A Practical Approach to Donation System Privacy Using Paillier Homomorphic Encryption

Christoper Daniel - 18222034
Program Studi Sistem dan Teknologi Informasi
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: christoper.daniel04@gmail.com , 18222034@std.stei.itb.ac.id

***Abstract*—As digital donation systems become more common, they bring significant risks to the donor data privacy. Sensitive information, such as donation amounts is vulnerable to inappropriate use if it is not properly protected. This paper proposes a practical approach to preserving privacy in donation systems using homomorphic encryption, the Paillier algorithm. The designed system allows donation amounts to be encrypted on the client-side before being sent to the server. The server can perform additive operations directly on the encrypted data (ciphertexts) to count the aggregate total of all donations without knowing the value of individual donations. This implementation uses Python programming language with Flask framework, phe library for Paillier encryption, and SQLite for data storage. The results demonstrate that the system successfully aggregates encrypted donations accurately while maintaining donor confidentiality, offering a great privacy solution for modern donation platforms that balances security with practical usability.**

Keywords—homomorphic encryption; Paillier; data privacy; donation system; cybersecurity; cryptography;

I. Introduction

A. Background

Philanthropy has been changed by the growing popularity of the internet, which makes it easier for people to make use of online platforms to support something they care about than ever before. However, there are some serious issues with data security and privacy that come with this convenience [1]. Every transaction creates a digital trace especially when it comes to donations, this includes private financial and personal information. Public concern has been raised by significant data breaches that have demonstrated that even reputable companies are vulnerable to attacks from either outside or inside. Depending on the type of cause, donors might experience wealth profiling, unwanted targeted strikes, as well as social criticism as the consequence of their donation history being made public.

Therefore, protecting donor's information confidentiality isn't just a trivial requirement but also an important part of developing and maintaining trust which is the foundation of philanthropy [1]. Donors are more likely to make significant and consistent contributions when they feel comfortable with the platform. A system that enables organizations to process donations and count the total amount of money raised without ever having access to the unencrypted values of individual contributions is needed to solve this difficulty. It turns out that homomorphic encryption is the best option. It is a revolutionary type of encryption that enables direct calculation on ciphertexts or encrypted data [2]. This means the server can sum up total donations without having to decrypt each incoming donation, preserving individual privacy from end to end and protecting data even from internal threats, such as a compromised or curious database administrator.

B. Problem Statement

Old contribution systems commonly use basic database encryption or keep the donation amounts in plaintext. These techniques make the system have no defense against internal threats but they do protect data from outside attackers who could steal the actual database files. The plaintext details of each donation are accessible to any program, system administrator, even privileged user with application layer access.

There are significant disadvantages to other privacy-preserving techniques. For example, reduction or data masking permanently changes the data, making it impossible to get accurate aggregation. Despite being an excellent method to confirm the integrity of data, hashing is a one-way function that can't be used for mathematical operations. For example, two hashed values cannot be summed to get a hash of their total [3]. Therefore, the main challenge is to create a system that accurately calculates the total amount of donations from several donors without ever disclosing to any party, including the server that controls the data aggregation. This paper addresses that specific challenge.

C. Objective

This paper's main goal is to develop, implement, and evaluate a privacy-preserving donation system that makes use of the Paillier homomorphic encryption scheme's additive operation. This system will offer a proof that shows how easily accessible technology may be used to safely aggregate sensitive financial data without endangering the security of individual records.

II. Literature Review

A. Donation System Models

Digital donation can be categorized into two models, third-party crowdfunding sites such as GoFundMe or KitaBisa and direct-to-charity portals. The donor usually uses a web form to submit their payment and personal information in both models. Transport Layer Security (TLS) is used to protect this data while it is in motion but it is decrypted and processed once it gets to the server. Although databases can be encrypted while they are not in use, but the application layer itself becomes vulnerable. Individual donation records can frequently be viewed and queried by platform administrators, customer support representatives, or data analysts [1]. A single point of failure for privacy is created by this centralized trust approach.

B. Privacy in Cryptography

Data encryption is the foundation of modern information security. Cryptographic schemes are categorized as symmetric encryption and asymmetric encryption, each with different properties and use cases:

a. Symmetric Encryption

Uses a single shared secret key for both encryption and decryption. It is computationally very fast, making it ideal for encrypting large volumes of data. However, the requirement of a shared secret key makes key distribution difficult in public systems.

b. Asymmetric Encryption

Uses a pair of keys, specifically a public key for encryption and a private key for decryption. The public key can be openly shared without any further and complicated security as it cannot be used for decryption. This model is perfectly suited for donation systems as the public key can be used to encrypt donor contributions while only the organization holding the secret private key can decrypt the final aggregate total [3].

The difference between symmetric and asymmetric encryption can be seen in the following table below

TABLE I. Symmetric vs Asymmetric

Aspect	Symmetric Encryption	Asymmetric Encryption
Keys	Single shared key	Public key and private key
Speed	Fast	Slow
Key Management	Difficult (secure distribution)	Easy (public key can be shared)
Use	Bulk data encryption	Digital signatures, HE

C. Homomorphic Encryption Overview

Homomorphic encryption (HE) allows computations to be performed directly on the ciphertexts without having to decrypt it first. The result is the same as the result of performing the same operations on the plaintexts [2]. This concept was first proposed by Rivest, Adleman, and Dertouzos in 1978, not long after the invention of RSA.

There are two types of homomorphic encryption which are categorized by the operations they support:

a. Partially Homomorphic Encryption (PHE)

Supports operations such as addition or multiplication. The Paillier cryptosystem is a well-known PHE scheme that is additively homomorphic [4].

b. Fully Homomorphic Encryption (FHE)

Supporting both addition and multiplication operations. While incredibly powerful, FHE are currently too slow for most practical applications.

The Paillier cryptosystem was introduced by Pascal Paillier in 1999. Paillier is a probabilistic asymmetric algorithm with a powerful additive homomorphic property [4]. This property is mathematically expressed as:

$$D(E(m_1) \cdot E(m_2) \,(mod\, n^2)) \;=\; m_1 + m_2 \,(mod\, n)$$

This means that the decryption result of two ciphertexts multiplied by each other matches to the sum of their corresponding plaintexts. This feature makes it an excellent choice for applications requiring the summation of private data, such as the donation system proposed in this paper.

The additive homomorphic property of Paillier cryptosystem has made it a popular choice for privacy-preserving applications. In e-voting systems, encrypted ballots can be publicly counted to determine the winner of an election without revealing each individual choice. This is conceptually identical to counting donations. For example, the work by Adida and Rivest, explores the use of homomorphic encryption in voting protocols to ensure both voter privacy and vote integrity [5]. However, many of these systems remain theoretical because they are too complex for implementation. This paper is set by focusing on creating a simple, practical, and accessible web-based prototype specifically for donation system privacy, showing a clear real use case.

III. Design and Implementation

A. System Architecture

The system is designed with a client-server architecture based on the roles of donor (people who donates), server (aggregator), and admin. The architecture is designed to minimize trust placed in the server. The interaction of the system proceeds through this process:

1. Key Setup

In the first execution, the system automatically generates a single Paillier public and private key pair. This is a one time setup process. The private key is stored securely on the server and kept secret while the public key is loaded by the system for encrypting new donations.

2. Donation

   A donor navigates to the web interface, enters their name and donation amount then submits the form.

3. Encryption

   Upon receiving the POST request, the Flask server immediately encrypts the donation amount using the public key. After that, the original plaintext amount is discarded from memory.

4. Storage

   The server stores the donor's name as plaintext alongside with the encrypted donation amount as ciphertext in an SQLite database.

5. Aggregation

   The server fetches all ciphertexts from the database and combines them using homomorphic encryption by multiplying them together to get the encrypted total amount of the donation.

6. Decryption (for demonstration purposes)

   The resulting aggregated ciphertext is passed to a secure admin function. This function uses the private key to decrypt the total donation and show the final sum without exposing any individual donation.

The process can be seen visually through the diagram below

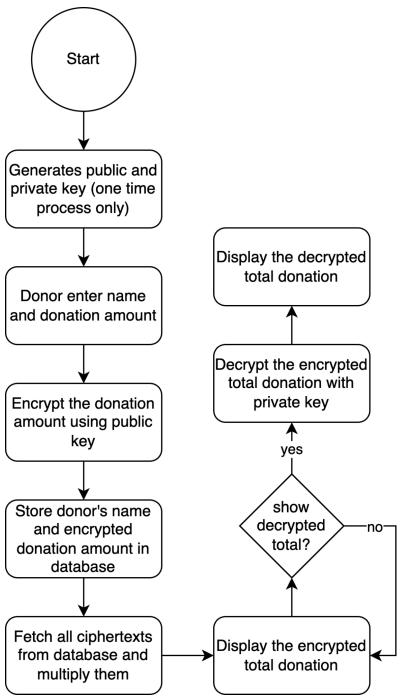

Fig. 1. System process flowchart

B. Paillier Encryption Scheme

The system is built with the mathematical foundation of the Paillier cryptosystem. The core operations are implemented in the phe library by python.

1. Key Generation
   - Choose two large prime numbers, p and q, where $gcd(pq, (p-1)(q-1)) = 1$
   - Find modulus $n = pq$ and Charmichael's function $\lambda = lcm(p-1, q-1)$
   - Find g where $g < n^2$
   - The public key is the pair (n, g)
   - The private key is the pair (λ, μ), where $\mu = (L(g^{\lambda}(mod\ n^2)))^{-1}(mod\ n)$ and $L(x) = (x-1)/n$

2. Encryption
   - To encrypt plaintext m where $0 \leq m < n$, select random integer r where $0 < r < n$ and $gcd(r, n) = 1$
   - The ciphertext is computed as $c = g^m \cdot r^n (mod\ n^2)$
   - The random r ensures that the scheme is probabilistic, encrypting the same plaintext multiple times will produce a different ciphertext each time so it prevents attackers from linking identical donations.

3. Decryption
   - To decrypt ciphertext c, use private key pair (λ, μ) where the ciphertext is recovered as $m = L(c^{\lambda}(mod\ n^2)) \cdot \mu\ (mod\ n)$

4. Homomorphic Addition
   - Given two ciphertext $c_1 = E(m_1)$ and $c_2 = E(m_2)$. When they're multiplied by each other and modulo by n^2, the result is $c_{sum} = c_1 \cdot c_2\ (mod\ n^2) = (g^{m_1} r_1^n) \cdot (g^{m_2} r_2^n)(mod\ n^2) = g^{m_1+m_2}(r_1 r_2)^n\ (mod\ n^2)$
   - The resulting c_{sum} is a true encryption of $m_1 + m_2$

C. Data Flow and Storage

The data flow is maintained by Flask in app.py. When someone submits a donation, the "/" route handler for POST requests will be triggered. The details of the implementation is shown by the following code below

```
@app.route('/', methods=['GET', 'POST'])
def index():
```

```
    if request.method == 'POST':
        try:
            name = request.form['name']
            amount =
float(request.form['amount'])
            enc = public_key.encrypt(amount)
            ciphertext_str =
f"{enc.ciphertext()}|{enc.exponent}"

            db = get_db()
            db.execute('INSERT INTO donations
(donor_name, ciphertext) VALUES (?, ?)',
(name, ciphertext_str))
            db.commit()
            flash(f"Donation by {name}
recorded successfully!")
        except Exception as e:
            flash(f'Error: {e}')
        return redirect(url_for('index'))
    return render_template('index.html')
```

The resulting ciphertext from phe library contains large integer results and a public key exponent. It needs to be separated by a pipe "|" to store it in a simple text field in SQLite database. The schema for the database is shown below

```
CREATE TABLE donations (
            id INTEGER PRIMARY KEY
AUTOINCREMENT,
            donor_name TEXT NOT NULL,
            ciphertext TEXT NOT NULL
        )
```

The implementation tech stack that is used in this system is quite simple. Python's micro web framework, Flask, is used in the backend for routing and handling web requests. The python-paillier (phe) library is used to implement the homomorphic cryptosystem which provides a high-level and user-friendly API. For the database, the system uses SQLite, a self-contained and serverless SQL database engine which is ideal for prototyping. Lastly, the front-end is built with standard HTML5, CSS3, and JavaScript to create a responsive and interactive user interface. Here's the project implementation directory structure:

```
/DonationSystemPrivacyPaillier
```

```
|-- app.py
|-- donations.db
|-- public_key.pickle
|-- private_key.pickle
|-- README.md
|
|-- interface/
|     |-- index.html
|     `-- admin.html
|
`-- static/
      |-- css/
      |     |-- main.css
      |     `-- admin.css
      `-- js/
            |-- main.js
            `-- admin.js
```

App.py is the main entry point of the application where it contains all the Flask backend logic, such as routing, request handling, and Paillier encryption decryption process. Interface is the directory containing HTML templates that are rendered by Flask. Static directory contains all the static files that are served directly to the browser, such as JS and CSS. As for donations.db, public_key.pickle, and private_key.pickle are automatically generated on the first run.

D. Security Features

To protect donor privacy, the system's architecture includes several kinds of security measures. The threat model assumes an external attacker and a server administrator who is honest but also curious who will follow protocols but may try to extract information from the data that they can access. Here is some of the security features from the system:

a. Confidentiality of the donation amount
   Individual donation amount are never shown or processed in plaintext on the server. They are encrypted immediately and only the final sum is ever decrypted (for demonstration purposes). This protects from both database breaches and insider threats.

b. Semantic security
   Due to the characteristics of Paillier encryption, where the encryption is based on probability, an attacker can't determine if two donors donated the same amount or not as the ciphertexts will be different from one another.

IV. Results

A. System Demonstration

The implemented donation privacy system provides two web interfaces: a public-facing donation page and a restricted admin dashboard. Note that in this implementation program, there is no user registration or role-based access control because this implementation is for demonstration purposes.

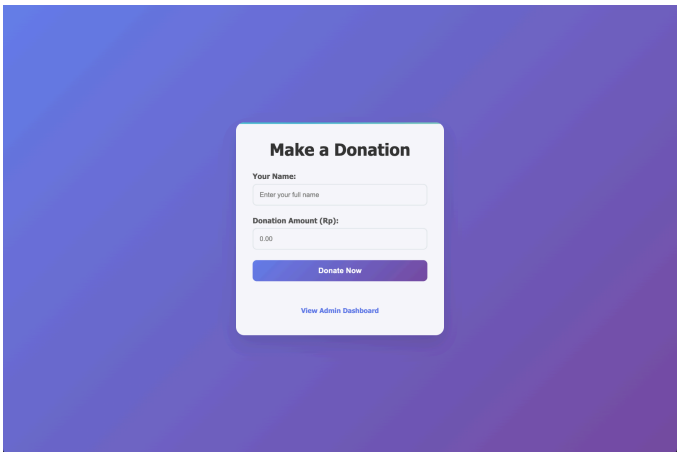

Fig. 2. The donation interface for submitting donation amount

The public-facing donation page has a clean and simple form that allows users to enter their name and a donation amount they desire. The interface provides immediate feedback upon successful submission.

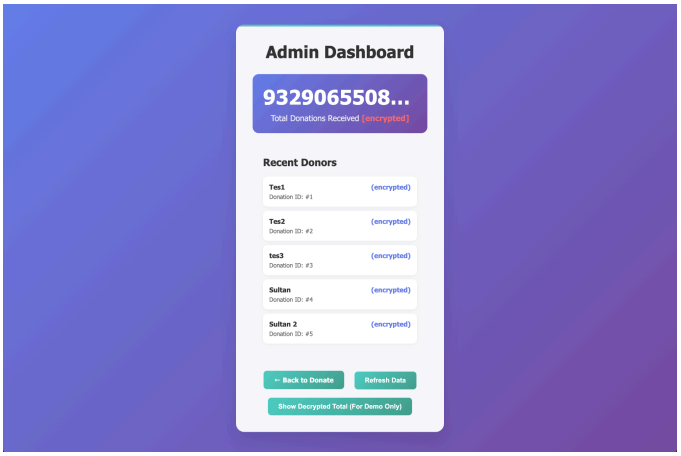

Fig. 3. Admin dashboard page showing encrypted total and donors list

This admin dashboard displays a list of all donors with their donation encrypted and the homomorphically aggregated total donation which is shown as a large and encrypted integer. Button "Show Decrypted Total," allows the administrator to trigger the decryption of the final sum but this is for demonstration purposes only.

B. Encryption Performance and Scalability

The system's performance was evaluated qualitatively. The time needed for key creation, encryption, and aggregation is low for small to moderate donations, resulting in a pleasant user experience. However, it is well known that homomorphic encryption requires greater computing capacity to operate than conventional cryptography.

Aggregation and decryption times would rise for a larger scale system that handles millions of donations or utilizing much bigger key sizes (e.g. 4096-bit or 2048-bit). Aggregation requires getting every ciphertext and multiplying each one, which is an O(N) operation. For production systems, the trade-off between scalability (number of donors) and security (key size) is important.

C. Security Validation and Correctness

The correctness of the homomorphic aggregation was the most important aspect to validate. This was done by comparing the decrypted total with the manual sum of all plaintext donations that was made during the testing. Here is the example that illustrates the process:

1. Alice donates Rp100.000 and the system computes $c_1 = E(100.000)$
2. Bob donates Rp150.000 and the system computes $c_2 = E(150.000)$
3. The system counts the aggregate ciphertext $c_{total} = c_1 \cdot c_2 \, (mod\ n^2)$
4. The admin requests the total of donation, then the system sends c_{total} to the decryption function.
5. The decryption function then counts $D(c_{total}) = D(E(100.000) \cdot E(150.000))$ which has the same result as $100.000 + 150.000$.

This process was tested repeatedly with various amounts and the decrypted total accurately matched the sum of the individual plaintext amounts. This confirms that the additive homomorphic encryption of the Paillier scheme was implemented correctly and reliable for this use case.

D. Limitations

While this prototype system successfully demonstrates the core concept, it has several limitations that would need to be addressed before going into a production environment:

a. Partially Homomorphic Nature

The system only supports addition operation. It can't perform complex analytics, such as calculating the donation average or performing standard deviation calculations which would require a fully homomorphic scheme.

b. Key Management

The public keys and private keys are stored in local pickle files (public_key.pickle, private_key.pickle). This is insecure and not scalable because in a real-world scenario, the private key must be stored in a secure database or a dedicated key management service (KMS) to prevent robbery and managing access needs.

c. Scalability Concerns

The computational burden of homomorphic operations might become a breakdown for systems with a very high volume of transactions or when using larger and more secure key sizes.

d. Lack of Public Verifiability

There is no way for the public to confirm that the decrypted total in the admin dashboard is accurate.

They have to trust that the administrator hasn't changed the aggregate process or the result. More complex cryptographic methods could be used to overcome this restriction.

V. Conclusion

The system implementation has successfully demonstrated a practical and privacy-preserving donation system using the Paillier homomorphic encryption. It is feasible to build a functional and secure system where financial data like donation value can be aggregated without exposing the individual values. This effectively prevents the risk of sensitive donor information. The implementation validates that properties of homomorphic addition can be practically integrated into a web application architecture using tools that are necessary to use.

The practical approach that is presented in this paper has significant implications for any real-world application where data confidentiality is very important nowadays. It provides a concrete cryptographic answer to privacy issues that may prevent people from participating in online activities, especially in the social and financial area. This encryption algorithm is appropriate for private medical data analysis, secure e-voting, and any other type of confidential financial reporting alongside the donation privacy system. In our growing data-driven society, this technology can improve security and trust by enabling computation on encrypted data.

Source Code Link at Github

Here's the GitHub repository link where you can access the code:

https://github.com/ChristoperDaniel/DonationSystemPrivacyPaillier

Acknowledgement

I would sincerely express my praise and gratitude to the Almighty God that this paper can be completed within the given deadline. I also would like to express my deepest gratitude to Mr. Dr. Ir. Rinaldi Munir, M.T. for all the lectures that have been given and for this opportunity to write this paper. I am also grateful for my family for always supporting me till the very end and for all the rightful owner of all the references that I used in order to complete this paper.

References

[1] A. Acquisti, L. Brandimarte, and G. Loewenstein, "Privacy and human behavior in the age of the Internet," Science, vol. 347, no. 6221, pp. 509-514, 2015.

[2] R. Munir, "Enkripsi Homomorfik," Lecture Slides for II4021 Kriptografi, Institut Teknologi Bandung, 2025.

[3] W. Stallings, Cryptography and Network Security: Principles and Practice, 7th ed. Pearson, 2017.

[4] P. Paillier, "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes," in Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security (EUROCRYPT '99), 1999, pp. 223–238.

[5] B. Adida and R. L. Rivest, "Scratch & Vote: A Verifiable Paper-Based Cryptographic Voting System," in Proceedings of the 5th ACM Workshop on Privacy in the Electronic Society, 2006, pp. 69-80.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 20 Juni 2025

Christoper Daniel / 18222034