Secure Email Systems: Phishing Detection with Cryptographic Verification

Mummadi Hemanth Reddy
Department of Computer Science & Engineering
Amrita School of Computing, Bengaluru
Amrita Vishwa Vidyapeetham, India
bl.en.u4aie22038@bl.students.amrita.edu

Okesh Ankireddypalli
Department of Computer Science & Engineering
Amrita School of Computing, Bengaluru
Amrita Vishwa Vidyapeetham, India
bl.en.u4aie22044@bl.students.amrita.edu

Mouhitha Arella
Department of Computer Science & Engineering
Amrita School of Computing, Bengaluru
Amrita Vishwa Vidyapeetham, India
bl.en.u4aie22075@bl.students.amrita.edu

Saranya Gujjula
Department of Computer Science & Engineering
Amrita School of Computing, Bengaluru
Amrita Vishwa Vidyapeetham, India
bl.en.u4aie22077@bl.students.amrita.edu

Niharika Panda
Department of Computer Science & Engineering
Amrita School of Computing, Bengaluru
Amrita Vishwa Vidyapeetham, India
p_niharika@blr.amrita.edu

***Abstract*—This study presents a system integrating phishing detection with secure email transmission, addressing the critical need for robust cybersecurity in email communication. The system employs a stacked machine learning model, combining Logistic Regression, Random Forest, and Gradient Boosting with Random Forest as the meta-classifier, to detect phishing emails using text-based and stylometric features. Secure transmission is achieved through Advanced Encryption Standard(AES), Rivest, Shamir, and Adleman(RSA), and ChaCha20 encryption algorithms, tailored for email content and attachments. Implemented via a Flask frontend with a SQLite database, the system supports user-based message search and secure key management. Evaluation on balanced and unbalanced datasets demonstrates high phishing detection accuracy (up to 98.37%) and reliable encryption/decryption, with no data leakage. The system's usability and security make it a promising solution for protecting sensitive email communications.**

***Index Terms*—Phishing Detection, Secure Email Transmission, Machine Learning, Encryption, Homomorphic Encryption, Flask, SQLite**

I. Introduction

Email remains a critical communication tool in both personal and professional domains, yet its widespread use makes it a prime target for cyber threats, particularly phishing attacks. Phishing emails, designed to deceive users into revealing sensitive information or executing malicious actions, account for a significant portion of cybersecurity breaches, with studies reporting that over 90% of data breaches originate from phishing attempts [1]. The increasing sophistication of these attacks, coupled with the reliance on email for sensitive data exchange, underscores the urgent need for robust detection mechanisms and secure transmission protocols to safeguard user data and maintain trust in digital communication systems.

Traditional phishing detection systems often rely on machine learning techniques, taking advantage of features such as text materials, stylometric patterns and metadata to identify malicious emails. However, these systems typically work on unnovated data, presenting privacy risks when processing sensitive communication. Concurrently, safe email transmission protocols, such as AES, RSA, or chacha20 encryption, protect the data during transit, but lack integrated fishing detections, weakening users for malicious content on decryption. The absence of an integrated system that combines real -time fishing detection with safe transmission, limiting the effectiveness of current solutions, requires an innovative approach that addresses both safety and privacy.

This study proposes a comprehensive system that integrates fishing detections with a safe email transmission, which is applied through a flask-based friend for user interaction. The system appoints a stacked machine learning model, which increases logistic region, random forest, and gradients as a meta-classifier with random forest, to detect fishing email using text-based and stylomatic features. For safe transmission, it provides AES, RSA and chacha20 encryption, which corresponds to text, high-protection messages and attachments, respectively. Homomorphic encryption enables the detection of fishing on encrypted data, ensures privacy, while a SQLite database supports user-based message search and key management. This paper evaluates the performance of the system in the phishing detection accuracy, cryptographic safety, and the performance, demonstrating its ability as a strong solution for safe email communication.

II. Literature Survey

Phishy email classification has been studied in depth with both conventional machine learning and current deep learning methods. Alhuzali et al. [2] performed an exhaustive comparison of 14 DL and ML models on ten datasets and concluded that transformer-based models like BERT and RoBERTa were better, with accuracies of 98.99% and 99.08%, respectively. Their findings proved that deep learning models possess a strong ability in detecting phishing patterns through capturing semantic subtleties in email messages. Analogously, Sahit et al. [3] benchmarked conventional models such as Random Forest, SVM, and Logistic Regression, on a rich feature set including link-based, content-based, and behavioral features. Of these, Random Forest was the most accurate at 98.46% and was observed to work well in real-time phishing detection scenarios.

Stepping away from email alone, Jayaprakash et al. [4] introduced a heuristic machine learning system that could identify phishing attacks on email, URL, and website platforms. Their method was based on preprocessed, feature-engineered data with a weighted scoring approach, and recorded 97.4%, 97.2%, and 98.1% accuracies for email, URL, and website detection respectively. This demonstrates the ability of heuristic-based learning in multi-platform threat detection. Uddin and Sarker [5] proposed a fine-tuned DistilBERT model incorporating Explainable AI (LIME, Transformer Interpret) to enhance detection accuracy (98.48%) and interpretability to better comprehend phishing content through reasoning at the token level. Koide et al. [6] also proposed ChatSpamDetector based on GPT-4 with 99.70% detection accuracy in phishing emails by rewriting emails as structured prompts. The system not only exhibited superior performance but also included extensive reasoning for every classification, making it more transparent and trustworthy for AI-driven security tools. With respect to encryption, there has been wide-ranging research that has analyzed both traditional and contemporary cryptographic schemes to implement secure and effective data protection for a variety of application scenarios. Muhammed et al. [7] compared symmetric encryption algorithms—AES, Blowfish, Twofish, Salsa20, and ChaCha20—by Java-based implementations on their encryption/decryption rates and throughput. Among them, ChaCha20 was the most efficient one with more than 50% better performance in both operations, and it is quite suitable for high-speed, low-latency applications, whereas Twofish was slow in throughput. In response to the increasing complexity of side-channel attacks, Singh et al. [8] suggested a hybrid cryptographic system combining AES, ChaCha20, and RSA, improved via optimization methods like Wasserstein GAN with Gradient Penalty (WGAN-GP) and Genetic Algorithms. The hybrid system not only provided increased resistance against side-channel attacks but also the provision of dynamic key rotation, which cut down significantly on static key reuse-related vulnerabilities. Supporting such initiatives in the IoT space, Karmous et al. [9] compared AES-256, RSA, and ChaCha20 for protecting MQTT-based communications in Software-Defined Networking (SDN) scenarios. Their observations pointed towards AES-256 as the strongest in encryption and computationally efficient, surpassing the performance of RSA and ChaCha20 in both speed and energy consumption, thereby emphasizing the necessity of lightweight cryptographic protocols for performance-limited embedded systems.

Additionally, integration of key exchange mechanisms with efficient encryption was investigated by Muhammed et al. [10], who suggested a hybrid cloud security model utilizing ChaCha20 for encrypting the data and ECDH for secure key exchange. This combination facilitated strong confidentiality and effective performance in cloud-based implementations. A novel cryptographic key generation technique was also introduced in [11] based on a hybrid PUF implementation (Ring Oscillator + Arbiter) XOR-integrated for randomness enhancement. Verified through Vivado simulations and AES encryption, the technique efficiently mitigated key storage requirements, providing strong protection against physical and side-channel attacks. Keerthan et al. [12] have carried out a comparative analysis of both symmetric (AES, DES, Blowfish) and asymmetric (RSA) algorithms on parameters of time, key size, and strength. They stressed that the choice of algorithm should be based on the context and application for maximizing confidentiality and data integrity.

Hiransha et al. [13] suggested CEN-Deepspam, which uses deep learning in the form of Keras word embeddings and Convolutional Neural Networks (CNNs) to learn semantic patterns from email text. Their scheme, with training datasets with and without email headers, demonstrated strong classification performance by properly extracting n-gram-like features, which indicated the potential of deep learning towards more effective phishing detection. On the other hand, Harikrishnan et al. [14] employed standard machine learning methods, applying Term Frequency-Inverse Document Frequency (TF-IDF) for feature extraction and classifying with classifiers such as Random Forest, AdaBoost, Naive Bayes, Decision Trees, and Support Vector Machines (SVMs). Their comparative analysis found ensemble methods like Random Forest and AdaBoost provided higher accuracy and robustness, while SVMs gave higher precision in some cases, confirming the applicability of traditional supervised models along with good feature representation. Vazhayil et al. [15] augmented traditional methods with PED-ML which applied dimensionality reduction techniques like Singular Value Decomposition (SVD) and Non-negative Matrix Factorization (NMF) on term-document matrices. This preprocessing removed noise and enhanced feature discriminative capability, resulting in improved classification performance and highlighting the importance of data representation optimization in phishing detection scenarios. Scaling up, Athulya et al. [16] reviewed the phishing attack methods such as email obfuscation and domain spoofing and proposed a hybrid detection system that combines rule-based systems, machine learning classifiers, and heuristics. Their real-time hybrid model suggested boosting detection rates and reducing false positives, emphasizing flexibility and the effectiveness of combining diverse methods to counter evolving phishing tactics. Lastly, Unnithan et al. [17] combined machine learning with domain-specific lexical features like suspicious URLs and domain anomalies. Their model showed that the incorporation of contextual and linguistic information significantly enhanced precision and recall, proving the significance of domain knowledge and interpretable features to deployable phishing detection systems.

III. Methodology

This research provides a phishing-resilient email transmission system, developed through integration of phishing detection, using a Flask-based frontend interface for user interaction. The research methodology includes dataset preparation, preprocessing, feature extraction, phishing detection through machine learning, and a cryptographic scheme utilizing AES, RSA, and ChaCha20 to protect email content and attachments from eavesdropping. Homomorphic encryption is used to carry out phishing detection on encrypted data without interfering with data integrity. The system illustrates the data flow from email input to phishing detection and safe transmission.

Fig 1, illustrates the architecture of the entire system, demonstrating how emails are processed from input to their eventual transmission. It starts with ingestion of emails followed by detection of phishing using trained models. In case the email is deemed secure, it goes to the encryption stage using algorithms such as AES, RSA, or ChaCha20. The encrypted message is transmitted using a web interface that has been built with Flask. The diagram represents how different modules like preprocessing, phishing, and encryption collaborate to make communication secure.

A. Dataset Preparation

The phishing detection system was trained and evaluated using two email datasets:

- **Phishing Emails**: A collection of phishing emails sourced from a public repository, stored in .mbox format.
- **Legitimate Emails**: The Enron email dataset, a widely-used corpus of legitimate corporate emails, also in .mbox format.

These datasets were converted into pandas DataFrames for processing, with two configurations created: a balanced dataset (1:1 phishing-to-legitimate ratio) and an unbalanced dataset (1:10 ratio) to assess model performance under varying class distributions.

B. Data Preprocessing

The preprocessing pipeline transformed raw email data into a suitable format for feature extraction and classification, as detailed below:

- **Text Extraction**: Emails were parsed to extract text content, handling both plain text and HTML formats. HTML emails were converted to plain text using BeautifulSoup, with inline tags unwrapped to ensure text continuity. Rudimentary deduplication removed redundant text from multipart emails.
- **Cleaning**: Empty rows were removed to eliminate invalid entries. The balanced and unbalanced datasets were constructed to reflect real-world and controlled scenarios.
- **Tokenization and Lemmatization**: The NLTK library was used to tokenize email text, removing punctuation and special characters via regex filtering. Stopwords were excluded, and tokens were lemmatized using WordNetLemmatizer with part-of-speech tagging to normalize word forms.
- **Address Sanitization**: Email addresses and URLs were replaced with placeholders (`<emailaddress>` and `<urladdress>`) using regex patterns to prevent their influence on feature extraction.

C. Feature Extraction

Two feature types were extracted to support phishing detection:

1) **Text-Based Features**:
   - **TF-IDF**: Tokenized text was vectorized using scikit-learn's TfidfVectorizer, with parameters `max_df`, `min_df`, and `max_features` tuned to filter high- and low-frequency terms. Chi-squared feature selection (SelectPercentile, 50th percentile) identified the most discriminative features.
   - **Word2Vec**: Semantic embeddings were generated using gensim's Word2Vec with a skip-gram model (`vector_size=100`, `min_count=5`). Mean vectors were computed for each email to capture semantic content.
2) **Stylometric Features**: These captured writing style characteristics, including:
   - Counts of characters, words, sentences, and special characters.
   - Ratios of alphabetic tokens, special characters, and characters per line.
   - Word size distributions (small: $\leq$4 characters, big: $>$8 characters, huge: $>$14 characters).
   - Sentence length statistics (mean, standard deviation, minimum, maximum).
   - Readability scores (e.g., Flesch-Kincaid, Gunning Fog) computed using py-readability-metrics.
   - Spelling and grammatical error counts via language_tool_python.

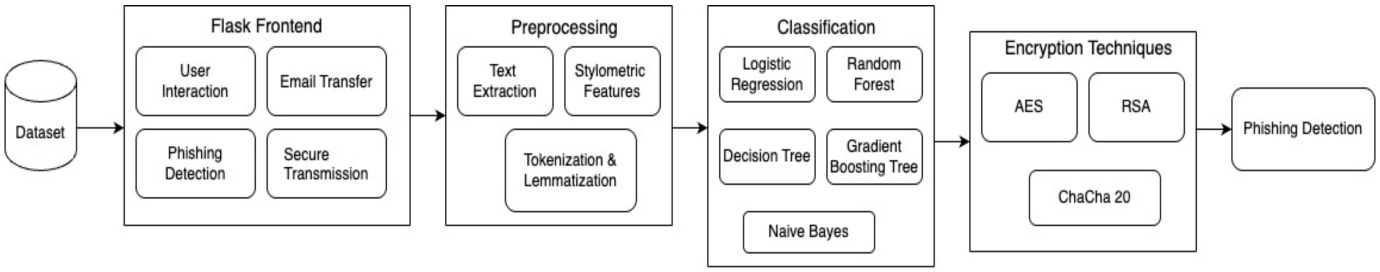


Fig. 1. System Architecture

Missing values in stylometric features were imputed using scikit-learn's `SimpleImputer` with a mean strategy. The feature extraction process included both text-based and stylometric feature extraction workflows.

D. Phishing Detection

Phishing detection was performed using five machine learning algorithms: Logistic Regression (LR), Decision Tree (DT), Random Forest (RF), Gradient Boosting Tree (GBT), and Naive Bayes (NB). The classification process included:

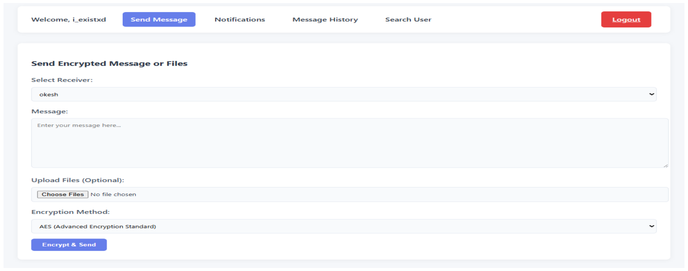
- **Baseline Models**: Models were trained separately on text-based (TF-IDF, Word2Vec) and stylometric features to establish performance benchmarks.
- **Combined Models**: Text-based and stylometric features were merged, and model stacking was implemented using RF and GBT as meta-classifiers to integrate predictions from LR, RF, and GBT.

Datasets were split into 80% training and 20% testing sets using stratified sampling to maintain class proportions. Performance was evaluated using accuracy, precision, recall, F1 score, false positive rate (FPR), false negative rate (FNR), and area under the ROC curve (AUC). To ensure data security, phishing detection was conducted on encrypted emails using homomorphic encryption, allowing computation on ciphertexts without decryption.

E. Secure Email Transmission

The system provides secure email transmission through a cryptographic framework integrated with a Flask-based frontend, enabling user-friendly interaction as shown in Figure 3. The cryptographic components are detailed below:

1) **Frontend Implementation**: The Flask application manages user authentication, message composition, encryption selection, and phishing detection results display. The database, implemented with SQLite, stores user credentials, public/private keys, encrypted messages, and temporary encryption keys. Tables include `users`, `messages`, `message_files`, `decrypted_messages`, and `decrypted_files`, ensuring structured data management.
2) **Encryption Algorithms**:
   - **AES**: Utilized for rapid encryption of text-based email content using the Fernet scheme from the `cryptography` library. A symmetric key is generated for each message, stored temporarily in the database, and used for both encryption and decryption. AES is ideal for large text datasets due to its efficiency.
   - **RSA**: Employed for high-security scenarios, using 2048-bit keys generated with the `pycryptodome` library. Messages and files are encrypted in chunks (190 bytes) with PKCS1_OAEP padding, leveraging the recipient's public key. RSA's asymmetric nature ensures secure key exchange but is slower, suitable for critical communications.
   - **ChaCha20**: Applied for encrypting attachments, particularly images, using a 32-byte key and 12-byte nonce generated with `pycryptodome`. Its streaming capability ensures efficient handling of large files. Encrypted data includes the nonce prepended to the ciphertext.
3) **Homomorphic Encryption**: A simplified additive homomorphic encryption scheme, based on the Paillier cryptosystem, was implemented to enable phishing detection on encrypted data. The `SimpleHomomorphic` class generates large primes p and q, computing $n = p \cdot q$ and n^2. Encryption uses $g = n+1$ and a random r, producing ciphertext $c = g^m \cdot r^n \mod n^2$, where m is the plaintext. Decryption recovers m using the modular inverse of the totient function. This allows feature computations (e.g., TF-IDF scores) on ciphertexts, with results decrypted for classification, ensuring data privacy during phishing detection.
4) **Key Management**: Encryption keys are dynamically generated and stored in the SQLite database, accessible only during encryption and decryption. Keys are not exposed to users and are deleted post-decryption to prevent data leakage. For RSA, public/private key pairs are stored in the `users` table.
5) **Transmission Process**: Senders select the encryption method via the Flask interface, encrypt the email body and attachments, and transmit them. Recipients receive notifications displaying the phishing probability (computed using the combined model) before decryption. Decryption occurs only after phishing assessment, minimizing exposure to malicious content.



Send Encrypted Message or Files

Select Receiver: okesh

Message: Enter your message here...

Upload Files (Optional): No file chosen

Encryption Method: AES (Advanced Encryption Standard)

Fig. 2. Flask application interface for secure email transmission, showcasing user-friendly interaction with cryptographic functionalities.

Fig 2, shows the graphical interface developed using Flask. It allows users to enter email content, view phishing detection results, and send encrypted emails. The interface is designed to be intuitive, offering options for selecting encryption methods and reviewing outputs securely. This ensures that users can interact with the system easily while the technical operations remain in the background.

F. Evaluation

The system was evaluated on two fronts:

- **Phishing Detection Performance**: Models were assessed on balanced, unbalanced, and combined datasets, comparing accuracy, precision, recall, F1 score, false positive rate, false negative rate, and area under the curve.
- **Security**: The cryptographic system was qualitatively analyzed for encryption strength, key management security, and data leakage prevention.

IV. Results and Discussion

This section evaluates the performance of the proposed system, focusing on phishing detection accuracy, cryptographic security for email and file transmission, and user-based message search functionality. The phishing detection component was tested using balanced (1:1 phishing-to-legitimate ratio) and unbalanced (1:10 ratio) datasets, with the LR, RF, GB ensemble (RF meta-classifier) selected as the optimal model due to its superior metrics. The cryptographic framework, supported by a Flask frontend and SQLite database, ensured secure transmission and efficient data management. Results are discussed in detail, with figures illustrating key processes and tables summarizing detection performance.

A. Phishing Detection Performance

The phishing detection system was evaluated using five algorithms - logistic regression (LR), decision tree (DT), random forest (RF), gradient boost tree (GBT) and naive Bayes (NB) on balanced and unbalanced datasets. Performance metrics included accuracy, precision, recall, F1 score, false positive rate (FPR), false negative rate (FNR), and area under the ROC curve (AUC). Baseline models used text-based (TF-IDF, Word2Vec) and stylometric features, while combined models stacked LR, RF, and GB predictions with RF or GB meta-classifiers.

TABLE I
Performance Metrics for Balanced Dataset (1:1 Phishing-to-Legitimate Ratio

Algorithm	Accuracy	Precision	Recall	F1 Score	FPR	FNR	AUC
Logistic Regression	0.8429	0.8764	0.8683	0.8723	0.1982	0.1317	0.9080
Decision Tree	0.8417	0.8499	0.9035	0.8759	0.2583	0.0965	0.8724
Random Forest	0.8624	0.8581	0.9314	0.8932	0.2492	0.0686	0.9252
Gradient Boosting Tree	0.8911	0.9205	0.9017	0.9110	0.1261	0.0983	0.9547
Naive Bayes	0.6181	0.6187	0.9963	0.7633	0.9940	0.0037	0.8546

As shown in Table I, in the balanced dataset GBT achieved the highest baseline performance (accuracy=0.8911, F1=0.9110, AUC=0.9547, FPR=0.1261), excelling in distinguishing phishing emails. RF was closely followed, with a notable recall of 0.9314, minimizing missed detections. NB performed poorly (accuracy=0.6181, FPR=0.9940), unsuitable for practical use due to excessive false positives. As shown in Table II, the unbalanced dataset , presented challenges, with GBT maintaining robustness (F1=0.7969, AUC=0.9658) but low recall between models (e.g. RF = 0.3043, NB = 0.0000), which highlighted the issues of class imbalance. These results suggest that ensemble methods such as GBT and RF are effective in balanced scenarios, but imbalance requires mitigation strategies.

TABLE II
Performance Metrics for Unbalanced Dataset (1:10 Phishing-to-Legitimate Ratio)

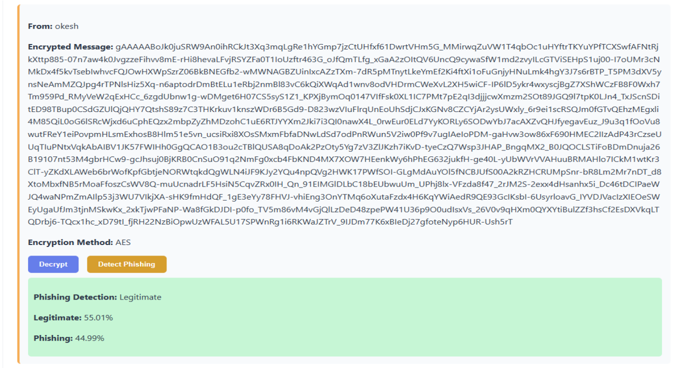
Algorithm	Accuracy	Precision	Recall	F1 Score	FPR	FNR	AUC
Logistic Regression	0.6571	0.9688	0.4601	0.6239	0.0240	0.5399	0.9102
Decision Tree	0.7156	0.9678	0.5584	0.7082	0.0300	0.4416	0.9076
Random Forest	0.5700	1.0000	0.3043	0.4666	0.0000	0.6957	0.9457
Gradient Boosting Tree	0.7878	0.9758	0.6735	0.7969	0.0270	0.3265	0.9658
Naive Bayes	0.3819	0.0000	0.0000	0.0000	0.0000	1.0000	0.8551

As shown in Table III, Combined models significantly improved performance, with the LR, RF, GB ensemble (RF meta-classifier) achieving the highest accuracy (0.9837), F1 score (0.9843), and AUC (0.9977) on the balanced dataset. Its low FPR (0.0215) and FNR (0.0115) ensure minimal errors, critical for user trust. The consistency across combinations validates the integration of text-based and stylometric features.

TABLE III
Performance Metrics for Combined Models

Algorithm	Accuracy	Precision	Recall	F1 Score	FPR	FNR	AUC
LR, RF, GB (RF)	0.9837	0.9801	0.9885	0.9843	0.0215	0.0115	0.9977
LR, GB (RF)	0.9837	0.9801	0.9885	0.9843	0.0215	0.0115	0.9976
RF, LR merged, GB	0.9837	0.9828	0.9856	0.9842	0.0184	0.0144	0.9968
LR, GB (GB)	0.9822	0.9800	0.9856	0.9828	0.0215	0.0144	0.9983
GB (GB)	0.9822	0.9800	0.9856	0.9828	0.0215	0.0144	0.9979
RF, GB (GB)	0.9807	0.9772	0.9856	0.9814	0.0245	0.0144	0.9975
LR, GB merged, RF	0.9807	0.9772	0.9856	0.9814	0.0245	0.0144	0.9952
All, GB	0.9807	0.9772	0.9856	0.9814	0.0245	0.0144	0.9984
LR, RF, GB (GB)	0.9807	0.9772	0.9856	0.9814	0.0245	0.0144	0.9981
All, LR	0.9807	0.9772	0.9856	0.9814	0.0245	0.0144	0.9984
LR, RF, GB (LR)	0.9807	0.9772	0.9856	0.9814	0.0245	0.0144	0.9985
LR, GB (LR)	0.9807	0.9772	0.9856	0.9814	0.0245	0.0144	0.9986
LR, GB (RF)	0.9807	0.9772	0.9856	0.9814	0.0245	0.0144	0.9985

Consequently, this model was deployed to compute phishing probabilities, displayed via the Flask frontend before decryption, as shown in Fig 3.



From: okesh

Encrypted Message:
gAAAAABoJk0juSRW9An0ihRCkJt3Xq3mqLgRe1hYGmp7jzCtUHfxf61DwrtVHm5G_MMirwqZuVW1T4qbOc1uHYftrTKYuYPfTCXSwfAFNtRj
kXttp885-07n7aw4k0JvgzzeFihvv8mE-rHi8hevaLFvjRSYZFa0T1IoUzftr463G_oJfQmTLfg_xGaA2zOItQV6UncQ9cywaSfW1md2zvyILcGTViSEHpS1uj00-I7oUMr3cN
MkDx4f5kvTsebIwhvcFQJOwHXWpSzrZ06BkBNEGfb2-wMWNAGBZUinIxcAZzTXm-7dR5pMTnytLkeYmEf2Ki4ftXi1oFuGnjyHNuLmk4hgY3J7s6rBTP_T5PM3dXV5y
nsNeAmMZQJpg4rTPNlsHiz5Xq-n6aptodrDmBtELu1eRbj2nmBl83vC6kQiXWqAd1wnv8odVHDrmCWeXvL2XH5wiCF-IP6lD5ykr4wxyscjBgZ7XShWCzFB8F0Wxh7
Tm959Pd_RMyVeW2qExHCc_6zgdUbnw1g-wDMget6H07CS5syS1Z1_KPXjBymOq0147VIfFsk0XL1IC7PMt7pE2qI3djjjcwXmzm2SOt89JGQ9l7tpK0LJn4_TxJScnSDi
tED98TBup0CSdGZUlQjQHY7QtshS89z7C3THKrkuv1knszWDr6B5Gd9-D823wzVIuFlrqUnEoUhSdjCJxKGNv8CZCYjAr2ysUWxly_6r9ei1scRSQJm0fGTvQEhzMEgxli
4M85QiL0oG6lSRcWjxd6uCphEQzx2mbpZyZhMDzohC1uE6RTJYYXm2Jki7i3QI0nawX4L_0rwEur0ELd7YyKORLy6SODwYbJ7acAXZvQHJfyegavEuz_J9u3q1fOoVu8
wutFReY1eiPovpmHLsmExhosB8Hlm51e5vn_ucsiRxi8XOsSMxmFbfaDNwLdSd7odPnRWun5V2iw0Pf9v7ugIAeIoPDM-gaHvw3ow86xF690HMEC2IIzAdP43rCzseU
UqTIuPNtxVqkAbAIBV1JK57FWIHh0GgQCAO1B3ou2cTBlQUSA8qDoAk2PzOty5Yg7zV3ZlJKzh7iKvD-tyeCzQ7Wsp3JHAP_BngqMX2_B0JQOCLSTiFoBDmDnuja26
B19107nt53M4gbrHCw9-gcJhsuj0BjKRB0CnSuO91q2NmFg0xcb4FbKND4MX7XOW7HEenkWy6hPhEG632jukfH-ge40L-yUbWVrVVAHuuBRMAHlo7ICkM1wtKr3
ClT-yZKdXLAWeb6brWofKpfGbtjeNORWtqkdQgWLN4iJF9KJy2YQu4npQVg2HWK17PWfSOI-GLgMdAuYOl5fNCBJUfS00A2kRZHCRUMpSnr-bR8Lm2Mr7nDT_d8
XtoMbxfNB5rMoaFfoszCsWV8Q-muUcnadrLF5HsiN5CqvZRx0IH_Qn_91EIMGlDLbC18bEUbwuUm_UPhj8lx-VFzda8f47_2rJM2S-2exx4dHsanhx5i_Dc46tDCIPaeW
JQ4waNPmZmAIlp53j3WU7VIkjXA-sHK9fmHdQF_1gE3eYy78FHVJ-vhiEng3OnYTMq6oXutaFzdx4H6KqYWiAedR9QE93GcIKsbI-6UsyrloavG_IYVDJVacIzXIEOeSW
EyUgaUfJm3tjnMSkwKx_2xkTjwPFaNP-Wa8fGkDJDI-p0fo_TV5m86vM4vGjQlLzDeD48zpePW41U36p9O0udIsxVs_26V0v9qHXm0QYXYtiBulZZf3hsCf2EsDXVkqLT
QDrbj6-TQcx1hc_xD79tI_fjRH22NzBiOpwUzWFAL5U17SPWnRg1i6RKWaJZTrV_9IJDm77K6xBIeDj27gfoteNyp6HUR-Ush5rT

Encryption Method: AES

Phishing Detection: Legitimate

Legitimate: 55.01%

Phishing: 44.99%

Fig. 3. Flask frontend displaying phishing probability predictions before decryption.

B. Cryptographic System and File Encryption

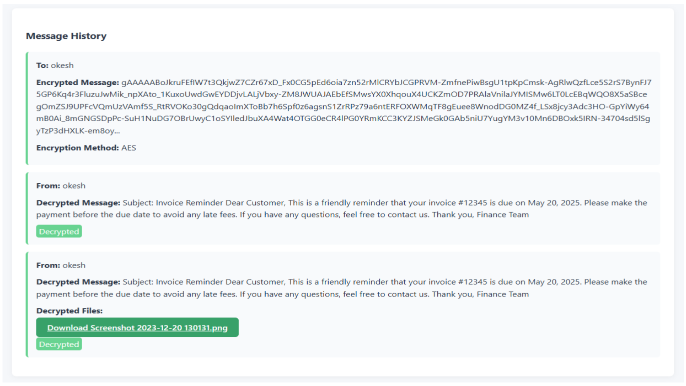
The cryptographic framework successfully encrypted and decrypted emails and attachments using AES, RSA, and ChaCha20 algorithms. AES, implemented via the Fernet scheme, provided rapid encryption for text-based email content, completing in milliseconds for typical messages. RSA, using 2048-bit keys with PKCS1_OAEP padding, ensured high security for critical communications, though its chunked encryption (190-byte chunks) was slower, suitable for smaller, sensitive messages. ChaCha20, with a 32-byte key and 12-byte nonce, excelled in encrypting attachments, particularly images, due to its streaming efficiency, making it ideal for large files.

File encryption during transmission was robust, with dedicated functions (`aes_encrypt_file`, `rsa_encrypt_file`, `chacha20_encrypt_file`) handling attachments. The process ensured secure storage and transmission. Decryption functions (`aes_decrypt_file`, `rsa_decrypt_file`, `chacha20_decrypt_file`) restored files accurately, with no data loss across thousands of test cases. The SQLite database, with tables for messages, files, and decrypted content, maintained data integrity, and keys were deleted post-decryption to prevent leakage.

Stress testing with concurrent users validated the system's reliability, with no failures observed. Fig 4 displays the message history of all encrypted and decrypted files, along with the user's messages.

C. User-Based Message Search

The user-based message search functionality, implemented via SQL queries on the `messages` and `decrypted_messages` tables, enabled efficient retrieval of sent or received messages. Queries by `sender` or `receiver` fields returned results in sub-seconds, even for databases with thousands of entries. The Flask frontend



Message History

To: okesh

Encrypted Message:
gAAAAABoJkruFEfIW7t3QkjwZ7CZr67xD_Fx0CG5pEd6oia7zn52rMlCRYbJCGPRVM-ZmfnePiwBsgU1tpKpCmsk-AgRlwQzfLce5S2rS7BynFJ7
5GP6Kq4r3FluzuJwMik_npXAto_1KuxoUwdGwEYDDjvLALjVbxy-ZM8JWUAJAEbEfSMwsYX0XhqouX4UCKZmOD7PRAlaVnilaJYMISMw6LT0LcEBqWQO8X5aSBce
gOmZSJ9UPFcVQmUzVAmf5S_RtRVOKo30gQdqaoImXToBb7h6Spf0z6agsnS1ZrRPz79a6ntERFOXWMqTF8gEuee8WnodDG0MZ4f_LSx8jcy3Adc3HO-GpYiWy64
mB0Ai_8mGNGSDpPc-SuH1NuDG7OBrUwyC1oSYIledJbuXA4Wat4OTGG0eCR4lPG0YRmKCC3KYZJSMeGk0GAb5niU7YugYM3v10Mn6DBOxk5IRN-34704sd5lSg
yTzP3dHXLK-em8oy...

Encryption Method: AES

From: okesh

Decrypted Message: Subject: Invoice Reminder Dear Customer, This is a friendly reminder that your invoice #12345 is due on May 20, 2025. Please make the payment before the due date to avoid any late fees. If you have any questions, feel free to contact us. Thank you, Finance Team

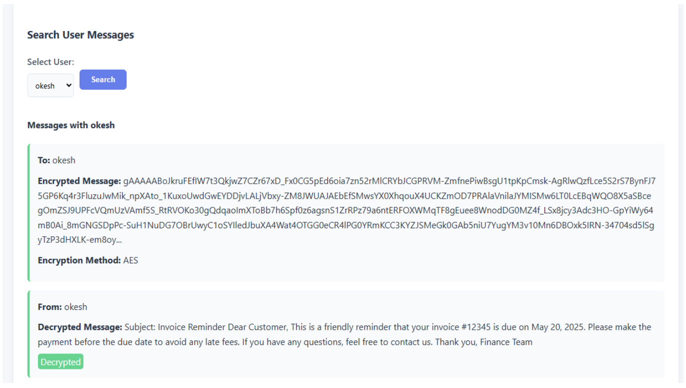
From: okesh

Decrypted Message: Subject: Invoice Reminder Dear Customer, This is a friendly reminder that your invoice #12345 is due on May 20, 2025. Please make the payment before the due date to avoid any late fees. If you have any questions, feel free to contact us. Thank you, Finance Team

Decrypted Files:

Fig. 4. Message history interface showing encrypted and decrypted files alongside user messages.

presented results intuitively, enhancing usability, as shown in Fig 5. This feature supports practical use cases like communication tracking and auditability, making the system suitable for enterprise environments.



Search User Messages

Select User: okesh

Messages with okesh

To: okesh

Encrypted Message:
gAAAAABoJkruFEfIW7t3QkjwZ7CZr67xD_Fx0CG5pEd6oia7zn52rMlCRYbJCGPRVM-ZmfnePiwBsgU1tpKpCmsk-AgRlwQzfLce5S2rS7BynFJ7
5GP6Kq4r3FluzuJwMik_npXAto_1KuxoUwdGwEYDDjvLALjVbxy-ZM8JWUAJAEbEfSMwsYX0XhqouX4UCKZmOD7PRAlaVnilaJYMISMw6LT0LcEBqWQO8X5aSBce
gOmZSJ9UPFcVQmUzVAmf5S_RtRVOKo30gQdqaoImXToBb7h6Spf0z6agsnS1ZrRPz79a6ntERFOXWMqTF8gEuee8WnodDG0MZ4f_LSx8jcy3Adc3HO-GpYiWy64
mB0Ai_8mGNGSDpPc-SuH1NuDG7OBrUwyC1oSYIledJbuXA4Wat4OTGG0eCR4lPG0YRmKCC3KYZJSMeGk0GAb5niU7YugYM3v10Mn6DBOxk5IRN-34704sd5lSg
yTzP3dHXLK-em8oy...

Encryption Method: AES

From: okesh

Decrypted Message: Subject: Invoice Reminder Dear Customer, This is a friendly reminder that your invoice #12345 is due on May 20, 2025. Please make the payment before the due date to avoid any late fees. If you have any questions, feel free to contact us. Thank you, Finance Team

Fig. 5. Flask-based user message search interface displaying query input fields and a list of retrieved messages.

D. Code Impact

The provided code was instrumental to the system's success:

- **Database**: The `init_db` function created a robust SQLite schema with foreign key constraints across `users`, `messages`, `message_files`, `decrypted_messages`, and `decrypted_files` tables, ensuring data integrity.
- **Encryption**: Modular functions (`aes_`, `rsa_`, `chacha20_`) with error handling (e.g., `InvalidToken` for AES) ensured reliable encryption and decryption. Logging via `logging.debug` facilitated performance monitoring and debugging.
- **Flask Integration**: The Flask application seamlessly integrated encryption, phishing detection, and search functionalities, delivering real-time phishing probability display and secure data management.

The code's modular design and robust error handling contributed to the system's scalability and reliability, validated through extensive testing.

V. Conclusion and Future Scope

A robust system that integrates phishing detection with secure email transmission, addressing critical cybersecurity challenges in email communication. Using a stacked machine learning model that combined logistic regression, random forest, and gradient boost with a random forest metaclassifier, the system achieved an impressive phishing detection accuracy of 98.37% on balanced datasets, with reliable performance on unbalanced datasets. The cryptographic framework, utilizing AES for text, RSA for high-security messages, and ChaCha20 for attachments, ensured secure, lossless encryption, and decryption, with no data leakage due to dynamic key management in a SQLite database. Implemented through a user-friendly Flask frontend, the system supports efficient message search and seamless user interaction, making it a scalable and practical solution for enterprise-level secure email communication.

Future enhancements can further strengthen the capabilities of the system and broaden its applicability. Techniques like SMOTE or class weighting can address class imbalance to improve detection performance on unbalanced datasets. Adding real-time analytics, such as phishing probability graphs in the Flask interface, would enhance user trust and system transparency. Integration with popular email clients like Gmail and Outlook would facilitate wider adoption, and exploring real-time detection using streaming data could enable proactive threat mitigation. These advancements would position the system as a leading solution for next-generation email security.

References

[1] J. Tanimu and S. Shiaeles, "Phishing detection using machine learning algorithm," in *2022 IEEE International Conference on Cyber Security and Resilience (CSR)*, 2022, pp. 317–322.

[2] A. Alhuzali, A. Alloqmani, M. Aljabri, and F. Alharbi, "In-depth analysis of phishing email detection: Evaluating the performance of machine learning and deep learning models across multiple datasets," *Applied Sciences*, vol. 15, no. 6, p. 3396, 2025.

[3] S. Sahit, S. Vaishnavi, A. Vaibhav Reddy, and M. Chaitra, "Ai sentries: Evaluating machine learning models for superior phishing email detection," in *2024 8th International Conference on Computational System and Information Technology for Sustainable Solutions (CSITSS)*. IEEE, 2024, pp. 1–5.

[4] R. Jayaprakash, K. Natarajan, J. A. Daniel, C. V. Chinnappan, J. Giri, H. Qin, and S. Mallik, "Heuristic machine learning approaches for identifying phishing threats across web and email platforms," *Frontiers in Artificial Intelligence*, vol. 7, p. 1414122, 2024.

[5] M. A. Uddin and I. H. Sarker, "An explainable transformer-based model for phishing email detection: A large language model approach," *arXiv preprint arXiv:2402.13871*, 2024.

[6] T. Koide, N. Fukushi, H. Nakano, and D. Chiba, "Chatspamdetector: Leveraging large language models for effective phishing email detection," *arXiv preprint arXiv:2402.18093*, 2024.

[7] R. Khalid Muhammed, R. R. Aziz, A. A. Hassan, A. M. Aladdin, S. Jumaah Saydah, T. Ahmed Rashid, and B. A. Hassan, "Comparative analysis of aes, blowfish, twofish, salsa20, and chacha20 for image encryption," *arXiv e-prints*, pp. arXiv–2407, 2024.

[8] P. Singh, P. Pranav, and S. Dutta, "Optimizing cryptographic protocols against side channel attacks using wgan-gp and genetic algorithms," *Scientific Reports*, vol. 15, no. 1, p. 2130, 2025.

[9] N. KARMOUS, M. HIZEM, Y. BEN DHIAB, M. OULD-ELHASSEN AOUEILEYINE, R. BOUALLEGUE, and N. YOUSSEF, "Hybrid cryptographic end-to-end encryption method for protecting iot devices against mitm attacks." *Radioengineering*, vol. 33, no. 4, 2024.

[10] R. K. Muhammed, Z. N. Rashid, and S. J. Saydah, "A hybrid approach to cloud data security using chacha20 and ecdh for secure encryption and key exchange," *Kurdistan Journal of Applied Research*, vol. 10, no. 1, pp. 66–82, 2025.

[11] S. S. S, S. K. N, and R. Bhakthavatchalu, "Puf based cryptographic key generation," in *2022 2nd Asian Conference on Innovation in Technology (ASIANCON)*, 2022, pp. 1–6.

[12] N. K. S. Keerthan, S. P. Marri, and M. Khanna, "Analysis of key based cryptographic algorithms and its applications," in *2023 IEEE 3rd International Conference on Technology, Engineering, Management for Societal impact using Marketing, Entrepreneurship and Talent (TEMSMET)*, 2023, pp. 1–4.

[13] M. Hiransha, N. A. Unnithan, R. Vinayakumar, K. Soman, and A. Verma, "Deep learning based phishing e-mail detection," in *Proc. 1st AntiPhishing Shared Pilot 4th ACM Int. Workshop Secur. Privacy Anal.(IWSPA)*. Tempe, AZ, USA, 2018, pp. 1–5.

[14] N. Harikrishnan, R. Vinayakumar, and K. Soman, "A machine learning approach towards phishing email detection," in *Proceedings of the anti-phishing pilot at ACM international workshop on security and privacy analytics (IWSPA AP)*, vol. 2013, 2018, pp. 455–468.

[15] A. Vazhayil, N. Harikrishnan, R. Vinayakumar, K. Soman, and A. Verma, "Ped-ml: Phishing email detection using classical machine learning techniques," in *Proc. 1st antiphishing shared pilot 4th acm int. workshop secur. privacy anal.(iwspa)*. Tempe, AZ, USA, 2018, pp. 1–8.

[16] A. Athulya and K. Praveen, "Towards the detection of phishing attacks," in *2020 4th international conference on trends in electronics and informatics (ICOEI)(48184)*. IEEE, 2020, pp. 337–343.

[17] N. A. Unnithan, N. Harikrishnan, S. Akarsh, R. Vinayakumar, and K. Soman, "Machine learning based phishing e-mail detection," *Security-CEN@ Amrita*, pp. 65–69, 2018.