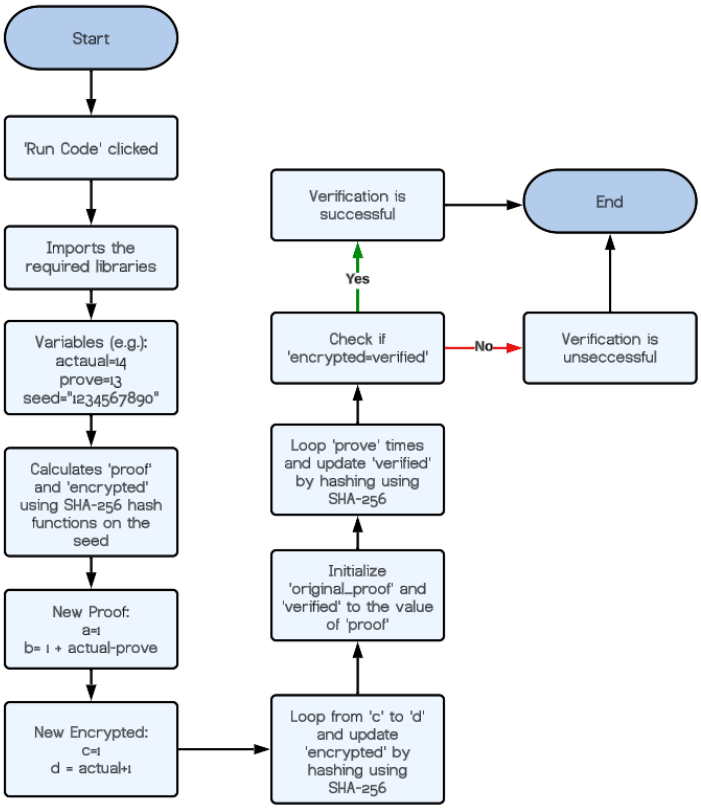


Fig.4

The basic algorithm seen above (Fig.4) for my first iteration was a simple algorithm that encrypts a piece of data (e.g. age) utilising zero-knowledge to ensure nothing is disclosed during the encryption. I found that mapping out the data flow before I coded it, helped me to plan for any tricky coding elements such as where I would need to put a conditional statement or a function.

Development Phase – Model Development Steps

In this section of the Iteration 1 Development Phase, you will find the steps it took to develop a basic ZKP algorithm to encrypt some sort of input data. The following steps are important in this development process.

1. Imports:

```
1 import hashlib;
2 import passlib.hash;
3 import sys;
```

Fig.5

These lines (1-3) import the necessary modules (hashlib, pass.hashlib and sys) for cryptographic hashing and system-related operations (Fig. 5)

2. Initialization:

```
5 # Verifier does not know the age
6 age_actual = 14
7 age_to_prove = 13
```

Fig.6

Here, *actual_age* represents the actual age, and *age_to_prove* represents the age that needs to be proven. E.g. you are 14 years old, but you want to prove you are at least 13 years or older. (Fig.6)

3. Seed and Proof initialization:

```
9  # This is seed is a random value generated by a Third Party, to ensure security
10 seed = "12345678910111213141516171819 20"
11
12 # This is a digital signature proof by Trusted Entity
13 proof = hashlib.sha256(seed.encode()).hexdigest()
14 print("Proof is " + proof)
15
16 # Proof and Encrypted have different values for distinct purposes
17 encrypted_age = hashlib.sha256(seed.encode()).hexdigest()
18 print("Encrypted Age (initialized with seed) is " + encrypted_age)
```

Fig.7

The *seed* is a random value used for cryptographic operations. *proof* is initialized by hashing the *seed* using SHA-256. SHA-256 (Secure Hash Algorithm 256-bit) is a cryptographic hash function that generates a fixed-size 256-bit (32-byte) hash value, providing a highly secure and unique representation of input data. The values of *proof* and *encrypted_age* are printed out. (Fig. 7)

4. Updating proof:

```
# If b < a, we don't loop, so we don't hash
a = 1
b = 1 + age_actual - age_to_prove
for i in range(a, b):
    proof = hashlib.sha256(proof.encode()).hexdigest()
```

Fig.8

proof is updated by hashing it multiple times based on the difference between *actual_age* and *age_to_prove*, in this case, is 1. This value is also printed out. (Fig.8)

5. Updating Encrypted Age:

```
# Prover hashes age_actual times his age to secure it
c = 1
d = age_actual + 1
for i in range(c, d):
    encrypted_age = hashlib.sha256(encrypted_age.encode()).hexdigest()
print("New Encrypted Age is " + encrypted_age)
```

Fig.9

Here, *encrypted_age* is updated by hashing it multiple times based on *actual_age*, in this case is 14. (Fig. 9)

6. Verifying Age:

```
# Verified Age will be Hashed age_to_prove times
original_proof = proof
verified_age = proof
for i in range(0, age_to_prove):
    verified_age = hashlib.sha256(verified_age.encode()).hexdigest()
```

Fig.10

The code verifies the age by repeatedly hashing *verified_age* using SHA-256 *age_to_prove* times. (Fig. 10)

```
# We check if Encrypted Age (By Prover) and Verified_Age (By Verifier)
if (encrypted_age == verified_age):
    print("You have proven your age!")
else:
    print("You have NOT proven your age!")
```

Fig.11

Finally, it checks is *encrypted_age* matches *verified_age* and print out the result based on if its true or not. All of the other values are also printed out. (Fig. 11)

Testing Phase

```
Proof is ac0f6c5ad9f795ea5b958fedb929eec2201bbb7af8b65d393d7efe51275ae254
Encrypted Age (initialized with seed) is ac0f6c5ad9f795ea5b958fedb929eec2201bbb7af8b65d393d7efe51275ae254


New Proof is 7b66e8cfe268191daf348c59138a36c977a01d4187c10de47decd4dd7694f935
New Encrypted Age is 8461e3a101d583335acbda82b9e0cd24b051cde7f8f3c79cb196b8b0a075f1dc


Verified Age is 8461e3a101d583335acbda82b9e0cd24b051cde7f8f3c79cb196b8b0a075f1dc
Actual Age:             14
Age to prove:           13
....
Proof:          7b66e8cfe268191daf348c59138a36c977a01d4187c10de47decd4dd7694f935
Encrypted Age:  8461e3a101d583335acbda82b9e0cd24b051cde7f8f3c79cb196b8b0a075f1dc
Verified Age:   8461e3a101d583335acbda82b9e0cd24b051cde7f8f3c79cb196b8b0a075f1dc


You have proven your age!
```

Fig.12

Above is what the output of the ZKP code is (Fig. 12). You can see that if the encrypted age matches the verified age then the ZKP evaluates successful. I got parts of this code from GitHub, but it wasn't entirely working so I had to modify parts of it.

After trying it with different values, it overall worked fairly well. I had to start thinking about incorporating an algorithm similar to this but with a proper dataset and link it to a linear regression AI model. This ZKP algorithm encrypts the data using SHA-256 hashing, and with a linear regression model I wasn't sure how to make that work. That was one of the big factors I had to think about for the next iteration taking that into consideration.

Overall, this iteration was mainly about getting myself comfortable and explore a few different ways to use ZKPs in Python. Once I was getting good results from the code, I was ready to move on and start developing the algorithm further.

Iteration 2:

Planning Phase

My plan for iteration 2 was to start developing the real algorithm and try implement it into a simple linear regression model as well. My main focus was to get the architecture correct before I start experimenting with adding it into a Flask GUI or using different datasets. I aimed to just focus on the algorithm and linear regression because I knew this would be the most difficult and time consuming part of the development.

Design Phase

After some research, I realised that it can be extremely difficult to implement ZKPs into the encryption of data fir AI models on a simple level, for various reason that impacted my design choices:

- Complexity of ZKP Protocols: ZKPs involve complex cryptographic protocols such as zk-SNARKs or zk-STARKs (Zero-Knowledge Scalable Transparent Arguments of Knowledge). Implementing these protocols correctly requires a deep understanding of cryptography and mathematical concepts, which can be challenging for developers who are not experts in these areas.
- Integration with AI Models: Integrating ZKPs into existing AI models and frameworks can be non-trivial. AI models typically require access to raw data for training and inference, but ZKPs aim to prove properties about the data without revealing it. Balancing data privacy with AI model functionality and performance adds complexity to the integration process.

- Data Preprocessing and Representation: ZKPs usually operate on specific data representations or transformations to enable zero-knowledge proofs. Preprocessing data and representing it in a format suitable for ZKPs can be challenging and may require careful design to ensure both privacy and utility.
- Security and Trust Assumptions: Implementing ZKPs for data privacy requires strong security and trust assumptions. Any vulnerabilities or flaws in the ZKP implementation could compromise data privacy and lead to unintended data disclosure, highlighting the importance of rigorous security practices.

Taking these factors into account, I decided to take a different approach to this (Fig. 13).

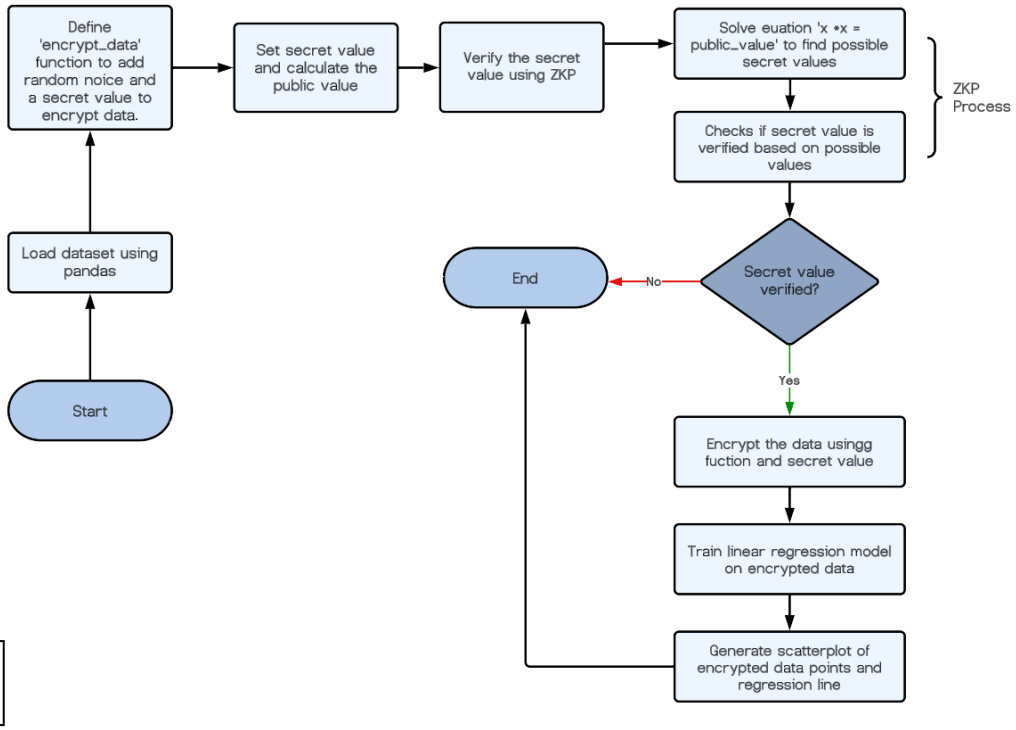


Fig. 13

Development Phase

The development phase during this iteration was particularly important. Following the flow chart above, I was able to create a working encryption algorithm along with a simple linear regression model.

1. Import Libraries:

```python
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sympy import symbols, Eq, solve
import random
```

Fig. 14

Import the necessary libraries (numpy, pandas, sklearn, sympy, and random) (Fig. 14)

2. Read Data and Define Values

```python
# Read data from CSV file using pandas
df = pd.read_csv('6april.csv')

# Define the secret value and a related public value
secret_value = 5.67
public_value = secret_value * secret_value
```

Fig. 15

Next, it reads data from a csv file (in this case, a csv with two columns: house size and price) into a DataFrame *df* using pandas. After, the secret value *secret_value* is defined and calculates the related *public_value* for the ZKP. I created this csv file with only a few rows of data, and it was only used for testing purposes. (Fig. 15)

3. ZKP setup

```python
# Define symbols for the zero-knowledge proof
x = symbols('x')

# Define the equation for the zero-knowledge proof
eq = Eq(x * x, public_value)

# Solve the equation to find possible secret values
possible_secret_values = solve(eq, x)
```

Fig. 16

Now it defines a symbol *x* for the ZKP. It sets up the equation *eq* for the ZKP as '$x * x = public\_value$' ($x^2 = public\ value$). It then solves the equation to find possible secret values. (Fig. 16)

4. Verification of ZKP

```python
# Verify the zero-knowledge proof
verified = False
for val in possible_secret_values:
    if val == secret_value:
        verified = True
        break
```

Fig. 17

It iterates through the *possible_secret_values* to verify is any matches the *secret_value*. Next, its sets *verified* based on whether the proof is successful or not. (Fig. 17)

5. Model Training (if verified)

```
if verified:
    # Updated encrypt_data function with random noise
    def encrypt_data(data):
        encrypted_data = []
        for x in data:
            noise = random.uniform(-0.5, 0.5)  # Generate random noise between -0.5 and 0.5
            encrypted_value = int(x[0]) + secret_value + noise  # Add noise to the encrypted value
            encrypted_data.append([encrypted_value])
        return encrypted_data

    # Decrypt the data after model training
    def decrypt_data(data):
        if isinstance(data, float):
            return data - secret_value  # Handle single value
        else:
            return [x - secret_value for x in data]  # Handle list of values
```

Fig. 18

If the proof is verified, then it can proceed to the model training. It first defines an updated *encrypt_data* function to add random noise to data for encryption. Next it defines a *decrypt_data* function to decrypt data after model training. (Fig. 18)

```
    # Mask the input data using a more secure encryption (with random noise)
    X_encrypted = encrypt_data(df[['size']].values.tolist())

    # Print the encrypted data
    print("Encrypted data:")
    print(X_encrypted)

    y = df['price']

    # Train a linear regression model using the encrypted dataset
    model = LinearRegression()
    model.fit(X_encrypted, y)

    # Print the coefficients of the trained model
    print("Trained model coefficients:", model.coef_[0])

    # Predict prices for new house sizes (example)
    new_sizes_encrypted = encrypt_data([[1800], [2200]])  # Encrypt new data
    predicted_prices_encrypted = model.predict(new_sizes_encrypted)
    predicted_prices = decrypt_data(predicted_prices_encrypted)  # Decrypt predictions
```

Fig. 19

The input data, in this case is house size (*x* or independent variable) is encrypted using the *encrypt_data* function. A simple linear regression model is trained on the encrypted dataset and the coefficients are printed out. Next, new data is encrypted to predict prices using the trained model and then they are decrypted after they have been predicted. (Fig. 19)

6. Output

```
    # Print the encrypted predictions
    print("Predicted prices for new house sizes (encrypted):")
    print(predicted_prices_encrypted)

    # Print the decrypted prices
    print("Predicted prices for new house sizes (decrypted):\n", predicted_prices)

else:
    print("Cannot proceed with model training due to failed zero-knowledge proof.")
```

Fig. 20

The encrypted data, model coefficients, encrypted predictions and decrypted predications are printed based on the process. If the ZKP is unsuccessful, the model wont train and encrypt the data. (Fig. 20)

In the code, the ZKP is used to verify a secret value *secret_value* without revealing it directly. The ZKP involves setting up an equation where *x* represents a potential secret value, and *public_value* is a known value derived from the secret value. It then solves this equation to find possible secret values. This ZKP benefits the rest of the code by ensuring that sensitive information, represented by the *secret_value*, is securely validated before proceeding with critical operations like model training and predictions.

By verifying the *secret_value* without revealing it directly, the ZKP adds a layer of security and privacy to the process. This is particularly important in scenarios involving encrypted data or confidential computations where the actual values must remain hidden but still need to be validated for correctness. The successful verification of the ZKP (i.e., *verified* is True) guarantees that subsequent operations can be safely executed, such as encrypting and decrypting data, training machine learning models, and making predictions, with confidence in the integrity and authenticity of the secret value without exposing it unnecessarily. Thus, the ZKP enhances the overall security and trustworthiness of the code's functionality involving sensitive information handling.

```
Zero-knowledge proof verified successfully.
Encrypted data:
[[1005.76333000478], [1505.2505086964336], [2005.8451713061581], [2505.4072677012855], [3005.193362047414], [3505.88746821807], [4005.769281783408], [4505.2866
752073105], [5006.031297609827]]
Trained model coefficients: 199.65644277551928
Predicted prices for new house sizes (encrypted):
[409352.97602818 489250.75963146]
Predicted prices for new house sizes (decrypted):
 [409347.30602817825, 489245.0896314629]
```

Fig. 21

This method incorporating a ZKP worked a lot better. The code from iteration 1 would've been difficult to incorporate into linear regression because it hashes each data input, which can be a lot trickier to make predictions based on if its not entirely numerical data. As well as the ZKP verification, it adds random noise to each piece of data, increasing its overall security once it's gone through the AI model. There is also a decrypt function, so the data can be previewed as both encrypted and decrypted in the output. (Fig. 21)

Testing phase

Testing in iteration 2 was important as I had added new functions and features to the code. Like iteration 1, I tested each function and ensured they were giving the correct outputs. I experimented with the code by changing the different values involved in the ZKP and some of the other values. I used a small, simple dataset with two columns for now, and for the test data I used two values that aren't included in this csv. (Fig. 22)

	size	price
1	size	price
2	1,000	250000
3	1,500	350000
4	2,000	450000
5	2,500	550000
6	3,000	650000
7	3,500	750000
8	4,000	850000
9	4,500	950000
10	5,000	1050000
11		

Fig. 22

Everything was working quite well during this iteration and, I figured that next I should focus on some of the visual aspects and possibly using a different dataset. The regular Thonny/VS Code interface didn't help with the visual aspects either so I decided that in the next iteration I could try incorporate a simple GUI.

Iteration 3:

Planning phase

For the last iteration, I needed to focus on completing a few main things. I needed to think a method to create a GUI and develop it. Although it was not the main focus of my project, I decided to add a bit of CSS to make the GUI look cleaner and more organised. Adding extra CSS or HTML wouldn't be too difficult as a few of my past projects involved a lot of CSS/HTML.

I also planned to train the AI on a proper dataset. The program works well on a small dataset so using a larger dataset shouldn't cause too many problems. Additionally, If there is any parts of the code that need to be modified then I would complete that in this iteration.

Design phase

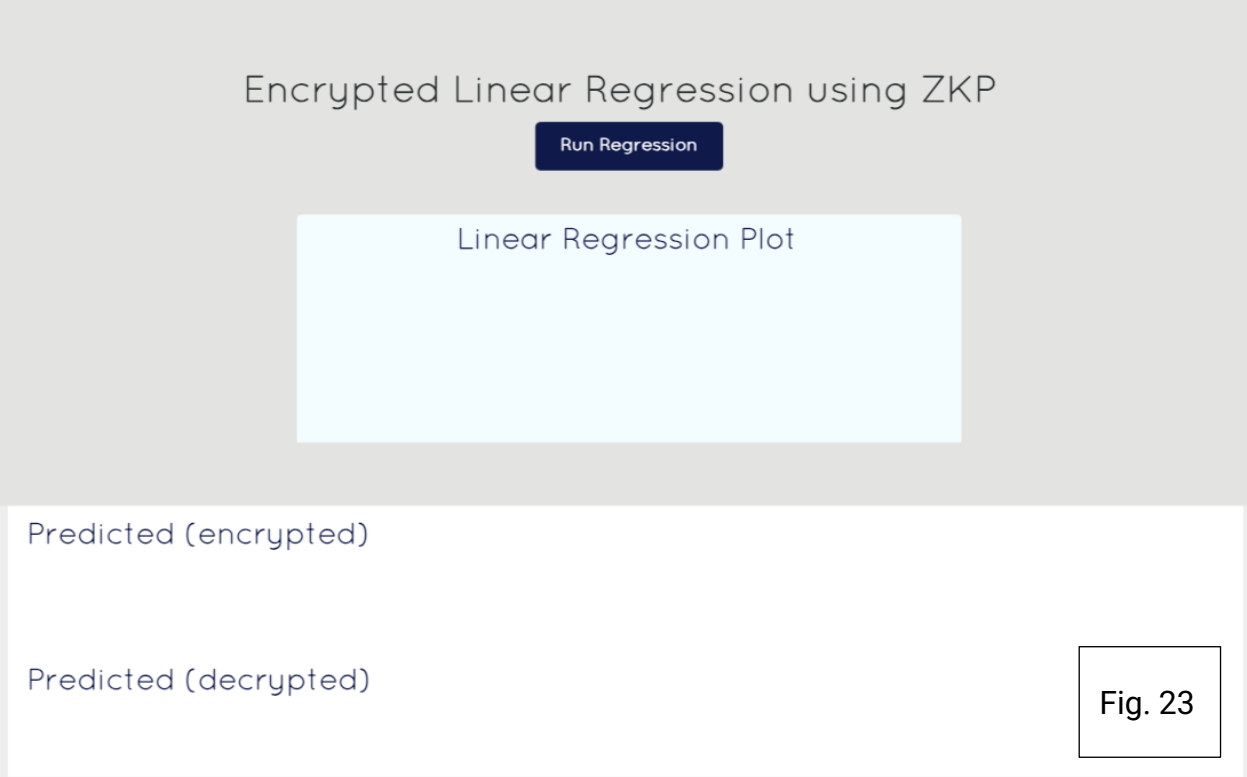


Fig. 23

I designed a simple wireframe of what I wanted the GUI to look like (Fig. 23) if I could achieve that level of customisation. I wasn't entirely sure if I would have the time or if I would be able to get something like that. If I ended up being able to get a GUI that looks similar, I would additionally change things like the colour and font styles. Again, this was not a main component of my project.

Development phase

In one of my recent projects, I initially went with Tkinter for a GUI. Tkinter is a Python library used for creating graphical user interfaces (GUIs) with widgets like buttons, labels, and entry fields, making it easier to build interactive applications. Tkinter doesn't allow for a deep level of customization, and it's also caused problems for me in the past. After some thinking I disregarded the Tkinter GUI approach due to time constraints and focused on creating a simple Flask based GUI interface (Fig. 24). I had initially thought about using a Django based GUI, but my teacher, concerned for the time necessary to become competent in Django, suggested that I try an online course in beginner web applications using Flask. I used this tutorial to help (Loeber 2021).

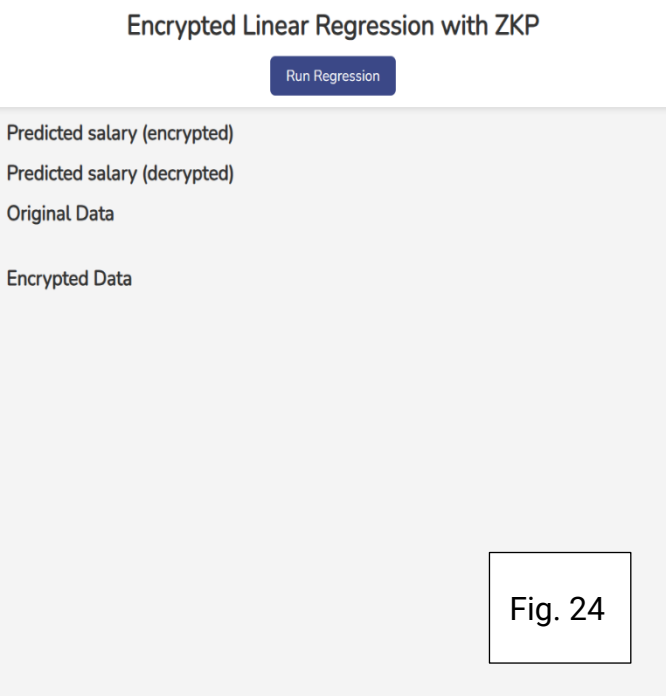


Fig. 24

Using Flask allowed me to use my HTML/CSS/JavaScript coding experience once I was able to deploy the Flask module.

Flask also allowed me to incorporate the matplotlib scatterplot (Fig. 25).

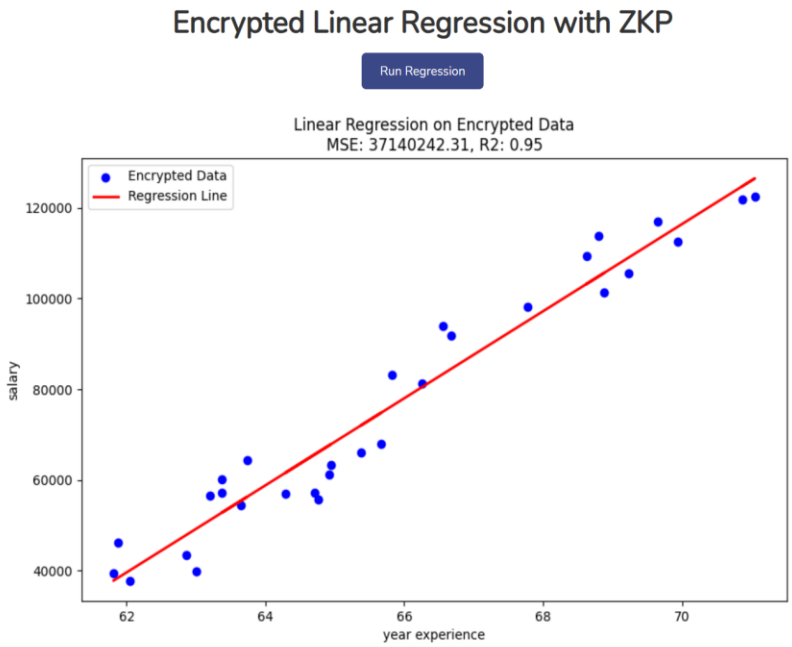


Fig. 25

I was also able to display some of the predicted outputs both encrypted and decrypted on the same page. (Fig. 26)

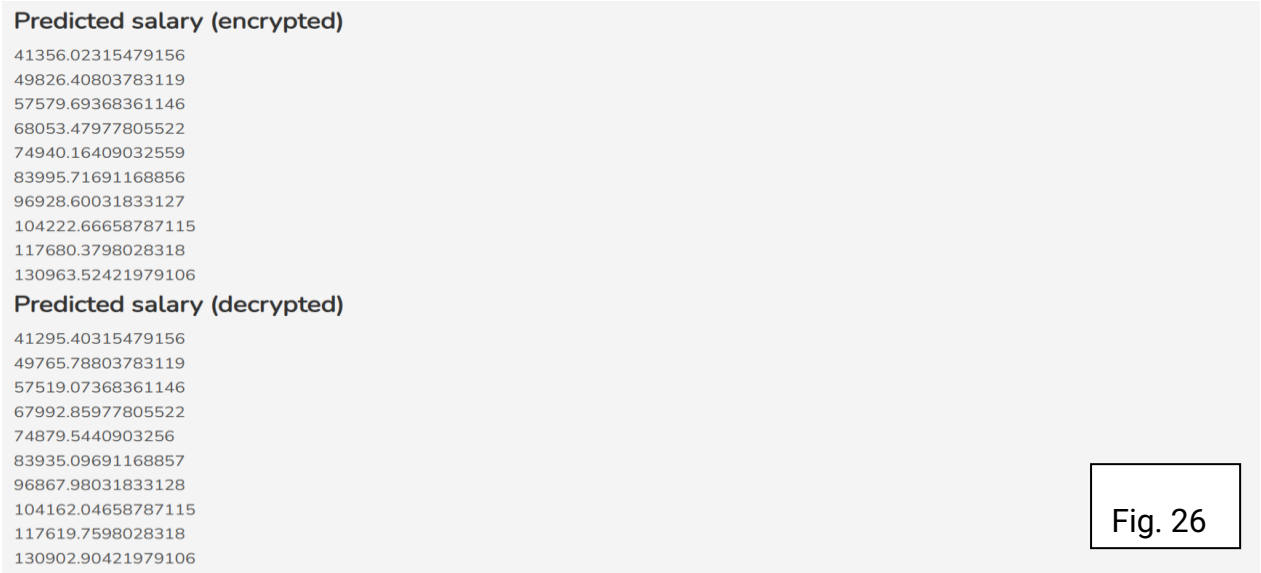


Fig. 26

This involved some simple HTML where it creates a list `<*ul*>` of the encrypted predicted salaries. It uses a loop (line 26, 32) to go through each encrypted salary (line 27, 33) and display it on the screen. (Fig. 27)

```
23  <div class="data">
24    <h2>Predicted salary (encrypted)</h2>
25    <ul>
26        {% for item in predicted_sal_e %}
27        <p>{{ item }}</p>
28        {% endfor %}
29    </ul>
30    <h2>Predicted salary (decrypted)</h2>
31    <ul>
32        {% for item in predicted_sal %}
33        <p>{{ item }}</p>
34        {% endfor %}
35    </ul>
```

Fig. 27

Once the Flask GUI was built and working with the Python code, I focused on the final few bits with the AI and ZKP. I needed to find and train the AI on a more realistic dataset and adjust the code in any way that might fit more appropriately with the dataset.

My next step was to find a dataset to train the AI on. I searched on Kaggle to find a simple dataset that I could use for my project. I settled on a simple Salary Dataset (Abhishek 2023) which can be used for linear regression, as it only has two columns, which is perfect for implementing it into my code. I wanted to choose a dataset that contained some sort of data that would be considered ‘sensitive’ in real world scenarios.

Sensitive data refers to any information that, if disclosed or accessed without authorization, could result in harm, discrimination, or privacy violations for individuals or organizations. This can include personal information such as financial data, health records, biometric data, and confidential business information.

In the context of AI, sensitive data is crucial because AI systems often rely on vast amounts of data to learn and make decisions. Handling sensitive data in AI involves ensuring proper security measures, ethical considerations, and legal

compliance to protect individuals' privacy and prevent misuse or unauthorized access. I chose this salary dataset because it is an example of financial data, which in real life would be considered sensitive.

Luckily enough, it was a small enough dataset, so I didn't have to do any data cleaning as there weren't any NaN or null columns. Although, the training data (Fig. 28) was the only csv that came with the file, so I had to create my own test data (Fig. 29) for the model to make predictions based on.

	YearsExpe	Salary
0	1.2	39344
1	1.4	46206
2	1.6	37732
3	2.1	43526
4	2.3	39892
5	3	56643
6	3.1	60151
7	3.3	54446
8	3.3	64446
9	3.8	57190
10	4	63219

Fig. 28

	YearsExpe	Salary
0	1.9	43567
1	2.5	52787
2	3.5	63993
3	4.5	73992
4	5.5	83123
5	6.5	94783
6	7.5	105793
7	8.5	116783
8	9.5	128923
9	10.5	139872

Fig. 29

I updated the code to read in these two csv viles using pandas (Fig. 30).

```
17  # Load datasets
18  df = pd.read_csv('Salary_dataset.csv')
19  df1 = pd.read_csv('Salary_dataset_test.csv')
```

Fig. 30

There were a few more places I had to update so that it would encrypt the correct columns of data. Below shows that once the proof is verified, the training data is encrypted and the model is able to train (Fig. 31).

```
75      if verified:
76          # Encrypt the training data
77          X_encrypted = encrypt_data(df[['YearsExperience']].values.tolist(), secret_value)
78          y = df['Salary']
79
80          # Train the linear regression model
81          model = LinearRegression()
82          model.fit(X_encrypted, y)
83
84          # Encrypt the test data
85          test_data = df1[['YearsExperience']].values.tolist()
86          new_years_encrypted = encrypt_data1(test_data, secret_value)
```

Fig. 31

Another area I had to update was where once the *YearsExperience* column in the test data is encrypted, it will be able to predict the salaries and produce the MSE and R-squared. (Fig. 32)

```
88          if new_years_encrypted:
89              # Predict salaries for new data
90              predicted_salary_encrypted = model.predict(new_years_encrypted)
91              predicted_salary = decrypt_data(predicted_salary_encrypted)  # Decrypt predictions
92
93              # Calculate Mean Squared Error (MSE) and R-squared (R2 score)
94              mse = mean_squared_error(y, model.predict(X_encrypted))
95              r2 = model.score(X_encrypted, y)
96
97              # Generate the plot
98              plt.figure(figsize=(10, 6))
99              # Update plot title with MSE and R2 score
100             plt.title(f'Linear Regression on Encrypted Data\nMSE: {mse:.2f}, R2: {r2:.2f}')
101             plt.scatter([x[0] for x in X_encrypted], y, color='blue', label='Encrypted Data')
102             plt.plot([x[0] for x in X_encrypted], model.predict(X_encrypted), color='red', linewidth=2, label='Regression Line')
103             plt.xlabel('year experience')
104             plt.ylabel('salary')
105             plt.legend()
```

Fig. 32

Once the dataset was in, I ran it a few times to make sure that it was working properly. Below is an image of a scatterplot produced by the model. (Fig. 33)

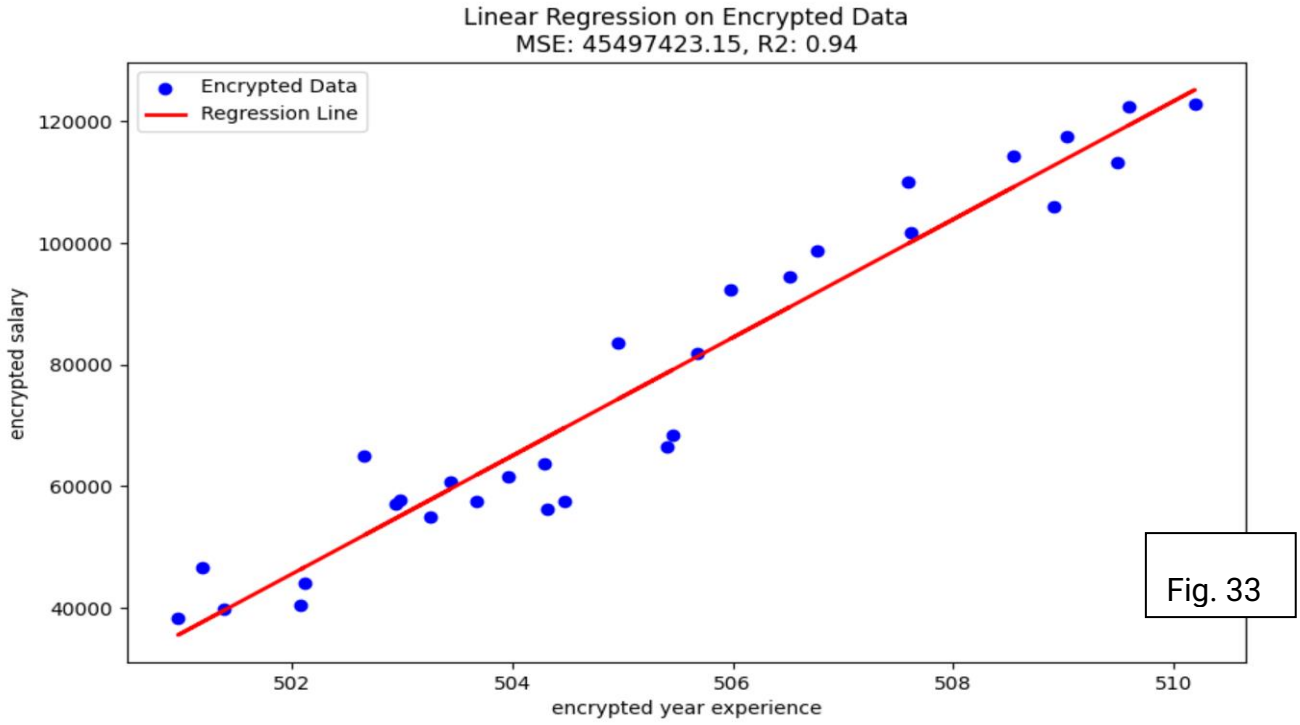


Fig. 33

Additionally, I wanted there to be some representation of the encrypted values on the HTML. I created a simple table to display the predicted salary encrypted, predicted salary decreypted (test data) as well as the years of experience encrypted and decrypted (test data). (Fig. 34)

```
<div class="data">
    <h2>Data Overview</h2>
    <table>
      <tr>
        <th>Predicted Salary (Encrypted)</th>
        <th>Predicted Salary (Decrypted)</th>
        <th>Test Data (Decrypted)</th>
        <th>Test Data (Encrypted)</th>
      </tr>
      {% for index in range(predicted_sal_e|length) %}
      <tr>
        <td>{{ predicted_sal_e[index] }}</td>
        <td>{{ predicted_sal[index] }}</td>
        <td>{{ original_data[index] }}</td>
        <td>{{ encrypted_data[index] }}</td>
      </tr>
      {% endfor %}
    </table>
  </div>
```

Fig. 34

This is what the code displays on the GUI. (Fig.35)

Data Overview

Predicted Salary (Encrypted)	Predicted Salary (Decrypted)	Test Data Years (Decrypted)	Test Data Years (Encrypted)
46548.59354917705	45662.18354917705	[1.2]	[887.768396324284]
48595.53968119994	47709.129681199935	[1.4]	[887.340206218319]
64195.52811775729	63309.11811775729	[1.6]	[887.0935422024731]
68800.1172472816	67913.70724728159	[2.1]	[888.6983351367086]
84292.38214113005	83405.97214113004	[2.3]	[888.0810631519189]
93847.8737158645	92961.46371586449	[3.0]	[889.694555649598]
96275.69433411583	95389.28433411583	[3.1]	[889.6459010386034]
108600.01767147705	107713.60767147705	[3.3]	[889.6630602840149]
118624.83867330477	117738.42867330476	[3.3]	[889.3419113101362]
131229.43064370193	130343.02064370192	[3.8]	[889.849015662878]

Fig. 35

Mathematical Interpretation of the Data Encryption:

I've attempted to show an example of how the encryption process works using the training data in the code. I've already explained the maths behind the ZKP process in the explanations of components and terms.

Example: I'll use the first row from the training data:

$$\text{YearsExperience} = 1.2$$

$$\text{Salary} = 39344.0$$

For this example, lets assume the secret value from the ZKP is 100:

$$secret\_value = 100$$

Random noise is added to the data during the encryption process. Lets assume that:

$$\text{Noise} = 0.1$$

In the actual code, the noise is any random float decimal between -0.5 to 0.5.

```
noise = random.uniform(-0.5, 0.5)  # Generate random noise between -0.5 and 0.5
```

Additionally, the secret value is any random float decimal (to 2 decimal places) between 0 and 1000.

```
secret_value = round(random.SystemRandom().uniform(0, 1000), 2)
```

In the actual code, the noise term is randomly generated for each data point, for added security.

$$\text{Encrypted Salary} = \text{Salary} + secret\_value + \text{Noise}$$

$$\text{Encrypted Salary} = 39344.0 + 100 + 0.1$$

$$\therefore \quad \text{Encrypted Salary} = 39444.1$$

So, the first row in the training data:

- Original Salary: 39344.0
- Encrypted Salary: 39444.1 (after encryption)

This process ensures that the salary data is encrypted before being used for training the linear regression model. Decrypting the data involves reversing this process, where the secret value is subtraccted to retrieve the original data.

Mathematical Interpretation of the LR Model:

I've attempted to show how the maths works to predict values with the decrypted (original) data below (Gurjot 2024). The encrypted values change each time the code is run due to the ZKP and random noise added, so its more difficult to calculate.

The training data is broken down into two variables:

- X: Years of Experience (input feature)

- y: Salary (target variable)

Linear regression aims to model the relationship between the input feature (X) and the target variable (y) using a linear equation of the form: $y = mx + b$ where:

- m is the slope (coefficients) of the line.
- b is the intercept (bias) of the line.

We can denote the training data as (X_i, y_i) for $i = 1,2, \dots, n$, where n is the number of data points.

The linear regression model seeks to minimize the Mean Squared Error (MSE) between the actual target values and the predicted values by the model:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - (mx_i + b))^2$$

To minimize MSE, we differentiate it with respect to m and b and set the derivatives to zero to find the optimal values for m and b.

The optimal coefficients m and b can be calculated using these formulas:

$$m = \frac{n(\sum_{i=1}^{n} x_i y_i) - (\sum_{i=1}^{n} x_i)(\sum_{i=1}^{n} y_i)}{n(\sum_{i=1}^{n} x_i^2) - (\sum_{i=1}^{n} x_i)^2}$$

$$b = \frac{\sum_{i=1}^{n} y_i - m(\sum_{i=1}^{n} x_i)}{n}$$

Once we have the coefficients m and b, we can predict the salary for neaw years of experience (X_{new}) using the equation:

$$\text{Predicted Salary} = m \times X_{new} + b$$

To do this we need to calculate several sums:

- $\sum_{i=1}^{n} x_i$: Sum of all years of experience
- $\sum_{i=1}^{n} y_i$: Sum of all salaries

- $\sum_{i=1}^{n} x_i^2$: Sum of all squares of years of experience
- $\sum_{i=1}^{n} x_i\, y_i$: Sum of the product of years of experience and salaries
- n: Number of data points

Calculations:

- $n = 30$
- $\sum_{i=1}^{30} x_i = 76.1$
- $\sum_{i=1}^{30} y_i = 1669780.0$
- $\sum_{i=1}^{30} x_i^2 = 119.71$
- $\sum_{i=1}^{30} x_i y_i = 4580197.0$

Now we can substitute these values for the formulas *m* and *b*:

$$m = \frac{30(4580197.0) - (76.1)(1669780.0)}{30(119.71) - (76.1)^2}$$

$$b = \frac{1669780.0 - m(76.1)^2}{30}$$

These formulas give us:

$$m \approx 9449.962$$

$$b \approx 25792.200$$

Therefore, the LR equation for this data is approximately:

$$\text{Salary} = 9449.962 \times \text{Years of Experience} + 25792.200$$

Now that we have these coefficients, we can use them to predict salaries for the test data or any new data points. Using that equation, we can plug in the *YearsExperience* values from the test data to predict the values.

Example: $\text{YearsExperience} = 1.9$

$$\text{Salary} = 9449.962 \times 1.9 + 25792.200$$

$$\text{Salary} \approx 42511.082$$

Here, I've created a table showing the prediction for each *YearsExperience* in the test data:

YearsExperience	**Calculation**	**Predicted Salary**
1.9	$9449.962 \times 1.9 + 25792.200$	≈ 43747.1278
2.5	$9449.962 \times 2.5 + 25792.200$	≈ 49417.105
3.5	$9449.962 \times 3.5 + 25792.200$	≈ 58867.067
4.5	$9449.962 \times 4.5 + 25792.200$	≈ 68317.03
5.5	$9449.962 \times 5.5 + 25792.200$	≈ 77766.991
6.5	$9449.962 \times 6.5 + 25792.200$	≈ 87216.952
7.5	$9449.962 \times 7.5 + 25792.200$	≈ 96666.913
8.5	$9449.962 \times 8.5 + 25792.200$	≈ 106116.874
9.5	$9449.962 \times 9.5 + 25792.200$	≈ 115566.835
10.5	$9449.962 \times 10.5 + 25792.200$	≈ 121916.796

Its important to keep in mind that these calculations are done from scratch and some of the values are rounded, whereas in the code its possible they have a larger decimal. Keeping these two things in mind, my calculations might **not be as accurate** as the code because I've done them based on the rounded values in the formulas.

Testing phase

Once I had the real dataset complete, I completed my last testing of the model. There were a few small problems, for example if something in the python didn't match up with the HTML, it wouldn't show up on the screen. I had to be careful

when adding in features like this to go and check to make sure that they are both connected from each file.

The new salary predictions dataset worked quite well, additionally, the test dataset also worked very well. Although the dataset it very small, I'm glad that it worked in the linear regression and salary values are able to be predicted based on the input.

Results and graphs produced using the final version can be view in the results section. The link to the GitHub Repository with all of the code is located in the Appendices.

Results

Evaluating the performance of the linear regression model and the security of the ZKP are very important and crucial steps. I performed a few tests on the ZKP functions which can be viewed below:

```python
def verify_zkp(secret_value, public_value):
    #Verify zero-knowledge proof.
    # Simulate the verification process (example equation x * x = public_value)
    if secret_value ** 2 == public_value:
        return True
    else:
        return False

# Test the zero-knowledge proof verification function
def test_zkp_verification():
    # Test valid proof
    secret_valid = random.randint(1,100000)
    public_valid = secret_valid ** 2
    assert verify_zkp(secret_valid, public_valid) == True, "Valid proof failed"

    # Test invalid proof
    secret_invalid = 7
    public_valid = secret_valid ** 2  # Use the correct public value
    assert verify_zkp(secret_invalid, public_valid) == False, "Invalid proof passed"



    print("All zero-knowledge proof verification tests passed.")

# Run the test function
test_zkp_verification()
```

Fig. 36

This code (Fig. 36) defines a ZKP verification function *verify*_zkp and a test function *test_zkp_verification* to validate the ZKP process. The verify function checks if a given *secret_value* when squared matches a corresponding *public_value*, simulating a basic ZKP equation. The test verification function tests this verification process by generating random secret values and their correct public values to validate successful proofs. It also includes a test case with intentionally incorrect input to ensure failed verification. Assertions are used to verify the expected outcomes of these tests, and if all tests pass, a success message is printed, indicating that the ZKP verification tests have succeeded.

```
All zero-knowledge proof verification tests passed.
```

This next test (Fig. 37) measures the time taken to perform ZKP verification 10,000 times, helping assess the function's efficiency. While this test isn't the most important in my situation, it's helpful when there's a more advanced ZKP algorithm.

```python
import time

def test_performance():
    start_time = time.time()
    for _ in range(10000):  # Perform verification 10,000 times
        secret = random.randint(1, 100000)
        public = secret ** 2
        verify_zkp(secret, public)
    end_time = time.time()
    print(f"Performance test completed in {end_time - start_time} seconds.")

test_performance()
```

Fig. 3

```
All zero-knowledge proof verification tests passed.
Performance test completed in 0.00823664665222168 seconds.
```

On the top of the scatterplots produced by the linear regression model, you can see the Mean Squared Error (MSE) and the R-squared (R^2) value. MSE is a measure of the average squared difference between actual and predicted values in a regression model. It quantifies the overall accuracy of the model's predictions, with lower MSE values indicating better fit to the data. R^2 value, also known as the coefficient of determination, measures the proportion of variance in the dependent variable that is explained by the independent variables in a regression model. It ranges from 0 to 1, where 1 indicates a perfect fit where the model explains all the variability, and 0 indicates that the model does not explain any variability beyond the mean.

Linear Regression on Encrypted Data
MSE: 47225609.41, R2: 0.93

The MSE value of 47225609.41 suggests that, on average, the model's predictions deviate by this amount from the actual salary values. In simpler terms, it tells us how "wrong" the model's predictions are from the actual values, considering both the size of the errors and their positive or negative nature. Lower

MSE values indicate that the model's predictions are closer to the actual values, which is what we aim for in a good predictive model.

The high R2 value of 0.93 indicates that a significant portion of the variability in salaries is explained by the linear relationship with years of experience. This suggests that the model captures the underlying patterns well and is a good predictor of salaries based on experience.

These values might not be the most accurate in this case, as the model has had little training on the data, and it makes predictions at a very basic level. As I've said previously, my main aim in this project was to focus on the ZKP and encryption aspect, and the AI was an example of how the data can be manipulated after its been encrypted.

Looking back at my hypothesis, my goal was to develop a ZKP that will be able to encrypt data ensuring zero-knowledge properties, and a simple Linear Regression model will be able to train on that encrypted data, with no disclosure or access to the original data. I believe I have achieved the brief of what I have set myself to, and with more time and further research I believe that this project can go a lot further.

Conclusions

There are several reasons as to why ZKPs can be extremely useful in the world of AI and why they should really be considered:

1. Privacy Preservation: ZKPs allow for computations to be performed on encrypted data without revealing the data itself. This is crucial for protecting sensitive information, such as personal data in healthcare or financial records.
2. Data Security: By using ZKPs, AI systems can access data securely without the risk of data breaches or leaks. Encrypted data can be used in computations without exposing it to potential attackers.
3. Compliance with Regulations: Many regulations, such as GDPR in the EU or HIPAA in the US, require strict data privacy measures. ZKPs can help AI systems comply with these regulations by ensuring that sensitive data remains confidential.
4. Enhanced Collaboration: ZKPs enable parties to collaborate and share data without actually sharing the underlying data itself. This can foster collaboration in research, healthcare, and other fields where data sharing is critical, but privacy concerns are paramount.
5. Increased Trust: With ZKPs, individuals and organizations can trust that their data is being used appropriately and securely. This can lead to greater acceptance and adoption of AI technologies that rely on sensitive data.
6. Improved AI Model Training: AI models often require large amounts of data for training. ZKPs can enable secure access to diverse datasets without centralizing or exposing the data, leading to more robust and accurate AI models.

These advantages highlight how ZKPs can play a pivotal role in unlocking the full potential of AI while addressing privacy and security concerns in data-driven applications. Although, there can be some disadvantages that come along with it:

1. Complexity and Resource Intensive: Implementing ZKPs can be complex and resource-intensive, requiring specialized cryptographic knowledge and computational power. This can increase development time and costs.
2. Performance Overhead: ZKPs often introduce performance overhead due to the computational burden of generating and verifying proofs. This can impact the speed and efficiency of AI algorithms, especially in real-time applications.
3. Trust in Implementation: The security of ZKPs relies heavily on the correct implementation of cryptographic protocols. Any vulnerabilities or flaws in the implementation can compromise the entire system's security and privacy guarantees.
4. Potential for Misuse: ZKPs can also be used in malicious ways, such as masking illegal activities or facilitating illicit transactions. Balancing privacy with the need for accountability and transparency is a delicate issue.
5. Regulatory and Compliance Challenges: Compliance with data protection regulations and industry standards can be more complex in ZKP-based systems. Ensuring that ZKPs align with legal requirements without compromising privacy is an ongoing concern.
6. Compatibility Issues: Integrating ZKP technologies with existing systems or platforms may face compatibility challenges. Ensuring seamless interoperability with legacy systems or third-party services can be non-trivial.

While ZKPs offer compelling privacy and security benefits, careful consideration of these disadvantages is essential for their successful adoption and deployment in AI and other applications.

My Improvements

There are several issues that, mainly given time and research, would improve both the ZKP and AI aspects of my project.

My ZKP algorithm obviously isn't the most secure and is only used on a very basic level. I believe that with further research and a lot more time, that I could develop a proper ZKP system that would completely prevent the disclosure of data to AI models.

With using the Flask GUI, which means the interface is run as a local host on my computer. This is not necessarily the best option as I cannot easily get people to beta test my project unless they run the whole program on their device, which is not efficient. Beta testing is really important when creating any GUI or front end.

To show that I can manipulate the encrypted data, I created a very simple linear regression model. Obviously, with more time I would be able to develop the LR model to be a lot more accurate. Additionally, I would love to see ZKPs being used in other areas of machine learning, particularly it would be amazing in the area of deep learning and GANs so that the model would learn and improve over time.

With a lot more time and development, my project could improve significantly. I truly believe that using ZKPs to help with sensitive data in AI models is a true possibility in the future, and it has the ability to completely revolutionize the world of AI.

Acknowledgments

I would like to acknowledge the help I received from:

My parents for always encouraging me in my exploration of Computer Science. They have always supported me and make sure I have everything I need.

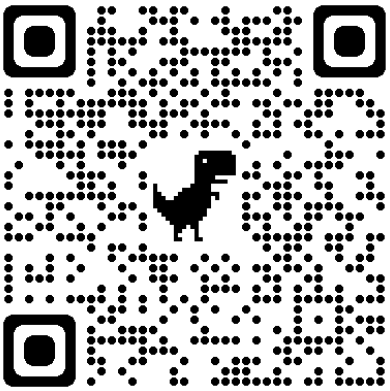
My teacher, Zita Murphy, for organising everything for SciFest and for the help and support she has given me towards my projects.

My school for letting me apply to SciFest@College in TU Dublin. I hope I get to represent the school many more times in the future.

The many YouTube content creators for posting free tutorials on how ZKPs work and the mathematical aspects to them, and tutorials on linear regression models.

Appendices - My Entire Script

Here you will find all my code for my ZKP and AI model. I decided that this section is the best place to include the code as it would become quite tedious and repetitive to comment on every line of code in my early sections. All of the final code and code from early iterations can be viewed on my GitHub Repository, as typing the code here would take up too many pages. The entire project directory and code can be viewed here with the QR code.



References

Abhishek, Allena Vankata Sai. 2023. *Kaggle.* Accessed April 29, 2024. https://www.kaggle.com/datasets/abhishek14398/salary-dataset-simple-linear-regression?rvi=1.

Bi, Qifang, Goodman Katherine E, Kaminsky Joshua, and Lessler Justin. 2019. " What is machine learning? A primer for the epidemiologist." *American journal of epidemiology 188, no. 12* 2222-2239.

Corona, Erika, and Filippo Eros Pani. 2013. ""A review of lean-kanban approaches in the software development."." *WSEAS transactions on information science and applications 10,.* 1-13.

Disch, Wendy, and Rachel Slaymaker. 2023. "Housing affordability: Ireland in a cross-country context." *Economic and Social Research Institute Research Series*, 164.

Fowler, Martin, and Jim Highsmith. 2001. ""The agile manifesto."." *Software development 9* 8: 28-35.

Gurjot. 2024. *GeeksForGeeks.* 16 April. Accessed April 29, 2024. https://www.geeksforgeeks.org/linear-regression-formula/.

Hasan, Jahid. 2019. "Overview and applications of zero knowledge proof (ZKP). ." *International Journal of Computer Science and Network 8, no. 5* 2277-5420.

Loeber, Patrick. 2021. *Python Flask Beginner Tutorial - Todo App - Crash Course.* https://youtu.be/yKHJsLUENl0?si=1QD3M1uN5Mg9-T7v.

Tranmer, Mark, and Mark Elliot. 2008. " "Multiple linear regression."." *The Cathie Marsh Centre for Census and Survey Research (CCSR) 5, no. 5 (2008): 1-5.*

Xing, Zhibo, Zhang Zijian, Liu Jiamou, Zhang Ziang, Li Meng, Zhu Liehuang, and Russello. Giovanni. 2023. "Zero-knowledge proof meets machine learning in verifiability: A survey." *arXiv preprint arXiv:2310.14848.*