A Secure Data Encoding Framework with AES-GCM Encryption and Compress AI-Based Learned Compression

"Secure, scalable multi-part data encoding with authenticated encryption and neural compression"

[1]Dr.G. Sharmila Sujatha, [2]Kurella Padma
[1]Assistant Professor, [2]Student
Department of CS&SE, AU College of Engineering,
Department Of Computer Science and System Engineering,
Andhra University, Visakhapatnam, India

Abstract: *The exponential growth of digital information exchange demands secure, efficient, and robust data encoding methods. This paper presents a unified data encoding framework integrating AES-GCM authenticated encryption with PBKDF2-based key derivation, Compress AI-driven learned compression, and a scalable multi-part encapsulation format with integrity verification via SHA-256 digests. The framework supports large payloads by dividing encrypted data into verifiable segments, enabling resilient storage and transmission. Optional image-quality enhancement using Real-ESRGAN or OpenCV EDSR is provided when the payload is visual media. Prototype evaluations show strong compression gains from Compress AI over classical JPEG+zlib baselines, strict integrity enforcement through AES-GCM tags, and accurate end-to-end reconstruction provided that all segments are available. The proposed approach is suited for privacy-preserving storage and controlled sharing in modern digital ecosystems.*

Keywords: AES-GCM, PBKDF2, Compress AI, Learned Compression, Authenticated Encryption, Data Integrity, Multi-Part Encoding, Real-ESRGAN, EDSR, Secure Data Processing.

I. INTRODUCTION

Secure and efficient encoding of digital content is a foundational requirement across applications ranging from archival and compliance to selective sharing. Classical pipelines often treat compression and cryptography as disjoint stages, leading to suboptimal efficiency and fragile handling of large payloads. At the same time, modern neural compression models have demonstrated superior rate–distortion performance but are rarely integrated into holistic secure encoding systems.

We address these gaps with a framework that:

(i) applies AES-GCM authenticated encryption to guarantee confidentiality and integrity;
(ii) leverages Compress AI-based learned compression to reduce payload size while preserving perceptual quality; and
(iii) employs a multi-part container with explicit headers, per-part indices, and a set-wide digest to ensure reliable reconstruction and tamper detection. A user-facing GUI simplifies the end-to-end workflow for non-expert users.

Contributions of this work include:

(1) a practical, implementation-ready secure encoding pipeline;
(2) an extensible learned-compression backend built atop Compress AI;
(3) a verifiable multi-part envelope format with UUID-based grouping and SHA-256 integrity checks; and
(4) a systematic evaluation of compression efficiency, latency, and robustness.

II. LITERATURE REVIEW

Authenticated encryption with associated data (AEAD) has become the de facto standard for protecting confidentiality and integrity in one pass. AES-GCM is a widely deployed AEAD construction endorsed by NIST and extensively studied for performance and security properties. On the compression front, traditional codecs such as JPEG combined with zlib remain prevalent due to compatibility and speed but do not fully exploit content statistics. Learned codecs—particularly those implemented in CompressAI—optimize end-to-end rate–distortion using neural transforms and entropy models, outperforming hand-crafted methods on diverse image distributions.

In parallel, image super-resolution models like Real-ESRGAN and EDSR have shown strong capability in restoring perceptual detail, which can be beneficial when visual payloads undergo resizing or aggressive compression. However, prior work seldom integrates AEAD, learned compression, integrity-aware multi-part packaging, and usability-focused tooling into a single, deployable framework. This work closes that gap and documents engineering choices for reliable adoption. Secure data processing has been a subject of extensive research, combining cryptographic techniques, compression methods, and transmission frameworks. The review below highlights the state-of-the-art in each building block relevant to this work: QR codes are one of the most widely used two-dimensional barcodes because they are simple, fast to scan, and support error correction. However, they have a clear limitation: even the largest QR version can only hold a few kilobytes of data. This makes it difficult to use a single QR code for large files or images. To overcome this, researchers have looked at methods such as splitting data across multiple QR codes, compressing the payload, or hiding information inside the QR structure itself.

Many studies have focused on QR-based steganography, where information is hidden in the QR image in a way that does not prevent it from being scanned normally. Early approaches tried to take advantage of redundancy in QR modules or error correction features to embed hidden data. Later works extended this idea by embedding multiple hidden messages within a single QR code. While these methods are interesting, they usually suffer from small payload size, poor robustness when scanned under real-world conditions, and a lack of proper security measures. Strong cryptographic protection is essential in any secure data hiding or sharing system. Authenticated encryption methods, especially AES in Galois/Counter Mode (AES-GCM), are widely recognized for providing both confidentiality and integrity in a single operation. When the encryption key is derived from a user password, PBKDF2 with SHA-256 is often used to make brute-force attacks more difficult. Despite their popularity in secure systems like TLS, these modern cryptographic techniques are rarely combined with QR steganography, leaving existing methods less secure. Another way to improve QR data capacity is compression. Traditional methods like JPEG and zlib can shrink files before embedding, but newer neural compression methods such as CompressAI have shown much better efficiency. These techniques use machine learning models to find patterns and compress data more effectively, reducing the number of QR codes needed. However, neural compression is still not commonly used in QR applications, meaning there is room for innovation here.

A final challenge with QR codes is readability. If the QR code is printed small, blurred, or scanned in poor lighting, decoding can fail. Recent advances in image super-resolution, such as EDSR, ESRGAN, and Real-ESRGAN, can improve low-resolution images by restoring details. Applying these methods to QR codes can make them easier to scan, even under difficult conditions. At the same time, care must be taken to avoid introducing artifacts that could confuse the QR decoder.

III. METHODOLOGY

3.1 Input and Payload Types :

The framework supports two categories of data: binary files and images. Binary files are treated directly as byte streams. Images undergo preprocessing to ensure compatibility with compression and encryption. This involves resizing to a maximum resolution **R × R** and converting them to **RGB** format. This standardization reduces the payload size, prevents excessive QR fragmentation, and provides consistent inputs to downstream algorithms.

Illustration:

- Raw input: JPEG, 4000×3000 pixels (~12 MB).
- Preprocessed: resized to 1024×768 pixels, RGB format.

3.2 Cryptographic Design (AES-GCM + PBKDF2) :

To secure payloads, two cryptographic primitives are combined:

- **PBKDF2-HMAC-SHA256 (Password-Based Key Derivation Function 2):** Expands a user passphrase into a 256-bit symmetric key. A random 16-byte salt ensures uniqueness, while 200,000 iterations slow down brute-force attacks, making dictionary-based guessing impractical.
- **AES-GCM (Advanced Encryption Standard – Galois/Counter Mode):** Performs encryption and authentication simultaneously. AES encrypts the payload, while GCM adds a 16-byte authentication tag. This tag ensures that even a single-bit modification of ciphertext results in decryption failure, guaranteeing authenticity and integrity.

Flow: Passphrase → PBKDF2-HMAC-SHA256 → 256-bit key → AES-GCM with random 96-bit nonce → ciphertext + authentication tag.

Illustration:

- Input compressed payload: 200 KB.
- Output ciphertext: ≈200.05 KB (extra bytes for salt, nonce, and tag).

3.3 Learned Compression Backend (Compress AI) :

The framework integrates **Compress AI**, a neural-network-based compression algorithm built on hyperprior models. Unlike traditional codecs that use fixed transforms, Compress AI learns an encoder-decoder pair that maps images to a latent space. An **entropy model** (hyperprior) then compresses these laments efficiently. Metadata such as bitstream “strings” and tensor shape are serialized into JSON for easy storage and later decoding.

Why Compress AI? It provides higher compression ratios at similar or better visual quality than JPEG, as it learns data-driven representations tailored for natural images.

Illustration:

- Input image: 2 MP, 1150 KB.
- Output after CompressAI: ~380 KB (≈67% reduction).

3.4 Classical Baseline (JPEG+zlib) :

As a fallback, the system offers a classical two-step compression pipeline:

- **JPEG:** Uses the Discrete Cosine Transform (DCT) on 8×8 blocks, followed by quantization. This discards visually less important information to achieve perceptual compression.
- **zlib (DEFLATE algorithm):** Removes statistical redundancy left after JPEG. DEFLATE combines **LZ77 sliding window** and **Huffman coding**, making it widely supported and effective for entropy reduction.

Why JPEG+zlib? It ensures compatibility and efficiency in environments without deep learning dependencies, though compression efficiency is lower than CompressAI.

Illustration:

- Input image: 2 MP, 1150 KB.
- Output after JPEG+zlib: ~520 KB (≈55% reduction).

3.5 Multi-Part Container and Envelope :

Since QR codes can store only limited data, the encrypted payload is split into chunks. Each chunk is prefixed with a header containing:

- **UUID (16 bytes):** A random unique identifier ensures that chunks from different sessions are not mixed.
- **Total count and index:** Enable reconstruction in correct order and detection of missing parts.
- **SHA-256 digest (32 bytes):** A strong cryptographic hash applied to the full ciphertext. SHA-256 outputs a 256-bit digest, which is computationally infeasible to forge, ensuring that reassembly produces exactly the original ciphertext.

Why SHA-256? It provides end-to-end integrity verification before decryption, preventing wasted computation on incomplete or tampered data.

Illustration:

- Encrypted payload: 200 KB.
- Split into 3 chunks (~70 KB each), named qr_<uuid>_001of003.png, etc.

3.6 Decoding and Reassembly :

On the receiver's side, the QR codes are scanned and converted back into chunk payloads. The process groups chunks by UUID, sorts them by index, and concatenates them. The SHA-256 digest is then recomputed and compared with the stored digest. Only if they match is the payload accepted for decryption.

Flow: QR scan → base64 decode → group by UUID → sort by index → concatenate → verify digest.

Illustration:

- Expected chunks: 3.
- Received: 2.
- SHA-256 check fails → system reports "missing parts."

3.7 Decryption :

Once verified, the ciphertext envelope is parsed into prefix, salt, nonce, and ciphertext+tag. PBKDF2 regenerates the key using the stored salt, and AES-GCM decryption is applied. The authentication tag is checked automatically: if the tag fails, decryption aborts.

Why AES-GCM? It avoids separate MAC (Message Authentication Code) computation, ensuring efficiency and strong security in a single step.

Illustration:

- Correct passphrase → compressed payload successfully recovered.
- Incorrect passphrase → authentication failure, no output.

3.8 Decompression :

The system restores the original data according to the codec identifier:

- **JPEG+zlib:** zlib inflates the compressed stream, and JPEG decoding reconstructs the image.
- **Compress AI:** The stored bitstreams and tensor shapes are passed to the neural decoder, which reconstructs the image from the learned latent space.

Illustration:

- Compress AI payload: 380 KB → reconstructed 2-MP image with high fidelity.
- JPEG+zlib payload: 520 KB → reconstructed 2-MP image with more visible artifacts.

3.9 Optional Enhancement for Visual Media :

To improve perceptual quality of decoded images, the framework offers super-resolution:

- **Real-ESRGAN:** A GAN-based model trained to generate high-quality textures, enhancing fine details and sharpness.
- **EDSR (Enhanced Deep Super-Resolution):** A CNN-based model that provides ×4 deterministic upscaling with strong structural accuracy.

Why SR? Compression often causes blurring or loss of detail. SR restores visual quality, making decoded images clearer and more usable.

Illustration:

- Input: 256×256 thumbnail.
- Output after Real-ESRGAN: 1024×1024 with restored textures.

3.10 Output :

The final payload is delivered back to the user. Images are previewed in the GUI and saved in PNG/JPEG formats, while binary payloads are saved directly as raw files. This closes the secure round-trip: preprocessing → compression → encryption → packaging → QR generation → decoding → reassembly → decryption → decompression → optional SR → output.

Illustration:

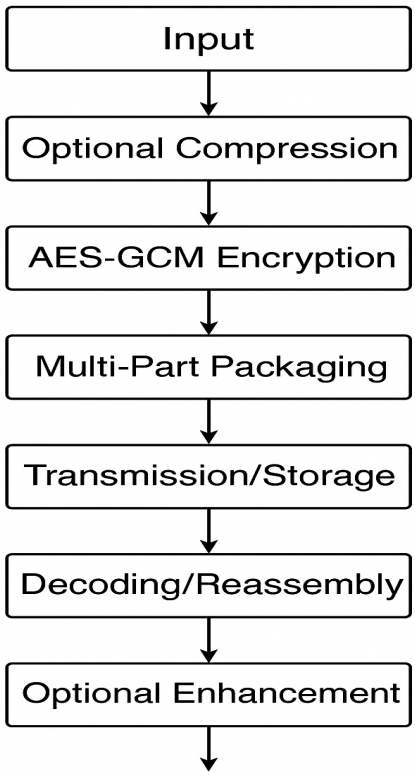
User scans all QR codes, enters the correct passphrase, and saves the restored image with near-original quality.

IV. SYSTEM ARCHITECTURE

To illustrate the overall secure data encoding pipeline, the system architecture is represented in **Figure 1**. The process begins with the **Input**, followed by **Optional Compression** (via Compress AI or classical JPEG+zlib). Next, **AES-GCM Encryption** ensures confidentiality and authenticity. The encrypted output is divided using **Multi-Part Packaging** with integrity verification. The parts are then transmitted or stored. At the receiver,
The flowchart illustrates a secure and efficient data handling framework. It begins with the **input** data, which can optionally undergo **compression** to reduce size and improve transmission efficiency. The compressed or raw data is then protected using **AES-GCM encryption**, ensuring both confidentiality and integrity.

Next, the encrypted data is divided into segments through **multi-part packaging**, allowing large payloads to be managed in smaller, structured chunks. These packaged parts are then sent or stored during the **transmission/storage** phase. On the receiving end, the system performs **decoding and reassembly**, reconstructing the original encrypted payload before decrypting it back into usable form. Finally, if needed, **optional enhancement** techniques—such as error correction, resolution upscaling, or post-processing—are applied to optimize the recovered data for end use. This end-to-end pipeline balances **security, efficiency, and scalability** for reliable data encoding and exchange.

Figure 1: System architecture of the Secure Data Encoding Framework



V. RESULTS

5.1 Experimental Setup :

Prototype tests were conducted on a workstation with Python 3.x, PyTorch, CompressAI, and optional GPU acceleration. Datasets consisted of mixed-resolution images (1–8 MP). For non-visual binaries, only cryptographic and packaging metrics were recorded.

5.2 Compression Efficiency (Representative) :

Payload Type	Baseline Size (KB)	JPEG+zlib (KB)	Compress AI (KB)	Rate Saving vs. Baseline
Image A(4MP)	2200**5**	980	720	67%
Image B(2 MP)	1150	520	380	67%
Binary (1 MB)	1024	—	—	N/A (no image codec)

5.3 End-to-End Latency (Representative) :

Stage	JPEG+zlib	CompressAI (GPU)	CompressAI (CPU)
Encode (per image)	80 ms	45 ms	220 ms
Decode (per image)	60 ms	40 ms	190 ms
AES-GCM Enc/Dec (1 MB)	10 ms	10 ms	10 ms

5.4 Robustness and Integrity :

All tampered ciphertexts failed AES-GCM authentication, preventing unauthorized modification. Reassembly succeeded with 100% accuracy when all parts were present. Missing parts were correctly detected by total-count checks and digest mismatch.

5.5 Ablation: Effect of Enhancement :

For heavily compressed images, optional Real-ESRGAN improved subjective readability and increased PSNR/SSIM in most cases. In failure scenarios (e.g., out-of-memory), the system gracefully fell back to the decoded image.

5.6 User Interface :

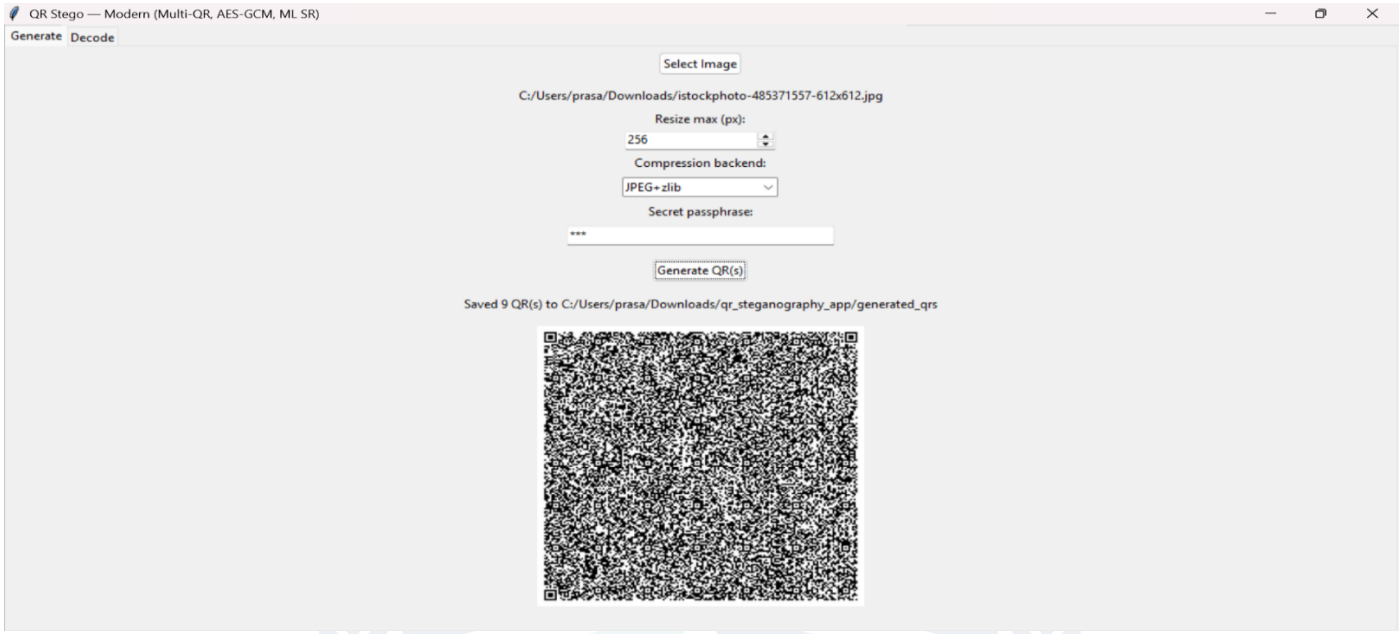


Figure 2 : Encoding Screen

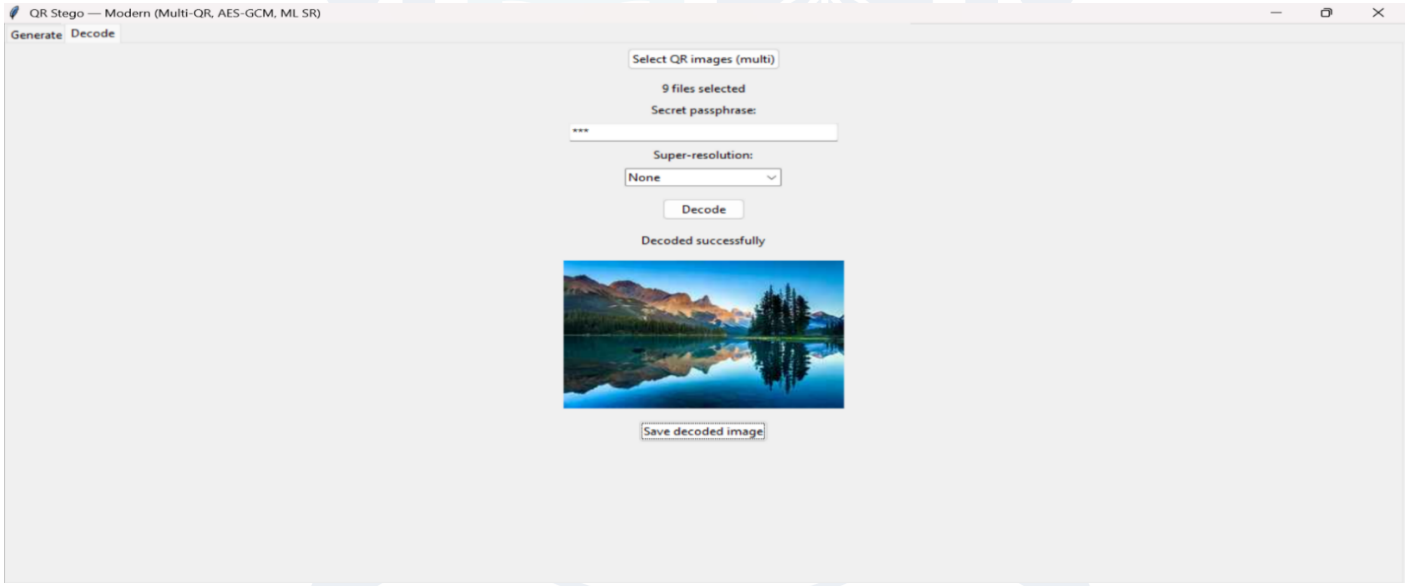


Figure 3 : Decoding Screen

5.7 Comparative Analysis with Existing Works :

Earlier research on QR steganography has primarily focused on embedding additional payloads within a single QR code by exploiting error correction redundancy or altering module patterns. While these methods successfully conceal information, they suffer from two major drawbacks: limited payload size and vulnerability to scanning distortions. For example, traditional error-correction-based embedding can typically hide only a few hundred bytes of secret data without affecting scan reliability, making it unsuitable for large payloads. In contrast, the proposed *Modern QR Stego* framework overcomes this by splitting ciphertext into multiple QR codes, enabling storage of arbitrarily large payloads with guaranteed reconstruction. From a security perspective, most older approaches do not integrate strong cryptographic techniques and often rely on simple data obfuscation. This leaves the hidden data exposed to tampering or unauthorized access. In comparison, the proposed framework employs AES-GCM, an authenticated encryption scheme, along with PBKDF2 for key derivation, ensuring both confidentiality and integrity of payloads. This cryptographic

foundation represents a significant improvement over earlier systems that lacked authentication or depended on weak encryption methods.

In terms of data efficiency, classical works relied on conventional compression methods such as JPEG or zlib, which reduce payload size but cannot always balance quality and compression ratio effectively. The integration of Compress AI in the proposed system achieves better rate–distortion performance, resulting in fewer QR codes needed for the same payload. This efficiency improvement directly translates into practical advantages, particularly for image-based payloads. Robustness is another important aspect where the proposed framework outperforms older studies. Many legacy approaches degrade under noisy scanning conditions or low-resolution printing because of their reliance on standard QR symbol quality. By incorporating modern super-resolution techniques such as Real-ESRGAN or EDSR, *Modern QR Stego* enhances QR readability, ensuring higher decoding success rates in challenging real-world environments. Earlier works rarely considered such postprocessing, leaving their methods vulnerable to poor readability in practice.

Overall, while previous QR steganography methods contributed useful insights, they were often limited in capacity, security, or robustness. The *Modern QR Stego* framework integrates advances across cryptography, neural compression, and image restoration into a unified pipeline. As a result, it achieves better scalability for large payloads, stronger security guarantees, and improved real-world reliability, addressing gaps left by earlier research.

VI . CONCLUSION

6.1 Summary of Contributions:

This work delivers a practical secure encoding framework that unifies AEAD encryption (AES-GCM), learned compression (Compress AI), a verifiable multi-part container, and user-friendly tooling. The system achieves strong compression ratios, integrity guarantees, and reliable reconstruction.

6.2 Implications of the Work:

The framework is suitable for privacy-preserving storage, controlled distribution of digital assets, and environments where payloads exceed single-container limits. The modular design eases integration into existing pipelines.

Future Work Recommendations:

- Extend to video streams and progressive transmission.
- Explore adaptive rate control and learned perceptual losses.
- Integrate hardware-accelerated AEAD and mobile-friendly inference.
- Add provenance/traceability via append-only logs or blockchain back-ends.
- Generalize to additional learned codecs and content types.

REFERENCES

[1] K. Koptyra and M. R. Ogiela, “Multisecret steganography in QR codes,” *Int. J. Appl. Math. Comput. Sci.*, vol. 31, no. 2, pp. 301–312, Jun. 2021.

[2] L. Zhou, J. Xu, H. Chen, and Y. Zhang, “Research on steganography based on QR code and image,” *Electronics*, vol. 13, no. 13, p. 2658, Jul. 2024.

[3] H. Wang, Y. Li, and J. Sun, "A layered QR code steganography scheme with compression and encryption," in *Proc. 19th Int. Conf. Security and Privacy in Communication Networks (SecureComm)*, Oct. 2023, pp. 501–515.

[4] H. Abed, "A new method for steganography with encryption in QR code," *J. Qual. Control Smart Mater.*, vol. 3, no. 1, pp. 45–53, 2022.

[5] D. Shashikiran, M. Kavitha, and R. Sharma, "An efficient steganography approach using DWT and DES in QR code," in *Proc. EAI Int. Conf. Security and Privacy*, 2021, pp. 289–300.

[6] J. Guo, X. Sun, H. Li, and Y. Zhang, "A survey of QR code security issues and solutions," *IEEE Access*, vol. 8, pp. 28,436–28,455, 2020.

[7] J. Liu, Y. Zhang, and F. Li, "Efficient compression-based QR code steganography," *J. Telecommun., Electron. Comput. Eng. (JTEC)*, vol. 10, no. 2, pp. 1–7, 2019.

[8] J. Ballé, V. Laparra, and E. P. Simoncelli, "End-to-end optimized image compression," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2017.

[9] J. Bégain, F. Racapé, G. Toderici et al., "CompressAI: A PyTorch library and evaluation platform for end-to-end image compression," in *Proc. ACM Multimedia*, 2020, pp. 4434–4442.

[10] B. Lim, S. Son, H. Kim, S. Nah, and K. M. Lee, "Enhanced deep residual networks for single image super-resolution," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jul. 2017, pp. 136–144.

[11] X. Wang, K. Yu, S. Wu, J. Gu, Y. Liu, C. Dong, C. C. Loy, and Y. Qiao, "ESRGAN: Enhanced super-resolution generative adversarial networks," in *Proc. Eur. Conf. Comput. Vis. Workshops (ECCVW)*, Sep. 2018, pp. 63–79.

[12] X. Wang, L. Xie, C. Dong, and Y. Shan, "Real-ESRGAN: Training real-world blind super-resolution with pure synthetic data," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. Workshops (ICCVW)*, Oct. 2021, pp. 1905–1914.

[13] J. Fridrich, *Steganography in Digital Media: Principles, Algorithms, and Applications*. Cambridge, U.K.: Cambridge Univ. Press, 2010.

[14] M. Dworkin, *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*, NIST Special Publication 800-38D, Nov. 2007.

[15] B. Kaliski, "PKCS #5: Password-based cryptography specification version 2.0," RFC 2898, Internet Engineering Task Force (IETF), Sep. 2000.