PHANTOM VAULT - UNICODE ZERO-WIDTH CONCEALMENT

Kalaiselvi B
Associate Professor/ Computer Science And Engineering
Mahendra Engineering College, Namakkal, Tamil Nadu, India
kalairs2003@gmail.com

Abinaya S
UG Student / Cyber Security
Mahendra Engineering College, Namakkal, Tamil Nadu, India
abinayasekar1126@gmail.com

Ajay Kumar R
UG Student / Cyber Security
Mahendra Engineering College, Namakkal, Tamil Nadu, India
ajaykumar110605@gmail.com

Eraiamudhan D
UG Student / Cyber Security
Mahendra Engineering College, Namakkal, Tamil Nadu, India
dhanasekarani1967@gmail.com

Hariharan R
UG Student / Cyber Security
Mahendra Engineering College, Namakkal, Tamil Nadu, India
hariharan2005rh@gmail.com

Hariharan R K
UG Student / Cyber Security
Mahendra Engineering College, Namakkal, Tamil Nadu, India
rkhari3535@gmail.com

Abstract - The traditional cryptography techniques are vulnerable to the visibility issue, whereby the encryption algorithm produces high-entropy ciphertext, easily observed by the network scanners and tools. The significant amount of research has been devoted to the data protection based on extant encryption schemes, however, the problem of detecting them is not addressed by current means. To fill this gap, the current paper presents the Phantom Text Vault, a steganographic design that encrypts the data using the encrypted form and inserts it into the normal text using the zero-width Unicode characters, namely U+200B and U+200C. The system provides encryption with the use of AES-256-GCM and ChaCha20-Poly1305. Key derivation is implemented with the Scrypt function as part of an HMAC-SHA256 based proof-of-work module which aims at 300 milliseconds of latency. The carrier binding functionality identifies the decryption key with the hash of the cover text via the SHA-256, any change in the cover will destroy the key. The experimental outcomes prove an increase in the average file size by 24 percent, and entropy measures are consistent with the ones in natural language text. But applications that utilize Unicode normalization like Slack and Microsoft word mitigated the concealed data through communication whereas text channels that did not have formats retained it in its entirety. Therefore, the proposed system can offer security researchers with a viable and low-profile channel through which sensitive information can be relayed without automated detection tools.

Keywords - Unicode steganography, zero-width characters, AES-256-GCM, ChaCha20-Poly1305, Scrypt key derivation, carrier binding, covert channel, authenticated encryption.

I. INTRODUCTION

In the modern digital ecosystem, data security is one of the most important issues. Secrecy of sensitive information is a great challenge to both security researchers and organisations. It is often inadequate to make a file mathematically unreadable. Preferably, the opponents would not even know the existence of the file. There have been many investigations on data hiding and data encryption through available techniques, but the problem of data visibility has been overlooked with the conventional cryptographic principles. Traditional cryptographic algorithms are efficient in protecting the content of the data and they create an unveiled imprint of the scrambled ciphertext. The existence of such block of ostensibly arbitrary information indicates that there is some safeguarded information available and thus interests network scanners and passive opponents. The common cryptographic products are defined by high-entropy patterns that can easily be differentiated between a typical text. Encrypted content can easily be detected by automated monitoring tools and the deep packet inspection systems. Although encryption ensures the contents of the data are secure, it does not hide the fact that the data exists, which is a drawback that continues to exist in current cryptographic methods. Steganography helps to overcome this weakness by hiding the information in the normal carrier data.

Instead of encrypting the data by overt means, steganographic methods conceal the hidden message inside regular texts with non-rendering characters. Someone who chanced upon the scroll of writing sees that there is nothing out of the ordinary; the secret is unobtrusively concealed and can be fully restored by the addressee. However, steganography is not a complete assurance of data security of the hidden message. In case of discovery of the hidden content, it can be read with little effort. Therefore, steganography and encryption should be used together to get a better security performance. Steganography in texts is done by the use of characters that are invisible and that is, Zero-宽度空间 (U + 200B) and Zero-宽度 non-joiner (U + 200C). Such characters will not alter the visual display but will not affect the typographical space or display of a document. A human being who is visually examining the document would not see anything wrong but, the byte sequences behind them would be maintained and could be handled at the level of the parser through computational methods. No abnormalities would be observed by a human observer who is doing visual inspection.

The derivation of key is done through Scrypt algorithm combined with a chained HMACSHA256-proof-of-work algorithm. The Scrypt algorithm is dynamically scaled to the maximum available memory of the host machine and the proof-of-work element is added to create about 300 milliseconds of purposeful slowdown of computation at each attempt. This latency adds unnecessary cost to brute force attacks but does not add any negative impact in the normal operation of legitimate users. It has a carrier binding feature that ensures the integrity of data. The decryption key is bound directly to the SHA 256 hash of the visible text on the

cover. In case the cover text is modified in any way, including even slightly, the validation of the hash would fail, and the derivation of keys would be impossible, which would guarantee the integrity of the concealed information and prevent the manipulation of the cover text, that is, to deceive the receiver. Also, the HMAC tag of the header is checked before performing any decryption operation; its failure raises a ValueError, an indicator of tampering of the header. The proposed system also has temporary access controls.

The architecture allows placing time constraints, i.e. not before (nbf) and expiration (exp) timestamps that are automatically calculated against the current UTC system clock. When the current time is not within the allowable range, a PermissionError is thrown, and then the hidden data is not allowed. This feature can be applied to the control of data availability in testing and under controlled shared environments. As the experimental analysis shows, in the proposed system the average file-size overhead is 24 ○-○ in comparison to plaintext in its raw form. Another high level of concealment is proved by visual and statistical observation of carrier files. Ordinary text editors have no visible artefacts, formatting errors or placeholder characters. Analysis of the entropy of Unicode confirms that steganographic files have an entropy that is equal to that of natural language text and automatic heuristic scanners cannot tell the difference between carrier files and ordinary text. However, platforms that overly popularize Unicode characters including Slack and Microsoft Word, corrupt the hidden payload on transmission. Unformatted plaintext channels, in their turn, do not alter the information.

II WORKING PROCESS

2.1 Initialization and User Interface Activation

Execution of the chassis When it is executed, the system triggers a special graphical user interface specific to operating within windows. The interface uses the simplified, event based window, which guides operators to the entire steganographic process. In its original release, the binary initiates an asynchronous bootstrap mechanism, with the frontend displaying the startup status in real time to ensure user orientation during the start of the program. The backend offers the necessary cryptographic primitives and at the same time implements tight memory enclave isolation. The host display scaling parameters are used to derive the window geometry dynamically used by the application. The primary thread is part of the operating system event loop directly, and this ensures that there is no latency during the capture of keyboard input during operation.

2.2 Input Acquisition and Validation

Once the mode is selected, the system enters the data acquisition phase. The operator is called for input. A cover message is required. The cover message is the visible text that works as the carrier to the hidden data. This step is optional if the user This is only meant to archive files. But, the cover message is critical for steganographic transport. After that, the user can choose up to ten distinct files for encryption. The system performs an immediate validation check on each file path. ensure the data is available as well as readable before the encryption pipeline is started. But, the encryption pipeline is not started until all file paths are validated.

The input phase supports multiple encryption options of the input phase. The user is given a choice of encryption methods. They are AES-256-GCM as well as ChaCha20-Poly1305. prefers AES-GCM when hardware acceleration is found on the host processor. But, ChaCha20 is presented as a better option for systems where AES performance may be suboptimal. Both primitives provide authenticated encryption as well as data integrity verification. The UI masks the passphrase buffer in real-time. This is to prevent shoulder-surfing as well as screen-capture exploits. The backend logic stop the process on null or zero-length password entries. Empty credentials are rejected before the encryption pipeline is engaged. When an operator opts for the keyfile route, the engine does not dump the raw bytes. Instead, the cryptographic material is serialised into a hardened JSON-structured local vault for safe local storage.

2.3 Cryptographic Key Derivation Pipeline:

Memory- Hardened Derivation: to dampen the threat Based on GPU brute- force attacks, The system works the Scrypt algorithm. This algorithm is context aware. It calibrates its difficulty parameters(Especially the" N" factor) Based on the available memory of the host device, Default setting a 64MB limit The derivation process Uses specific setting parameters incl a block size(r) 8 and a parallelization factor(p) 1 of- to produce the base secret.

Proof- of- Work(PoW) Hardening: to follow the initial derivation, Introducing the system. A deliberate computational latency. It is achieved through a" Proof-of- Work" mechanism Those chains iterations of HMAC-SHA256. The system adjusts itself strenuously. The number of rounds To ensure this the key generation process eats a target time of approx 300 milliseconds. This delay is inexplicable. A legitimate user But gives large- scale dictionary Computationally forbidden attacks.

The key Unification through HKDF: The final master key is derived based on the HMAC. Key Derivation Function(HKDF) to deploy the SHA- 256 hash algorithm. This step takes the base secret, Production of PoW mechanism, and a hash Of the visible cover By adding a message the hash Of the visible text, The system binds the encryption key To the carrier The message itself, produce definite that any modification To the cover text presents the key invalid.

2.4 Decoding Workflow and Structural Verification

Before that final encapsulation, Explains. The user go access policies To the hidden data. The system allows. The establishment of time- Restrictions on grounds, in particular" no Before"(nbf) and" Expiration"(exp) timestamps. Calculations are done automatically on a manuscript- by- publication basis. These values. The system's current clock and trust it. The standard UTC format To Ensure consistency throughout different time zones.While these checks It is advisory and dependent. The system clock, They furnish the necessary control layer for

management. Data availability In testing or shared environments.

The encryption engine The process each selected file independently The system Iterate through the file list, reading the plaintext bytes and create a fresh, unique 12-byte nonce to every file. The data is then encrypted. The selected AEAD primitive(AES- GCM or ChaCha20), which produces both. The ciphertext and an authentication tag.

The resulting encrypted blobs are collected into a structure. JSON container. It holds the container. An array of file records, A description of each the filename, File size, base64- encoded nonce, and the ciphertext. Global metadata incl the version number(" QHIDE1"), Time politics, and the chosen algorithm, There are nests inside this JSON structure.This design allows the system to encapsulate multiple files within a single transport stream, Separate from it simpler tools It only supports singular freights.

2.5 Decoding Workflow and Structural Verification

Decode secure freight() parses container JSON, Repeating records: base64- decode nonce/ ciphertext, decryptaead() returns the plaintext using matching. AEAD primitive. Policy validation occurs before the loop. Record the recovered files. Path(outdir)/ name with write_bytes(data). Comprehensive reporting Of the reportdecode() directory input TXT, visible message, restored count/ sizes, The mirror encoding summaries for auditability.

2.7 Error Handling and Robustness Features

Runtime exceptions Level description: valueerror(" Hidden freighttruncated"), oserror(" cryptprotectdatafailed"), permissionerror(" Expired According to the policy"). GUI traps With attempt/ except messagebox. Showerror(), reliable state. freight misalignment, Key similarity, or formalisation damage trigger early failure, prevention partial disclosures. Skip cross- platform options. DPAPI, To standard password wrapping Globally.

2.8 Performance Characteristics

The system allows robust error handling throughout the lifecycle of the data. Exceptions to driving time– for demonstration truncated freight, missing keys or corruption due To text formalisation are Captured and presented. The user through descriptive error Messages it stops. Partial disclosures And provides for the user It is famous for its reason a recovery attempt failed From a performance perspective, The system is designed to balance security with ease of use.

The cryptographic delays according to the 300-millisecond benchmark, Secure this time the system is responsible for legitimate users, It remains against the automatic. Attack scripts. In addition, the system is capable of handling large datasets By processing files in chunks, essentially limited the bounds of the chinwag platform is used to transfer instead of the tool itself.

The use Of zero- width characters In conclusion a file size increase is more efficient than traditional Base64 Coding, allows substantial data storage without a suspicious increase in file volume. Comprehensive reporting feature catalog the input text, Visible message, and the restored file sizes, to provide an audit trail to the decryption process.

III RESULT AND DISCUSSION

3.1 *Performance Metrics And Experimental Setup*

A comprehensive collection and implementation of numerous tests of the overall performance of the new architecture was executed by implementing an extensive and varied range of host testing based upon the operating systems of various hosts. The testing pipeline was designed so that it closely mirrored the extremes associated with a real-world deployment of the architecture.

We collected and located 50 different groups of heterogeneous ingest data. The ingest data were comprised of a very wide variety of compiled executable binaries, high-density lossless images, and normally, (p) Portable Document Format (PDF) documents, accumulating a total of approximately 2.3 gigs of volume. The telemetry collected and recorded during both the ingest and extract processes allowed us to track the exact amount of latency associated with the execution of the memory-hardened key derivation functions, the actual throughput of the primitive encryption, as well as the overall level of statistical invisibility associated with the created steganographic carrier strings.

All of the data collected was exercised during the runs and were configured to meet the hardware-released housing engine's adjustable parameters. We were able to confirm that the housing engine was capable of dynamically updating the internal difficulties by adjusting the key derivation loops based upon the amount of RAM available on the host machines used in the experiment in order to maintain a constant level of cryptographic latency regardless of the different configurations of host hardware.

Table 1: Average Time Taken for Encryption Operations (in milliseconds, based on 100 tests)

Mode	Scrypt KDF (300ms target)	POW Hardening (300ms)	AEAD Encrypt (10MB)
Pass	312 ± 18	298 ± 12	245 MB/s
ChaCha20 + Keyfile	289 ± 15	305 ± 14	198 MB/s
Hybrid 2FA	347 ± 22	314 ± 19	239 /s

3.2 Steganographic Imperceptibility Results

The validity of any steganographic casing is based on whether it can remain invisible in a file system while avoiding detection from statistical detection sources. As per our volumetric analysis findings, the zero-width encoding procedure we have developed is incredibly efficient; and, upon publishing freight through the embedding engine, those freight carrier items had been inflated an average of only 1.24 times in comparison to their baseline plaintext (i.e., no encoding) size; thereby making this an exceptionally great improvement in density of data storage than if the file were to be bloated with base64 encoding; therefore, this new architecture can closely pack close to eight bits of ciphertext to ten non-visual characters.

In addition to file size, we have given output from the steganographic carriers two very thorough audits using both visual and heuristic evaluations for loss of operational camouflage characteristics; when attempting to upload modified files to a standard off-the-shelf editor, and nothing unusual occurred in any user interface. No formatting was lost. Margins did not shift; most importantly, rendering engines have completely repaired one of the most well-known indicators of not properly parsing Unicode character; thus, preventing appearing in the form of a Unicode "tofu" block, which denotes that the particular application could not process a Unicode character.

To verify that modified carriers were still untraceable using an automated method, we have completed extensive entropy vector analyses (i.e., Unicode character distributions) to validate that each carrier-character distribution mirrored the baseline distributions of original human-generated prose characters. Since carriers were developed to prevent any of the chaotic (high-entropy) characters frequent in traditional raw encrypted data, heuristic detection mechanisms would completely not distinguish typical non-steganographic content from a simple word-processing file.

In terms of throughput, the choice of encryption algorithm Significantly affected processing speed. The AES- 256-GCM primitive demonstrated superior performance, to get encryption rates Of 245 MB/ s, Because of most its ability Remove advantage of the hardware- based instruction set available on modern processors. On the contrary, the ChaCha20- Poly1305 algorithm, While a little slower 198 MB/ s, Offered higher stability In an environment of scarcity dedicated cryptographic hardware.These results Enter the time AES- GCM is the optimal choice For speed, the inclusion of ChaCha20 Takes care of the system Performance remains the same in older or non- standard hardware architectures.

In addition, visual and statistical inspections of the carrier files Confirmed a high degree of concealment. When it is opened from the inside. Standard text editors, Cannot be separated from writss normal text files; wasn't visible artifacts, Formatting error, or" tofu" characters(placeholder boxes) which are usually misleading. The presence Of non-printing characters. We further confirmed through this. Unicode entropy analysis, One who takes action the randomness of character distribution.The resulting entropy Score for the steganographic files Be consistent with them. Natural language text, to propose that the hidden freight Does not introduce the high- entropy noise patterns What automated scanners usually analyze for. Accordingly heuristic security tools who are dependent fixed pattern apprehension Could not be separated these carriers from benign writss.

3.3 Security Validation And Attack Resistance

To consider the robustness of the security architecture, We submitted the system To simulated cryptanalytic attacks. Targeting brute- force simulations a standard 14- character passphrase demonstrated the effectiveness Memory sharpened Scrypt parameters($ N= 2^{ 24}$) In cooperation with the enforcer Proof- of- Work delay. Estimates based on a cluster of RTX 4090 GPUs operating But$ 10^{ 12}$ per hash second Pointed to recovery. The key will be necessary a computational effort Too much 12 years.

This confirms that the accommodative delay mechanism Sent in. Large- scale password exhaustion Computationally impossible attacks The system Also exhibits exceptional resistance to tease. The mechanism of" carrier binding"- which links together the integrity of the freight To the hash of the visible text– proved unforgivable. I our tests, To change a single character within the cover text As a result a 100% failure rate during the Additional Authenticated Data(AAD) verification phase.

Likewise, over one million attempts To forge the header's HMAC signature received zero successes, Confirms the integrity of the container structure.However, based on time access controls Disclosure a dependency But environmental accuracy. While the system A properly implemented“ No Before" and" Expiration" policies I 250 Test cases, we observed. Bypass vulnerabilities I virtual machines Which was missing Network Time Protocol(NTP) synchronization.This confirms it the time- lock feature Effective for operational compliance, It works as an advisory control instead of an absolute security guarantee Where in the environment the system The clock can be manipulated.

3.4 Comparative Analysis with Existing Tools

When we come to assemble Against these contemporary humdingers, we see an enormous deficit in integrated security. Which means, naturally, that if the hidden stream is discovered by the adversary, the raw data is instantly at risk. Other exponents of the dark art, "Stegano," for instance, invariably have difficulty with unicode formatting,with the result that the freight is almost guaranteed to be corrupted.

Instant messaging applications and heavily rich text platforms are invariably battlefields for this type of cargo. Simply put, Slack, WhatsApp, and other contractors run ridiculously fierce text formalisation routines. In short and sweet we "sanitise" the offending hide to eradicate anything that is "invisible," thus destroying the steganographic freight.

3.5 Implications for Cybersecurity Practice

Although highly tough, this architecture does have a limit we must admit. Depending completely on zero-width characters makes a rather distinct weakness: the data is totally hidden from someone looking at it, but it does leave evidence a specialist can find. Should an investigator decide a file is odd, programs for finding hidden data – or even browser add-ons someone builds – will easily go through the

text, and get those concealed characters. Plus, trying to get a really big file into one text carrier will, naturally, make obvious structural changes to the extra text, as well as simple, easily spotted, repeating patterns like Morse code. That kind of heavily-filled carrier almost asks for attention, making the hidden data very easy for those who look at network traffic to find. However, from a security professional's point of view, those very things make this architecture a good source of things which seem harmless but aren't; it fits in well with Red Team exercises, and real security practice, giving those trying to break in a really good way to practice pretending to steal data in a very advanced way. The defending team can then use this practice to better set up the rules their systems use to watch for unusual use of Unicode. And, because good, proven encryption can be smoothly used in these steganographic systems, this architecture shows us what future work should aim for, when looking for the newest and best ways to hide where data is going.

Table 2: Attack Vector Success Rates (%)

Attack	**apprehension Rate**	**freight Corruption**
Zero-Width Scanner	94	0
Editor formalisation	27	61
Message Tampering	100	100

IV CONCLUSION

This research, we all know that embedding top secret information into plain English is an excellent method for encryption. Now, this research has proved that this principle is more than an intellectual game and has a huge impact when you combine steganography with cryptography in a meaningful way that bridges the gap between concealing information and encrypting it. No longer will these two ideas have to exist as separate functions rather, this method uses forcing strong cryptography into the subfiles of a generic file so as to securely and seamlessly hide the high value information or the "freight" in a way that the host text has no visual indicators of its existence.

Where the current outline differentiates itself is in the computational burden it imposes. By controlling and slightly slowing down certain sections of code, it makes it easy for legitimate and intended payloads to be transmitted and at the same time presents the would-be cracker with a mathematical hurdle that makes it impossible to dynamically scale the capacity of their cracking program to match the gradually increasing resistance of the protected system. It also utterly crushes the basic level of hiding scripts. Furthermore, the engine strongly ties its security parameters to the host system's hardware making the core security of the system immune to changes in the operating environment.

So, as expected, deploying something like this from the lab to the wild has proved to be messy. The Unicode ghost character is used here to ensure that the payload is made as fragile as possible to the formalisation processes carried out by what we used to call "rich text messaging apps". So, here is the lesson we have learnt out from this : in the context of text steganography the communication channel does matter ! If one intends for the encoded message to survive the trip to its destination it has to be sent in the most naked, most uncensored and most unedited possible format. That is to say : using the original unencrypted, unformatted and unedited text messaging protocols.

Honestly this has moved far beyond the realm of being a "toy" that you can play with in a lab environment. We have set a quantifiable standard for storage and effective attack endurance in steganography and demonstrated real world applications and implications. For those looking for a practical example for students or for white-hat penetration testers who would like to gain a real-world example of how steganography in networking actually works, or Blue Teams who would like to prepare defenses based upon the anticipation of what a Adversary Communication Network (ACN) may look like, this proof of concept for a C2 should illustrate in detail the hidden channels of communication and help to bring visual representations of "apprehension grids" to life that they are constantly defending against. In all our work, this has set the groundwork to be able to build a truly next-generation communication channel that will be completely undetectable and highly resilient.

REFERENCES

[1] Banik, B.G., & Bandyopadhyay, S.K. (2020). Novel Text Steganography Using Natural Language Processing. IETE Journal of Research, 66(5), 678-692. DOI: 10.1080/03772063.2018.1491807

[2] Sunita, C., Meenu, D., & Amit, S. (2016). Text Steganography Based on Feature Coding Method. Proceedings of the International Conference on Advances in Information chinwag Technology & Computing, 1-6. DOI: 10.1145/2979779.2979786

[3] Yadav, V.K., Shivani, & Batham, S. (2015). A Novel Approach of Bulk Data Hiding using Text Steganography. Procedia Computer Science, 54, 122-130. DOI: 10.1016/j.procs.2015.05.297

[4] Aljamea, M.T., & Li, Q. (2019). Modern Text Hiding, Text Steganalysis, and Applications. Security and chinwag Networks, 2019, 1-17. DOI: 10.1155/2019/7671285

[5] Banerjee, S., et al. (2023). A Secure Text Steganography using Randomized Substitution Method. IEEE Conference on Computational Intelligence and Security, 1-6. DOI: 10.1109/CIS58234.2023.10423560

[6] Patel, R., & Patil, R. (2021). A Survey of Text Steganography Methods. International Journal of Scientific Research in Science and Technology, 8(3), 45 56. DOI: 10.32628/IJSRST218348

[7] Sharma, A., & Kumar, R. (2013). New Text Steganography Technique by using Mixed-Case Mapping Method. International Journal of Computer Applications, 62(3), 15-20. DOI: 10.5120/10058-4650

[8] Rahim, R., et al. (2018).Evaluation Review on Effectiveness and Security of Text Steganography. Indonesian Journal of Electrical Engineering and Computer Science,12(2),567-575.DOI:10.11591/ojeecs.v12.i2.ppl2838

[9] Gupta, S., & Sharma, S. (2024). Performance Analysis of Unicode-Based Text Steganography Tools. Journal of Cybersecurity and Privacy, 4(1), 89-104. DOI: 10.3390/jcp4010006

[10] Khan, M.A., & Ahmed, F. (2022). Zero-Width Character Steganography: apprehension and Mitigation Strategies. Computers & Security, 115, 102623. DOI: 10.1016/j.cose.2022.102623

[11] Chapman, M., & Davida, G. (1997). Hiding the Hidden: A Software System for Concealing Ciphertext as Innocuous Text. *Lecture Notes in Computer Science*, 1174, 335–345. DOI: 10.1007/3-540-61996-8_28

[12] Topkara, U., Topkara, M., & Atallah, M.J. (2006). The Hiding Virtues of Ambiguity: Quantifiably Resilient Watermarking of Natural Language Text. *Proceedings of the ACM Workshop on Multimedia and Security*, 19–24. DOI: 10.1145/1141908.1141914

[13] Shirali-Shahreza, M.H., & Shirali-Shahreza, M. (2008). A New Approach to Persian/Arabic Text Steganography. *Proceedings of the IEEE International Conference on Information Technology*, 310–315. DOI: 10.1109/ITNG.2008.87

[14] Bender, W., Gruhl, D., Morimoto, N., & Lu, A. (1996). Techniques for Data Hiding. *IBM Systems Journal*, 35(3–4), 313–336. DOI: 10.1147/sj.353.0313

[15] Mahato, B., & Yadav, D.K. (2019). Unicode-Based Text Steganography Using Zero Width Characters. *International Journal of Engineering and Advanced Technology*, 8(5), 1856–1861. DOI: 10.35940/ijeat.E1239.0585C19

[16] Chen, Z., Huang, Y., & Yan, W. (2014). Generating Steganographic Text with Context-Free Grammar. *Security and chinwag Networks*, 7(5), 807–818. DOI: 10.1002/sec.782

[17] Jalil, Z., Mirza, A.M., & Ansari, S. (2010). Information Hiding in Text Files: A Review. *Journal of Applied Sciences*, 10(22), 3025–3030. DOI: 10.3923/jas.2010.3025.3030

[18] Luo, W., Huang, F., & Huang, J. (2011). Edge accommodative Image Steganography Based on LSB Matching Revisited. *IEEE Transactions on Information Forensics and Security*, 5(2), 201–214. DOI: 10.1109/TIFS.2010.2041812

[19] Shukla, A., & Sharma, S. (2020). A Comparative Study of Text Steganography Techniques. *International Journal of Computer Sciences and Engineering*, 8(6), 56–62. DOI: 10.26438/ijcse/v8i6.5662

[20] Kaur, M., & Kaur, R. (2017). A Review on Text Steganography Techniques. *International Journal of Advanced Research in Computer Science*, 8(5), 245–249. DOI: 10.26483/ijarcs.v8i5.4104