SocialStego: A Steganography Tool for the Modern Era of Social Media

Branden Palacio
Marquette University
branden.palacio@marquette.edu

Keyang Yu
Marquette University
keyang.yu@marquette.edu

***Abstract*—The widespread availability and routine use of social media platforms have created new opportunities for covert communication over channels that are often permitted within organizational networks. This work presents SocialStego, a proof-of-concept system that demonstrates how limited social media security policies can be exploited by an insider to exfiltrate sensitive information without violating nominal access controls. Adopting an insider-threat perspective, SocialStego combines Least Significant Bit (LSB) steganography with a hybrid cryptographic scheme to protect the confidentiality of embedded payloads. Specifically, AES-256 is used for payload encryption, while RSA-2048 supports secure key exchange. A custom encoding protocol is implemented to embed encrypted data into lossless PNG image files and WAV audio files. Encoded carrier files are transmitted using existing social media and messaging infrastructure that preserves lossless media formats. The system examines the trade-offs between embedding capacity and perceptual distortion, showing that WAV carriers support higher payload capacity under the proposed design due to their variable duration, while increasing the LSB bit depth introduces more noticeable and potentially detectable noise artifacts in the carrier. Collectively, these findings demonstrate the feasibility and associated risks of covert data exfiltration via commonly accessible social media channels and highlight the need for organizations to account for such mechanisms when developing security policies and controls.**

I. Introduction

This work examines a security gap arising from the interaction between organizational social media policies and internal network access controls. In practice, many organizations maintain social media policies that primarily regulate employee behavior and public-facing content, such as discouraging controversial political commentary, hate speech, or offensive remarks. However, organizations tend to place comparatively less emphasis on technical controls governing how social media platforms are accessed from within internal networks. At the same time, social media and messaging platforms often remain accessible from organizational devices, and in some cases are integrated directly into operational workflows through tools such as Teams, Discord, or Slack. This combination creates a gap in which users may lawfully access social media services from internal devices, yet the organization lacks effective mechanisms to monitor or restrict how those platforms are used as communication channels.

This work takes the perspective of an insider threat, whose objective is to extract sensitive information from a workplace internal network and distribute this information to an external third-party. Within this threat model, the recipient may be a trusted external party, the insider operating from a personal network, or a third-party organization interested in the exfiltrated data. The focus is not on exploiting software vulnerabilities, but rather on leveraging permitted communication channels in unintended ways.

Social media platforms are particularly attractive for this purpose due to the volume and ubiquity of media shared daily, especially images and audio files. This observation motivates the use of steganography—the embedding of information within seemingly benign carrier files—as the primary mechanism for covert data exfiltration. By encoding sensitive information into commonly shared media formats and transmitting them through existing social media infrastructure, an insider may bypass conventional content-based monitoring while remaining within nominal access authorizations.

A. Motivation and Insights

The primary motivation of this project is to inform security practitioners and policymakers of the risks associated with allowing unrestricted access to social media platforms from organizational devices. An insider with access to common social media services and basic steganographic tooling can exfiltrate sensitive information without triggering traditional security controls, potentially causing significant harm to the organization.

Rather than prescribing a single defensive solution, this work aims to illustrate the feasibility and implications of such an attack vector. Mitigation strategies necessarily depend on organizational context, risk tolerance, and operational requirements. While restricting access to social media platforms from internal networks is one possible response, effective defenses require careful consideration by security teams, including technical controls, policy enforcement, and user education.

B. Overview of Paper Organization

The remainder of this paper is organized as follows:

Workshop on Attack Provenance, Reasoning, and Investigation for Security in the Monitored Environment (PRISM) 2026
23 February 2026, San Diego, CA, USA
ISBN 978-1-970672-00-8
https://dx.doi.org/10.14722/prism.2026.23016
www.ndss-symposium.org

- **Background**: We introduce foundational concepts including Least Significant Bit (LSB) steganography, covert channels, and lossless media formats. We also review relevant prior work and clarify how this project differs from existing steganographic tools and studies.
- **Challenges**:We describe the key challenges encountered during system development, including carrier format selection, platform behavior constraints, automation limitations, and payload confidentiality considerations.
- **Design**: We present the architectural and methodological design decisions underlying the SocialStego system, including encoding and decoding workflows, configurable embedding parameters, and the custom encoding protocol.
- **Implementation**: We detail the practical implementation of the system, focusing on encoding pipelines, cryptographic integration, file handling, and platform interaction.
- **Evaluation**:We evaluate the system through qualitative steganalysis, encrypted payload extraction analysis, and analytical data throughput comparisons across carrier types.
- **Conclusion**: We summarize the results of the SocialStego proof of concept, discuss its limitations, and outline directions for future work.

II. Background

We begin by providing necessary context and background on the important concepts used in addressing the proposed problem of data extraction. From this, we discuss prior work in the field of Steganography tools and describe where this project differs from previously performed exercises. Lastly, we will describe some of the challenges faced during the construction of this project solution and how they were addressed.

A. Technical Background and Definitions

*1) **Steganography**:* The National Institute of Standards and Technology (NIST) defines Steganography as the "art and science of communicating in a way that hides the existence of the communication" [1]. Although steganographic techniques have existed for centuries and were historically exemplified by methods such as invisible ink [2], modern steganography is now primarily applied to digital media to conceal information transmitted over networks.

A common digital steganographic approach is Least Significant Bit (LSB) steganography, in which the least significant bits of a carrier medium are modified to encode payload data. In image-based steganography, this typically involves altering the least significant bits of RGB pixel values, while in audio-based steganography, payloads may be embedded into the least significant bits of audio sample frames. Because these bits contribute minimally to perceptual output, such as pixel color intensity or audio amplitude, LSB modification can be visually or audibly imperceptible when low embedding rates are used.

Figure 1 illustrates this process for a single RGB pixel group in a PNG image, where the least significant bits of the red, green, and blue channels are modified to encode a character while preserving the overall appearance of the pixel [3]. For the purposes of this project, PNG image files and WAV audio files were selected as carrier media due to their support for lossless data representation.

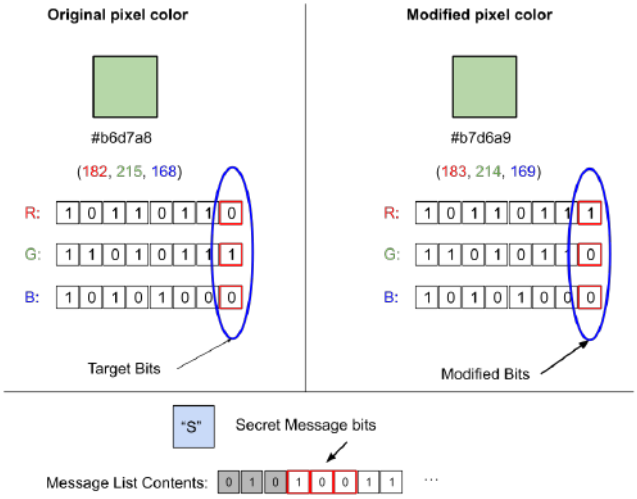

Fig. 1: Illustration of LSB modification in an RGB PNG pixel.

*2) **Covert Channels**:* NIST defines a covert channel as an "unintended or unauthorized intra-system channel that enables two cooperating entities to transfer information in a way that violates the system's security policy but does not exceed the entities' access authorizations" [4]. Covert channels are commonly classified as either storage channels or timing channels. Storage channels convey information through modification of shared system state, such as file contents or memory values, whereas timing channels encode information by introducing variations in observable system behavior over time [5], [6].

In the context of this work, steganography operates as a covert storage channel in which sensitive information is embedded within otherwise legitimate media files. As discussed in the Introduction, organizational policies that focus primarily on regulating the semantic content of user activity, rather than the underlying communication mechanisms, can inadvertently allow such channels to persist. The SocialStego prototype illustrates this scenario by encoding sensitive data into media files that are transmitted through permitted social media services.

*3) **Lossless Media Types**:* Lossless media formats employ compression techniques that reduce file size without discarding information, thereby allowing the original data to be reconstructed exactly [7]. Although such formats are commonly valued for preserving media quality, they are particularly important for steganographic systems that depend on deterministic bit-level manipulation. When lossy formats are used as carrier files, re-encoding or recompression can alter bit-level representations and consequently corrupt or destroy embedded payloads.

Accordingly, this project focuses on PNG image files and WAV audio files as carrier media, while recognizing that

TABLE I: Examples of Lossless File Types

Image [8]	Audio [9]	Video [10]
PNG	WAV	FFV1
BMP	Free Lossless Audio Codec (FLAC)	HuffYUV
TIFF	Apple Lossless Audio Codec (ALAC)	Logarith
GIF		
RAW		

additional lossless formats may be considered in future work. Table I summarizes representative examples of lossless image, audio, and video file types.

In addition to file format selection, the behavior of social media platforms with respect to uploaded media must also be considered. Even when lossless formats are accepted, some platforms may apply format conversion or recompression during storage or delivery. To address this concern, we conducted a preliminary investigation of platform support for lossless media upload and storage, as summarized in Table II. Based on these observations, the scope of this proof-of-concept evaluation was limited to platforms that preserve lossless PNG and WAV files, specifically X (formerly Twitter) and Discord.

B. Prior Work

Steganography research spans diverse carriers (images, audio), embedding domains (spatial vs. transform), and channel assumptions (lossless storage vs. platform-induced transformations). For the purposes of this project—LSB-style modification of RGB values in lossless carriers and evaluation on platforms that preserve uploads—we summarize prior work in four categories: (1) spatial-domain image steganography, (2) audio-based hiding, (3) social media as a communication channel, and (4) approaches commonly used for lossy/transformed environments.

1) Spatial-Domain Steganography on Lossless Images: Spatial-domain image steganography is among the earliest and most widely studied forms of information hiding. In this setting, secret data are embedded by directly modifying pixel values, most commonly using Least Significant Bit (LSB) replacement. Chandramouli et al. analyze a range of LSB-based image steganography techniques and their statistical properties, helping establish how LSB modifications can become detectable depending on embedding strategy and cover characteristics [17]. In parallel, Provos et al. provide a broad introduction to practical steganography, emphasizing the core trade-offs among capacity, detectability, and robustness, and surveying common system designs and threat models [18].

More recent survey work consolidates spatial-domain methods and highlights how design choices (e.g., embedding location selection, number of modified bits, use of encryption/compression, and image content characteristics) affect visual quality and statistical detectability. Hussain et al. specifically survey spatial-domain image steganography techniques and summarize representative approaches and evaluation practices used in the literature [19]. These works collectively motivate the core design pattern used in this project: direct modification of pixel-channel values (RGB) with careful control of how many bits are overwritten.

2) Audio-Based Hiding with Uncompressed Carriers: Not limiting to images, steganography also leverages audio carriers to share hidden message. Specifically, audio carriers provide substantial capacity due to the large number samples in typical recordings. The sample rates range from 16 kHz for voice/telephony, to 44.1 kHz (CD, music streaming), and 48 kHz as video standard. Foundational work on multimedia data hiding discusses techniques for embedding information in digital media, analyzed the constraints, and evaluated the system-level trade-offs between embedding rate and robustness to processing [20]. In practice, many educational and proof-of-concept systems focus on uncompressed audio (e.g., WAV/PCM), since sample-level modifications (often LSB-like) can be recovered deterministically as long as the carrier is not transcoded.

Beyond simple LSB-style embedding, transform-domain approaches have also been explored for media that undergo compression. For example, Chang et al. propose reversible hiding in DCT-based compressed images, exploring how embedding in transform coefficients can be used to survive compression workflows more natually than raw spatial-domain modifications [21]. Although [21] targets JPEG/DCT images rather than audio, it is representative of the broader principle used in lossy domains: embed in the representation most stable under the expected channel.

3) Social Media and Messaging Platforms as Channels: Social media platforms have been proven suitable not merely as a file host but as a communication channel with undocumented transforms (e.g., recompression, resizing, metadata changes). Tierney et al. proposed Cryptagram, focused on photo privacy rather than classic steganography. Cryptagram formalizes and evaluates designs intended to remain usable under social-network image processing pipelines, which contain notably lossy transformations [22]. More directly, Ning et al. study secret message sharing using online social media and demonstrate that end-to-end feasibility depends strongly on platform behavior; some pipelines preserve sufficient structure for hidden payloads, while others disrupt them [23]. These studies support the common systems-security framing: platform processing effectively becomes part of the adversarial or noisy channel, and survivability must be evaluated per platform and upload path. Motivated by these findings, the present work explicitly limits its scope to platforms and upload modes that preserve lossless PNG and WAV files. Rather than attempting to survive arbitrary platform-induced transformations, the goal is to demonstrate that simple spatial-domain steganography remains viable when lossless transmission paths are available.

4) Common Techniques for Lossy or Transformed Environments: When the channel is lossy or applies transformations, direct spatial-domain LSB embedding is often brittle. A representative family of approaches addresses this by defining embedding distortion and adaptively placing changes in less detectable or more resilient regions. Holub et al. propose

TABLE II: Social Media Upload and Storage Support

Social Media Outlet	Supports Lossless PNG/ WAV Upload	Supports PNG/ WAV Storage	Notes
Facebook/ Instagram (*) [11] [12]	**PNG**: Accepted **WAV**: Direct posting not accepted	**PNG**: May convert uploaded files to JPG **WAV**: Direct posting not accepted	**PNG**: Accepted but may convert to JPG **WAV**: Not directly supported for uploads, must be converted to a lossless video format or uploaded through a file sharing service and linked
Reddit [13]	**PNG**: Accepted **WAV**: Direct posting not accepted	**PNG**: Untested behavior **WAV**: Direct posting not accepted	**PNG**: Accepted for uploads but will likely be converted to JPG or Webp **WAV**: Not directly supported and must be converted to a lossless video format or uploaded through a file sharing service and linked
Soundcloud [14] [15]	**PNG**: Accepted for track artwork **WAV**: Accepted	**PNG**: Untested behavior **WAV**: File may be transcoded or maintain original data quality	It is recommended to upload WAV files to SoundCloud as this will maintain the highest sound quality. PNG files can be included as the track artwork.
X (*) [16]	**PNG**: Accepted **WAV**: Not accepted	**PNG**: Maintains original file type **WAV**: Not accepted	**PNG**: Supported as uploads and maintain their original file type **WAV**: Not supported and must be converted to a lossless video format or uploaded through a file sharing service and linked
Discord (*)	**PNG**: Accepted **WAV**: Accepted	**PNG**: Maintains original file type **WAV**: Maintains original file type	Both PNG and WAV files are supported as uploads and maintain their original file type

(*) Independently verified through experimentation.

defining steganographic distortion using directional filters, enabling adaptive embedding in the spatial domain guided by content-dependent costs [24]. Holub et al. introduce universal distortion for digital image steganography, continuing this line by designing embedding costs intended to generalize across content [25]. Separately, Pevný et al. demonstrate high-dimensional image models to achieve highly undetectable steganography, representing a modern direction where embedding strategies are informed by richer statistical models of images [26].

For this project, these works serve primarily as context: they illustrate the techniques typically used when one must withstand aggressive processing or evade stronger steganalysis, and they motivate future extensions such as transform-domain embedding and channel-specific robustness. However, consistent with the current evaluation focus, we treat lossy/transformed channels as out of scope and include them as future work rather than a claimed capability.

C. Summary

This section establishes the technical foundations required to understand the problem space addressed by SocialStego. We introduce steganography as a mechanism for covert communication that conceals the existence of information exchange, with particular focus on Least Significant Bit (LSB) techniques that embed data by minimally modifying carrier media. LSB-based approaches are attractive due to their simplicity and high embedding capacity; however, their correctness depends critically on the preservation of bit-level media representations during storage and transmission. We therefore emphasize the role of lossless media formats, such as PNG images and WAV audio files, which enable deterministic payload recovery and serve as practical carriers when transformation-free transmission paths are available.

We further situate steganography within the broader concept of covert channels, illustrating how permitted communication mechanisms—such as social media and messaging platforms accessible from internal networks—can unintentionally facilitate unauthorized data exfiltration. Prior work demonstrates that platform-specific media processing (e.g., recompression or format conversion) significantly affects steganographic viability, motivating careful channel and carrier selection. Finally, we review existing literature across spatial-domain image

steganography, audio-based hiding, and social media–based communication, clarifying how these studies inform the assumptions and constraints adopted in this work. Together, this background motivates the design challenges addressed in the following section, including carrier selection, platform behavior, automation feasibility, and payload confidentiality.

III. Challenges and Design Constraints

The development of SocialStego was guided by three primary challenges, each of which directly influenced the system scope and design decisions.

The first challenge concerned the interaction between steganographic embedding and media compression. Many commonly used image and audio formats employ lossy compression, which can alter or discard least-significant-bit information and thereby corrupt embedded payloads. To mitigate this risk, we conducted a preliminary investigation into media formats and platform upload behaviors, with the goal of identifying lossless carrier types that are preserved during social media transmission. This analysis led us to focus on PNG images and WAV audio files, both of which support lossless storage and deterministic bit-level manipulation. In parallel, this investigation informed platform selection by identifying social media services that preserve these formats during upload and storage. Based on these criteria, X (formerly Twitter) and Discord were selected as the primary platforms for evaluation, as both support lossless PNG uploads and, in the case of Discord, lossless WAV file sharing.

The second challenge involved automation of social media posting. One of the initial design goals was to support automated posting of encoded carrier files using platform-provided developer APIs. Discord provides a mature developer ecosystem that enables the creation of bots with granular permissions for server management and content posting. Using these APIs, we successfully deployed a bot in a private Discord server capable of automatically uploading encoded files to designated channels. However, a key limitation of this approach is that bot-generated content is explicitly labeled as such, which may reduce realism in threat modeling scenarios where posts are expected to originate from human-operated accounts. We did not identify a supported mechanism for automated posting from a standard user account on Discord.

In contrast, X provides developer APIs that do not inherently label posts as automated nor require accounts to be registered as bots for API-based posting [27]. Despite this, we were unable to complete reliable automated posting to X within the project timeline. Given these constraints, we prioritized core steganographic functionality—such as payload encryption, protocol design, and encoding flexibility—over full automation. To validate feasibility, we instead performed manual posting using a dedicated account and confirmed successful upload and recovery of encoded images. Automation for X remains an avenue for future work.

The final major challenge concerned data confidentiality and obfuscation. A well-known limitation of LSB-based steganography is that it provides no inherent protection for the embedded payload once extraction occurs; security relies primarily on obscurity rather than cryptographic guarantees. This assumption is insufficient in adversarial settings where an analyst may suspect the presence of hidden data and attempt extraction using readily available tools. To address this limitation, SocialStego incorporates a hybrid encryption scheme combining symmetric and asymmetric cryptography. Payload data are encrypted prior to embedding, ensuring that even if an unintended party successfully extracts the hidden bitstream, the recovered data remain unintelligible without the appropriate cryptographic keys. This design choice decouples payload confidentiality from the secrecy of the embedding mechanism and aligns the system more closely with realistic threat models involving partial compromise or forensic analysis.

Together, these challenges motivated a design that emphasizes lossless carriers, platform-aware deployment, and cryptographic protection, while deliberately limiting scope to maintain clarity and feasibility within a course-project setting.

IV. Design

This section provides a detailed account for the design considerations implemented in the SocialStego tool and discusses the significance of these decisions in achieving the project's goal of covert data exfiltration using lossless media and existing social media infrastructure. We start with introducing the threat model and assumptions.

A. Threat Model and Assumptions

This work adopts a constrained threat model consistent with insider-driven data exfiltration scenarios. We assume an adversary with authorized access to internal systems and permitted access to social media platforms from within the organizational network. The adversary seeks to exfiltrate sensitive data without triggering conventional content-based monitoring or access-control violations.

We assume that the communication channel preserves lossless carrier formats and that the recipient possesses knowledge of the steganographic protocol and decryption keys. Detection by advanced steganalysis tools, active wardens, or platform-level countermeasures is considered out of scope for this work.

We explicitly do not consider adversaries equipped with active wardens, advanced steganalysis capabilities, or platform-level transformations that intentionally modify uploaded media. Similarly, this work does not attempt to evade content moderation systems or forensic detection tools. Instead, the threat model focuses on scenarios in which an insider leverages permitted communication channels and carrier-preserving media formats to exfiltrate data without violating nominal access controls.

B. Encode & Decode Functionality

The primary design criterion was to provide both encoding and decoding functionality for PNG image files and WAV audio files. These capabilities are provided to ensure that

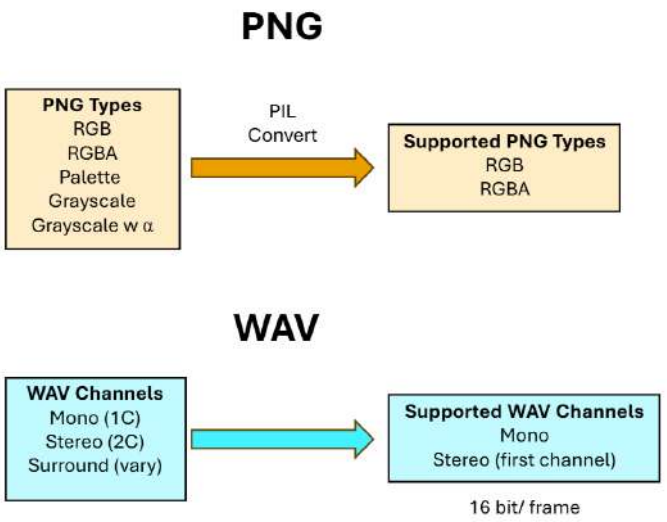

Fig. 2: Simplified visualization for the encoding flow for each of the supported file types

SocialStego functions as a complete proof-of-concept steganographic system rather than a one-way encoder. Program control flow is determined by required command-line flags specifying the operational mode (`-e --encode, -d --decode`).

PNG files exist in multiple internal representations, including RGB, RGBA, palette-based, and grayscale formats, each of which follows a different data layout within the file structure [28]. To simplify encoding logic and ensure consistent handling across PNG variants, the Python Image Library's `convert()` function was used to normalize all non-RGB/RGBA PNG files into RGB format prior to embedding [29]. This design choice enables a single, uniform spatial-domain encoding strategy across all supported image inputs.

For both PNG and WAV files, LSB Steganography was implemented. In the case of PNG images, the RGB channel values of each pixel are modified to encode payload bits, consistent with the approach illustrated in Figure 1. WAV files, however, require special handling due to their channel-based audio structure. WAV files may contain one or more channels (e.g., mono, stereo, or multichannel audio). For simplicity and consistency, payload data are embedded exclusively into the first channel when multiple channels are present. During encoding, each audio frame is loaded as a 16-bit integer, and the selected least significant bits are overwritten to store payload data before being written back to the output file.

Figure 2 illustrates the high-level encoding flow for both supported file types.

C. File Selection

File selection was implemented to support both command-line and interactive workflows. Users may specify input files directly via the optional `-f` , followed by the relevant file paths (carrier and payload files for encoding, or a single carrier file for decoding). If this flag is omitted, a graphical file selection interface is provided using the Python `Tkinter` library [30]. This dual approach improves usability while preserving scriptability for automated or batch-based use.

D. File Type Identifier

To improve control flow and decoding reliability, SocialStego incorporates a file type identifier mechanism based on file header inspection. A Python dictionary maps supported file types (e.g., PNG, WAV, PDF, GZ) to their corresponding magic bytes. This mechanism serves two purposes: first, it ensures that only supported file types are accepted during encoding; second, it enables automatic reconstruction of extracted payloads by appending the correct file extension during decoding. If a payload's magic bytes are not recognized, the extracted file is treated as plain text by default. The dictionary-based design allows straightforward extension to additional file types in future iterations.

E. Output File Naming

An optional `-o` flag allows users to specify custom output file names for both encoding and decoding workflows. Because both operations generate new files, this option improves usability without affecting encoding logic. Users are not required to specify file extensions explicitly, as the file type identifier described in Section IV-C the appropriate extension automatically during output generation.

F. LSB Bit Count

To provide finer-grained control over embedding capacity and distortion, SocialStego supports configurable LSB bit depth via the optional `-b` flag. This parameter specifies the number of least significant bits modified per RGB byte (for PNG) or per audio frame (for WAV). The default configuration embeds one bit per unit; however, users may specify values from 1 to 4, increasing payload capacity at the cost of greater perceptual noise. Figure 3 visualizes the range of modifiable bits enabled by this parameter and illustrates the trade-off between embedding density and cover distortion.

G. Encoding Across Multiple File Types

Although PNG images and WAV audio files differ significantly in structure, SocialStego employs a unified conceptual encoding pipeline across both types of carriers. In each case, the payload data is serialized into a bitstream, sequentially embedded into eligible carrier elements (pixels or audio frames) and reconstructed during decoding using identical control logic. File-type-specific differences are encapsulated within modular encoding routines, allowing the broader system design—including protocol handling, encryption, and metadata processing, to remain consistent across carriers.

H. Custom Encoding Protocol

With the number of design considerations implemented in this tool, it was essential to determine an effective way to encode the carrier files while providing sufficient context to the recipient regarding the encoding style. These specific considerations led to the development of a custom encoding protocol we refer to as Stegocol.

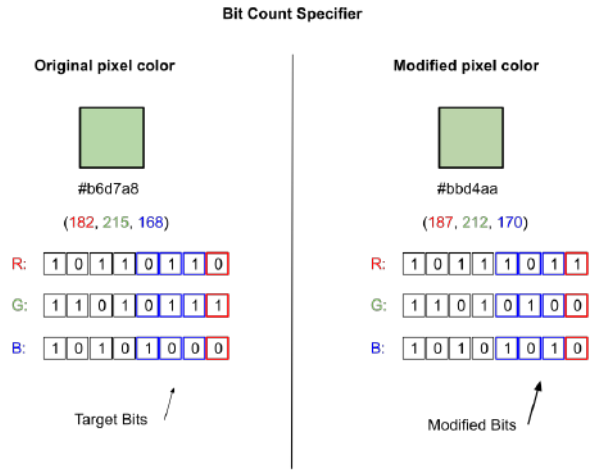

Fig. 3: Provides a visualization for the span of modifiable bits when using the bit count specifier during the encoding process

Stegocol is constructed of 5 distinct sections, starting with the payload length identifier, which is a 32-bit integer used to denote the size of the payload data. Second, is the 2-bit 'bit count' specifier, which denotes the number of significant bits encoded per pixel or frame of a PNG or WAV file, with the default binary b'00' mapped to 1 LSB encoding. Next are the 2 bits that denote 'custom flags', with one indicating compression of the payload data and the other indicating encryption of the payload data. The last section of the header is a 12-bit 'checksum' calculated using the binascii crc_hqx function exclusively on the 'payload' data [31]. These 4 sections make up the 48-bit header of the protocol, with the final section being the data bits of the payload. Important to note with this design is that because the default bit count encoding is 1-bit, even for bit count values 2-4, the 48 header bits are encoded into the carrier file using 1-bit LSB Steganography. This consistently allows the receiver to extract the protocol header, interpret the information (bit count, compression, encoding, etc.), and use the extracted header to determine the rest of the control flow.

I. Stegocol Design Rationale

The Stegocol protocol was designed to provide sufficient decoding context while minimizing additional embedding overhead and complexity. A fixed-length header was selected to ensure deterministic extraction of protocol metadata prior to payload decoding, independent of user-configured embedding parameters. Encoding the header exclusively using single-bit LSB modification ensures that protocol information can be reliably recovered even when higher bit depths are used for payload embedding.

The inclusion of an explicit payload length field allows the decoder to distinguish embedded data from residual carrier content without relying on sentinel values or external metadata. Similarly, the bit-count specifier enables adaptive decoding while supporting multiple embedding configurations within a single implementation. The checksum field provides lightweight integrity verification of the extracted payload, allowing receivers to detect corruption caused by carrier modification or transmission errors without introducing significant overhead.

Finally, protocol flags were incorporated to support optional encryption and future extensions such as compression. This design allows the encoding format to remain extensible while maintaining backward compatibility and avoiding reliance on carrier-specific assumptions.

J. Implementing Encryption

Previously mentioned as one of the significant limiting factors for most other Steganography tools, encryption was at the forefront of design considerations during the development of the SocialStego tool. The creation of the custom protocol allowed for the simple incorporation of encryption into the already defined encoding flow of the program. I utilized a hybrid encryption approach using both symmetric and asymmetric cryptography.

First, a random 32-byte symmetric key and 16-byte initialization vector were generated using the Python OS library [32]. Next, the symmetric key and initialization vector were used to generate an AES_256 cipher in Output Feedback Mode (OFB) using the Python Cryptography library [33]. AES_256 was used due to its recognition as a strong symmetric key encryption, while OFB mode was chosen because padding is not required due to OFB's functionality as a stream cipher [34]. Not having to incorporate padding created a simpler solution, and OFB mode also allows for parallelized decryption, decreasing the amount of time needed by the receiver to decrypt the encoded file. Once generated, the payload data was encrypted using the constructed AES_256 cipher.

From here, it was assumed that the insider already had access to the receiver's public key through a previously established back-channel. A thought experiment proposed by prior work considered the encoding of a recipient's public key into the recipient's profile picture, by which, the insider could extract the key to use for encrypting messages to the recipient [35]. For the contents of this proof-of-concept project, RSA_2048 was used as the PKI algorithm, again due to its recognition as an industry leader for asymmetric encryption [36]. The AES symmetric key was then encrypted using the receiver's public key, and the encrypted public key, the initialization vector, and the encrypted payload were combined into a bitsream and encapsulated as the 'new' payload data - continuing the encoding workflow and successfully encrypting the payload data.

K. Social Media Posting

The final design consideration was the incorporation of automated social media posting. Although many social media sites have developer APIs that permit users to automate posting on their platforms, the process for validation and approval requires the existence of verified accounts. This was a process considered outside the scope of this project, and therefore, was refined to automate Discord posting in a privately owned server

and to attempt manual posting on X. The Python Discord library was used to automate posting to a private Discord server using credentials stored in a local config.json file [37].

V. Implementation

This section describes the implementation details of the SocialStego prototype, focusing on practical considerations related to file handling, encoding workflows, cryptographic integration, and interaction with external platforms. The implementation prioritizes modularity and clarity, allowing individual components such as carrier parsing, payload encoding, encryption, and protocol handling to be developed and tested independently. All functionality was implemented in Python, leveraging standard libraries where possible to reduce complexity and improve reproducibility.

A. System Architecture

The SocialStego implementation follows a modular pipeline architecture consisting of four primary stages: (i) input processing, (ii) payload preparation, (iii) carrier encoding or decoding, and (iv) output generation. Input processing handles carrier normalization and file-type validation, ensuring that all inputs conform to supported formats before encoding begins. Payload preparation includes optional encryption and protocol header construction. The encoding stage performs bit-level modification of carrier data structures, while the output stage reconstructs files and metadata for storage or transmission.

This separation of concerns enables consistent handling across both PNG and WAV carriers, despite their differing internal representations. File-type–specific logic is isolated within dedicated modules, while shared components such as protocol parsing and encryption routines are reused across carrier types.

B. Encoding Pipeline

During encoding, payload data are first serialized into a contiguous bitstream, preceded by the Stegocol protocol header. For image carriers, pixel data are accessed in raster order, and selected least significant bits of RGB channel values are overwritten sequentially until the payload is fully embedded. For audio carriers, WAV frames are processed as 16-bit integers, and embedding is performed on a per-frame basis within the selected channel.

Decoding reverses this process by first extracting the fixed-length protocol header using single-bit LSB decoding, allowing the receiver to determine payload length, bit depth, and encryption status before extracting the remainder of the payload. This staged decoding process prevents ambiguity and enables deterministic recovery of embedded data.

C. Cryptographic Integration

Cryptographic operations are integrated directly into the encoding pipeline rather than applied as a post-processing step. Payload encryption occurs prior to carrier embedding, ensuring that cryptographic confidentiality is independent of steganographic concealment. Symmetric encryption is used for efficiency on large payloads, while asymmetric encryption is employed solely for key exchange. This design minimizes computational overhead while preserving confidentiality even if payload extraction is successful.

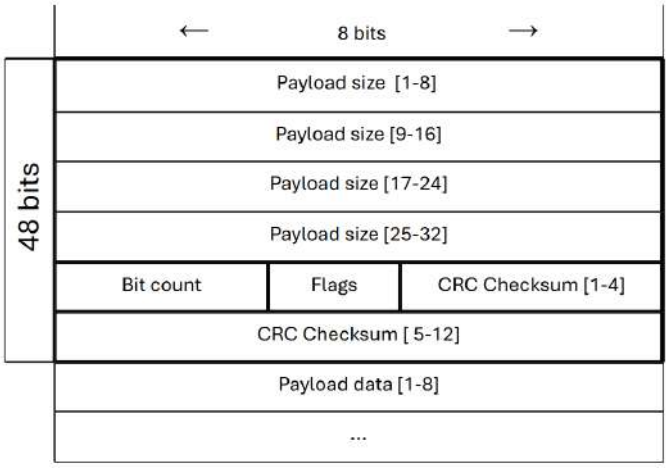

Fig. 4: Stegocol: Provides a visualization for the custom protocol developed to encode information into a carrier file in the SocialStego tool.

D. Platform Interaction

SocialStego includes optional functionality for posting encoded carriers to social media platforms. Platform interaction is intentionally decoupled from encoding logic to accommodate differing platform constraints. Automated posting is supported for Discord via bot-based APIs, while posting to X was evaluated through manual interaction. These mechanisms were used solely to validate feasibility and were not designed to evade platform moderation or detection systems.

VI. Evaluation

This section of the report analyzes the results of the SocialStego tool focusing on three distinct areas: Steganographic analysis, encryption evaluation, and a comparison of data throughput based on file type.

A. Evaluation Methodology

The evaluation of SocialStego is intentionally scoped to qualitative and analytical assessments appropriate for a proof-of-concept system. Rather than measuring resistance to advanced steganalysis or platform-level countermeasures, the evaluation focuses on three practical aspects: (i) the observable impact of LSB embedding on carrier media, (ii) the role of encryption in protecting extracted payloads, and (iii) the theoretical data capacity supported by different carrier types.

Steganographic impact is assessed through visual inspection of least-significant-bit representations of encoded PNG images under different bit-depth configurations. Payload confidentiality is evaluated by attempting extraction using a publicly available steganography decoding tool and comparing outcomes for encrypted and unencrypted payloads. Finally, data throughput

is analyzed analytically by deriving capacity equations for PNG and WAV carriers based on their structural properties. Together, these evaluations demonstrate the feasibility, limitations, and trade-offs of the proposed system without claiming robustness against active detection or transformation.

B. Steganalysis

One of the primary limitations of implementing LSB Steganography is the predictable impact encoding has on the carrier file. Having the ability to compare the original file contents to an encoded file presents an opportunity to identify patterns that would suggest a file contains encoded information. To evaluate the identifiable impact of SocialStego encoded files, we visually inspected the LSB half of two example PNG files based on their encoding orientation: original file, 1 bit encrypted, or 4 bit encrypted. This was done to determine discernible differences between the files as a result of encoding style.

As shown in Figure 5, comparing the LSB-half of the original files against their encoded counterparts, a discernible pattern emerges from these encoded images in the form of noise. viewing the 1-bit encrypted examples, we can see that this encoding style resulted in fairly minor noise in the encoded bytes allowing the original image to still be identifiable. However, looking at both the top and bottom examples, implementing 4-bit encrypted encoding resulted in significantly greater noise in the carrier file which would highly suggest the existence of encoded information within the file. Interesting to note was that the bottom 1-bit encrypted example contained noise that was somewhat difficult to verify through visual inspection. As a result, it is recommended to encode information in images that originally contain a significant amount of noise, and to use as low a bit count as necessary to encode information in order to distribute the noise throughout the image as opposed to concentrating the encoding at the top of the file.

A similar visual inspection was not performed for WAV files, however, the same recommendations are encouraged - LSB Steganography generates noise in the encoding bits, distributing this noise across a greater region of the file reduces the magnitude of noise concentration for the area housing encoded information.

C. Data extraction Analysis

As mentioned previously, one of the primary limitations of Steganography is the ease with which data can be extracted from the carrier file without encryption. Again, steganographic solutions often over-rely on the concept of data obfuscation without implementing any sort of information confidentiality practices. That said, SocialStego makes an effort to solve this limitation through the incorporation of hybrid encryption. To further assert the vulnerability of non-encrypted steganographic solutions, we attempted data extraction from a carrier file using an online Steganography tool for an unencrypted encoding and an encrypted encoding to compare the difference in data extraction capabilities. The results of which are shown in Figure 6.

As shown in Figure 6, without encryption anyone with access to a common Steganography decoder could extract encoded information, however, with encryption, even if an analyst suspects an encoding in the analyzed file, they would not be able to accurately extract the encoded file and decipher the sensitive information.

D. Data Throughput Comparison

The amount of data that can be encoded into a carrier file is entirely dependent on the carrier file type. The structure of a PNG file is consistent with that depicted in Figures 1 and 2 while the structure of a WAV file is described at the end of subsection *III-A*. The associated calculation for data throughput i.e. bandwidth (bytes/ message) is represented with the variable **B** in the following calculations.

PNG:

$$\mathrm{B} = \frac{(\text{height} \times \text{width}) \times 3 \times \text{bit count}}{8} \tag{1}$$

WAV:

$$\mathrm{B} = \frac{\text{Sample rate} \times \text{Duration} \times \text{Channels} \times \text{bit count}}{8} \tag{2}$$

With these equations it is important to note that WAV files are recommended for encoding a greater volume of information in one file. This is a result of the modifiable 'Duration' of a WAV file. The greater the duration of the audio file, the more available frames to encode. Additionally, audio files can often range between 10 seconds for a short message to even 5 minutes for a song all while remaining inconspicuous since the file type is considered to be used as intended by observers.

VII. Conclusion

SocialStego works as a proof-of-concept in verifying one's ability to encode encrypted information into carrier files and automate the export of these files using pre-existing social media infrastructure. This was verified by posting encoded images on Discord and X. The encoded data within the carrier files posted to these social media sites remained intact and the encoded files were extracted without issue. Ultimately, we was successful in creating a tool that leverages organizational security policy limitations by using pre-existing social media infrastructure to distribute sensitive information without raising suspicion.

Despite successfully creating a tool that met the intent of the project scope, there are a few limitations to consider with SocialStego in its current configuration. Firstly, like most steganographic solutions, the existence of the original, unencoded, file poses a concern. If original images are not removed from the transporting system, you risk someone recovering the original file. When comparing the contents of an original file to that of an encoded file, it would be a fairly simple conclusion that the file had been encoded. The second major

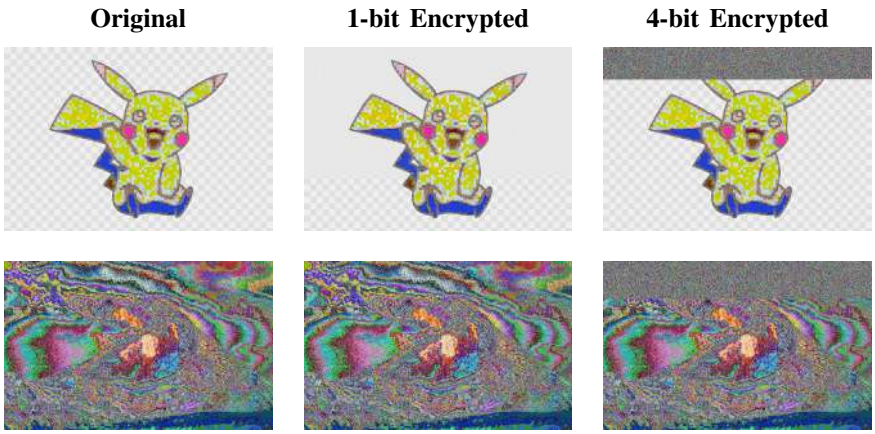

Fig. 5: Comparison of LSB-half bits for original PNG, 1-bit bit-count encrypted, and 4-bit bit-count encrypted images using LSB steganography [38].

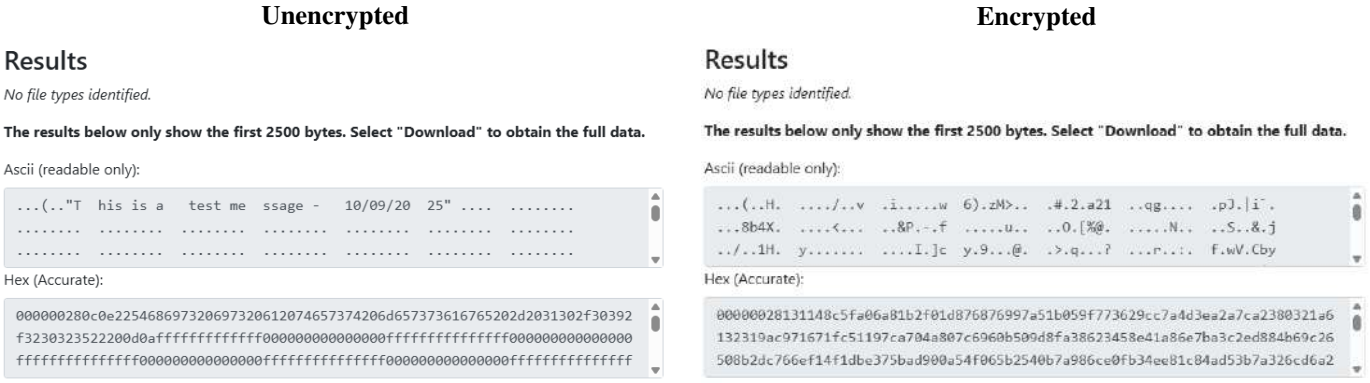

Fig. 6: Comparison of attempted data extraction for a file that was encoded with and without payload encryption using online Steganography tool StegOnline [38]

limitation comes with respect to the assumed distribution of the recipient's public key prior to encrypted payload encoding. In it's current configuration SocialStego is able to implement hybrid encryption due to the assumed existence of a back-channel by which the insider would be able to extract and implement the recipient's public key for encryption. Without an existing back-channel to share public keys between a recipient and an insider, we have not considered a secondary solution that would allow for the encryption of payload data.

As for future work, there are many different routes to pursue in the continuation of this project. However, the two immediate priorities would be to handle the issue visualized in Figure 5 by expanding the encoding process to cover the entire carrier file. At this point, the carrier file is only encoded up to the point at which the final byte of the payload is written - moving forward, if we were to write through the entire carrier file, the noise discrepancies shown in Figure 5 would be less jarring. Currently there is a clear identifiable horizontal line in the pixel representations of the image that hint toward the concept of encoded data, however, continuing to encode random noise into the LSBs of a PNG or WAV will create a more consistent noise representation in the pixel or frame values of their respective files. Secondly, we will continue to dive further into the analysis of the current solution with respect to entropy. Performing an entropy analysis on the encoded files could help to determine if SocialStego, in its current configuration, creates a recognizable file signature in the form of decreased entropy, which may suggest the existence of a covert channel. Lastly, some of the further considerations that would fall under the "nice-to-have" category would be to expand the tool to incorporate additional data types such as JPG for images or FLAC and ALAC file types for video encoding. Additionally, we are interested in designing a user-friendly GUI to aid in the process of tool implementation while also providing a clear basis by which upgrades could be implemented in the future.

Overall, SocialStego fulfills its intent as a proof-of-concept for the feasibility of exporting sensitive information from a target network using Steganography and pre-existing social media infrastructure.

Acknowledgment This work builds upon ideas explored in an earlier, preliminary research effort conducted during an internship at the Griffiss Institute (Rome, NY). The system presented in this paper was independently redesigned, reimplemented, and substantially extended within the scope of the present project. We thank Ms. Cassandra Horwege, Mr. Walter Kirkland Jr., and Mr. Nicholas Tey for their discussions and early exploratory contributions that helped motivate this line of inquiry.

REFERENCES

[1] C. S. R. Center, "steganography - glossary — csrc," csrc.nist.gov. [Online]. Available: https://csrc.nist.gov/glossary/term/steganography

[2] N. Alabdali and S. Alzahrani, "An overview of steganography through history," *International Journal of Scientific Engineering and Science*, vol. 5, p. 41, 2021. [Online]. Available: http://ijses.com/wp-content/uploads/2021/03/117-IJSES-V4N12.pdf

[3] D. Jain, "Lsb image steganography using python," Medium, 04 2021. [Online]. Available: https://medium.com/swlh/lsb-image-steganography-using-python-2bbbee2c69a2

[4] J. T. Force, "Security and privacy controls for information systems and organizations," *Security and Privacy Controls for Information Systems and Organizations*, vol. 5, 09 2020. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r5.pdf

[5] Landwehr, "Cwe-515: Covert storage channel (4.0)," cwe.mitre.org, 07 2006. [Online]. Available: https://cwe.mitre.org/data/definitions/515.html

[6] ——, "Cwe-385: Covert timing channel (4.6)," cwe.mitre.org, 07 2006. [Online]. Available: https://cwe.mitre.org/data/definitions/385.html

[7] Adobe, "Lossless compression: A complete guide — adobe," www.adobe.com. [Online]. Available: https://www.adobe.com/uk/creativecloud/photography/discover/lossless-compression.html

[8] U. of Michigan Library, "Research guides: All about images: Image file formats," Umich.edu, 2019. [Online]. Available: https://guides.lib.umich.edu/c.php?g=282942&p=1885348

[9] P. Winikoff, "What is lossless audio?" Avixa Portal, 2025. [Online]. Available: https://www.avixa.org/pro-av-trends/articles/what-is-lossless-audio

[10] B. Wagner, "Lossless video compression: What is it and why should i care?" Archival Works - Analog to Digital Media Conversion, 12 2019. [Online]. Available: https://www.archivalworks.com/blog/lossless-video-compression

[11] Meta, "Instagram photo posting," Instagram.com, 2025. [Online]. Available: https://help.instagram.com/442418472487929/?cms_platform=www&helpref=platform_switcher

[12] ——, "Using facebook photos," Facebook.com, 2025. [Online]. Available: https://www.facebook.com/help/167931376599294/?helpref=popular_articles

[13] I. Reddit, "Adding images — reddit for developers," Reddit.com, 2025. [Online]. Available: https://developers.reddit.com/docs/capabilities/blocks/app_image_assets#whats-supported

[14] "Adding an image when uploading a track," SoundCloud Help Center, 2023. [Online]. Available: https://help.soundcloud.com/hc/en-us/articles/115003565268-Adding-an-image-when-uploading-a-track

[15] SoundCloud, "Uploading requirements," SoundCloud Help Center. [Online]. Available: https://help.soundcloud.com/hc/en-us/articles/115003452847-Uploading-requirements

[16] X. D. Platform, "Developer platform best practices," X.com, 2025. [Online]. Available: https://docs.x.com/x-api/media/quickstart/best-practices

[17] R. Chandramouli and N. Memon, "Analysis of lsb based image steganography techniques," in *Proceedings 2001 International Conference on Image Processing (Cat. No.01CH37205)*, vol. 3, 2001, pp. 1019–1022 vol.3.

[18] N. Provos and P. Honeyman, "Hide and seek: an introduction to steganography," *IEEE Security & Privacy*, vol. 1, no. 3, pp. 32–44, 2003.

[19] M. Hussain, A. W. A. Wahab, Y. I. B. Idris, A. T. Ho, and K.-H. Jung, "Image steganography in spatial domain: A survey," *Signal Processing: Image Communication*, vol. 65, pp. 46–66, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S092359651830256X

[20] W. Bender, D. Gruhl, N. Morimoto, and A. Lu, "Techniques for data hiding," *IBM Systems Journal*, vol. 35, pp. 313–336, 01 1996.

[21] C.-C. Chang, C.-C. Lin, C.-S. Tseng, and W.-L. Tai, "Reversible hiding in dct-based compressed images," *Information Sciences*, vol. 177, no. 13, pp. 2768–2786, 2007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025507001016

[22] M. Tierney, I. Spiro, C. Bregler, and L. Subramanian, "Cryptagram: photo privacy for online social media," in *Proceedings of the First ACM Conference on Online Social Networks*, ser. COSN '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 75–88. [Online]. Available: https://doi.org/10.1145/2512938.2512939

[23] J. Ning, I. Singh, H. V. Madhyastha, S. V. Krishnamurthy, G. Cao, and P. Mohapatra, "Secret message sharing using online social media," in *2014 IEEE Conference on Communications and Network Security*, 2014, pp. 319–327.

[24] V. Holub and J. Fridrich, "Designing steganographic distortion using directional filters," in *WIFS 2012 - Proceedings of the 2012 IEEE International Workshop on Information Forensics and Security*, 12 2012.

[25] ——, "Digital image steganography using universal distortion," in *IH and MMSec 2013 - Proceedings of the 2013 ACM Information Hiding and Multimedia Security Workshop*, 06 2013.

[26] T. Pevný, T. Filler, and P. Bas, "Using high-dimensional image models to perform highly undetectable steganography," in *Information Hiding*, R. Böhme, P. W. L. Fong, and R. Safavi-Naini, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 161–177.

[27] X. D. Platform, "Introduction - x," X.com, 2025. [Online]. Available: https://docs.x.com/x-api/posts/manage-tweets/introduction

[28] libpng, "Png basics (png: The definitive guide)," Libpng.org, 2026. [Online]. Available: https://www.libpng.org/pub/png/book/chapter08.html

[29] Pillow, "Image module — pillow (pil fork) 6.2.1 documentation," Readthedocs.io, 2011. [Online]. Available: https://pillow.readthedocs.io/en/stable/reference/Image.html

[30] P. S. Foundation, "tkinter — python interface to tcl/tk — python 3.7.2 documentation," python.org, 2019. [Online]. Available: https://docs.python.org/3/library/tkinter.html

[31] ——, "binascii — convert between binary and ascii," Python documentation, 2025. [Online]. Available: https://docs.python.org/3.11/library/binascii.html#binascii.crc_hqx

[32] ——, "os — miscellaneous operating system interfaces," Python documentation, 2025. [Online]. Available: https://docs.python.org/3.11/library/os.html#os.urandom

[33] "Primitives — cryptography 46.0.3 documentation," Cryptography.io, 2025. [Online]. Available: https://cryptography.io/en/46.0.3/hazmat/primitives/

[34] CISA, "Transition to advanced encryption standard (aes)," 2024. [Online]. Available: https://www.cisa.gov/sites/default/files/2024-05/23_0918_fpic_AES-Transition-WhitePaper_Final_508C_24_0513.pdf

[35] J. Ning, I. Singh, H. V. Madhyastha, S. V. Krishnamurthy, G. Cao, and P. Mohapatra, "Secret message sharing using online social media," *2014 IEEE Conference on Communications and Network Security*, 10 2014.

[36] S. Wickramasinghe, "Rsa algorithm in cryptography: Rivest shamir adleman explained," Splunk-Blogs, 11 2024. [Online]. Available: https://www.splunk.com/en_us/blog/learn/rsa-algorithm-cryptography.html

[37] R. , "Api reference," Readthedocs.io, 2025. [Online]. Available: https://discordpy.readthedocs.io/en/v2.6.4/api.html

[38] GeOrg3, "Stegonline," GitHub, 05 2022. [Online]. Available: https://github.com/Ge0rg3/StegOnline/blob/master/README.md