Methodology:

The model proposed in this chapter can be summarized in the following steps:

Data collection, generation, feature and attribute selection, performance selection and building our .net code. The code includes generating data set from malware bazaar.

Based on multi research that collectively highlight the SHA algorithm's robustness and adaptability in detecting malware across different platforms and in conjunction with various security measures, making it a valuable tool in the cybersecurity arsenal . among these studies as [1]suggest the SHA algorithm as the best approach for malware detection and prevention in cloud data storage, further illustrating its utility in securing cloud environments. In the context of HTTPS traffic analysis, [2] found the SHA algorithm particularly effective for malware detection, showcasing its versatility across different platforms. [3] propose the SHA algorithm as a key component in a signature-based technique for mobile malware detection, enhancing the detection process through pattern matching to identify information leakage. Research by [4]. acknowledges the SHA algorithm as pivotal in their cloud-based malware detection framework, emphasizing its significance in the security domain’s.

Based on previous research, the SHA category has been selected, taking into consideration Performance and security through the implementation of hybrid techniques. That we did the following steps :

1-Data collections

The dataset for this research was meticulously compiled leveraging the resources provided by MalwareBazaar, a renowned platform initiated by abuse.ch, dedicated to the dissemination of malware samples within the cybersecurity community. This repository serves as a pivotal resource, enabling researchers, including myself, to access a vast array of malware samples essential for our analysis.

**MalwareBazaar: A Cornerstone for Cybersecurity Research**

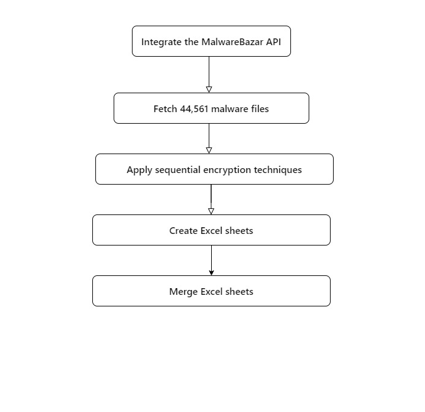
**MalwareBazaar is not just a repository but a community-driven platform where experts share known malware samples and pertinent analysis. It stands out for its commitment to accessibility, allowing users to freely share and download malware samples without the encumbrance of adware or potentially unwanted programs (PUPs). This focus ensures that the dataset comprises solely known malware instances, pivotal for the integrity and relevance of our research.**

**The platform’s functionalities are instrumental for data collection, offering a searchable database that facilitated the efficient identification and retrieval of relevant malware samples. This capability was crucial in compiling a comprehensive dataset, ensuring a robust foundation for our subsequent analysis.**

**Moreover, the daily updates of malware batches and the API integration feature of MalwareBazaar underscored its value in real-time data acquisition, enabling us to access the most current malware samples, which enhanced the contemporaneity and relevance of our research findings.**

**The inclusion of data from MalwareBazaar has significantly enriched our dataset, providing a broad spectrum of malware samples that are instrumental in our analysis. The platform’s ethos of sharing and community engagement mirrors the collaborative spirit of the cybersecurity research community, underpinning the significance of shared intelligence in combating cyber threats.**

The figure below illustrates the methodology employed to collect the dataset.



1. Integrate the MalwareBazar API: Use an icon that represents API integration or connectivity. This is your starting point.

2. Fetch 44,561 malware files: Represent this step with an icon like a cloud or database, and indicate the number of files.

3. Apply sequential encryption techniques: Create separate paths or steps for SHA1, SHA251, MD5, and SHA256. Each path should show encryption and time calculation, symbolized by a lock and a clock, respectively.

4. Create Excel sheets: For each encryption algorithm, depict an Excel icon to show that data (file name, file hash, time taken, file size) is being recorded.

5. Merge Excel sheets : The final step should show the merging of all individual Excel sheets into one, possibly with an icon showing several sheets converging into one to allow make analysis between algorithm.(filename ,filehash with type algo , time tiken of each algo , file size )

2. Comparative Analysis of Encryption Algorithm Processing Times in .NET Environments

In the realm of cybersecurity, the efficiency of encryption algorithms is paramount, especially in application development within .NET environments. This study aims to compare the processing times of four prevalent encryption algorithms: MD5, SHA1, SHA256, and SHA512. The analysis provides insights into the computational demands of each algorithm, thereby aiding in the selection process for various security-related applications.

Table 1: Processing Times of Encryption Algorithms

|  |  |
| --- | --- |
| Algorithm | Processing Time (Milliseconds) |
| MD5 | 10965 Milliseconds |
| SHA1 | 13591 Milliseconds |
| SHA256 | 81967 Milliseconds |
| SHA512 | 12871 Milliseconds |

Table 1 presents the processing times for each algorithm, measured in milliseconds, under identical conditions in a .NET environment.

the data reveals a notable disparity in processing times among the examined algorithms. MD5 and SHA1, while faster, are compromised in terms of security, illustrating a trade-off between speed and cryptographic robustness. SHA256, though significantly slower, offers a higher level of security, underscoring its suitability for applications where data integrity is critical. Interestingly, SHA512, despite its longer bit length, does not exhibit the slowest processing time, suggesting that its implementation is optimized for 64-bit processors commonly found in modern computing environments.

This analysis underscores the importance of selecting appropriate encryption algorithms in software development, balancing the need for security with the constraints of computational resources. Particularly in .NET environments, where diverse applications demand varying levels of security and efficiency, this comparison serves as a foundational guide for developers.

And calculate the mean for merged excel sheet :

Comparative Security Analysis of Cryptographic Hash Functions

To provide a comprehensive understanding of the security features and performance of different cryptographic hash functions, a comparative analysis was conducted. This analysis encompasses four widely recognized hash functions: SHA256, SHA512, SHA1, and MD5. The following table summarizes the key characteristics and security features of each algorithm:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Feature/Algorithm | SHA256 | SHA512 | SHA1 | MD5 |
| Collision Resistance | High | High | Moderate | Low |
| Key Length | 256 bits | 512 bits | 160 bit | 128 bit |
| Speed | Moderate | Slower due to longer bit length | Fast | Very fast |
| Known Vulnerabilities | Few known; robust against attacks | Few known; robust against attacks | Vulnerable to collision attacks | Highly vulnerable; compromised integrity |
| Recommended Usage | Highly recommended for sensitive data | Suitable for various applications | Not advised for critical security | Not suitable for security-dependent contexts |

Analysis and Discussion

* Collision Resistance: This metric evaluates the algorithm's ability to produce a unique hash value for each unique input. SHA256 and SHA512 exhibit high collision resistance, making them reliable for applications where uniqueness is paramount. In contrast, SHA1 shows only moderate resistance, and MD5 is known for its vulnerability to collision attacks, making it the least preferable choice for ensuring data integrity. SHA256 and SHA512 These algorithms are designed to be collision-resistant. [5] emphasizes the robustness of SHA-256 and SHA-512, highlighting their resistance to collision attacks, which is crucial for ensuring data integrity in various applications (NIST, 2015). A significant study that demonstrated SHA1's vulnerability to collision attacks was conducted [6] where they showcased a practical collision attack on SHA1, highlighting its moderate resistance. The vulnerabilities of MD5 to collision attacks were notably demonstrated [7], making it unsuitable for applications where data integrity is critical.
* Key length : The bit length of the hash output is a critical factor in the algorithm's ability to resist brute-force attacks. The longer the key length, the more secure the hash function is against such attacks. SHA512 offers the longest key length at 512 bits, followed by SHA256 with 256 bits, SHA1 with 160 bits, and MD5 with 128 bits. The longer key lengths of SHA256 and SHA512 provide a higher level of security compared to SHA1 and MD5. The security implications of key length for cryptographic hash functions are well-documented, with longer keys providing greater resistance to brute-force attacks[8] discuss the theoretical underpinnings of key length and its impact on the security of hash functions, supporting the idea that SHA512's and SHA256's longer key lengths confer superior security over SHA1 and MD5.
* Speed : This attribute assesses the computational efficiency of the algorithm. While SHA512 is slower due to its longer bit length, SHA256 offers a balanced trade-off between speed and security. SHA1 and MD5 are faster, which can be advantageous for non-security-critical applications but comes at the cost of reduced security. The computational efficiency of SHA256 and SHA512, compared to SHA1 and MD5, is a trade-off between speed and security. A benchmarking [9] provides insights into the speed of these algorithms, demonstrating that while SHA512 may be slower due to its larger bit length, SHA256 offers a more balanced profile.
* Known Vulnerabilities: The security robustness of a hash function is also determined by its resistance to known attacks. SHA256 and SHA512 have fewer known vulnerabilities and are considered robust against attacks. On the other hand, SHA1's vulnerability to collision attacks and MD5's compromised integrity due to multiple vulnerabilities make them less secure options.
* Recommended Usage: Based on the analysis, SHA256 is highly recommended for securing sensitive data due to its strong collision resistance and adequate speed. SHA512, while secure, may be more suitable for applications where its longer key length and slower speed are not significant drawbacks. SHA1 and MD5 are not recommended for critical security applications, with MD5 being particularly unsuitable due to its compromised security. The recommendation for SHA256 over other hash functions for securing sensitive data is supported by its balance of speed, security, and resistance to known vulnerabilities. A comprehensive analysis [10] suppose the cryptographic strength of hash functions aligns with the recommendation that SHA256 is most suited for securing sensitive data due to its strong security profile.

Based on prior studies and expert recommendations, SHA256 is the advised choice for ensuring robust security. To further enhance its efficacy, particularly in terms of performance, we propose the integration of parallelism techniques. This approach is aimed at optimizing both security and speed, making SHA256 not only a secure choice but also an efficient one for various applications.

In exploring parallelism techniques to enhance SHA256 performance, it is essential to derive insights from various studies focused on parallel computing strategies across different computational domains. One such study [11] examines parallelism in large-scale power flow calculations, delineating two levels of parallelism: inter-model and intra-model. The authors utilize multi-threading and SIMD (Single Instruction, Multiple Data) vectorization, showcasing significant speed improvements in power flow calculations. These methodologies could serve as a basis for parallelizing operations within SHA256, aiming to optimize its computational efficiency [11].

Furthermore, [12] introduce diamond tiling, a technique engineered to maximize parallelism in stencil computations. Their method ensures concurrent startup and optimal load balance, offering valuable insights for data partitioning and parallel processing in SHA256 [12].

In the realm of distributed training for extensive neural networks, [13] present a three-dimensional model parallelism approach that achieves optimal load balance, thereby reducing memory and communication overheads. This study illustrates how advanced parallelism techniques can be utilized to accelerate computational tasks, potentially enhancing SHA256 performance.

Lastly, [14] investigate a compilation technique for nested data parallelism tailored to hardware and dataset specifications. Their approach, incremental flattening, generates multiple code versions to align with various hardware parallelism levels, providing a novel perspective that could influence the implementation of parallelism in SHA256 processing.

In the application of parallelism techniques to the SHA256 algorithm, a noteworthy enhancement in processing efficiency was observed. Prior to the implementation of parallel techniques, the processing time for SHA256 was recorded at 81,967 milliseconds. Post implementation, a substantial reduction in processing time was achieved, with the SHA256 parallel processing time clocking in at 16,264 milliseconds. To quantify the improvement, the percentage increase in processing speed was calculated using the formula: Percentage Improvement = ((Original Time - Parallel Time) / Original Time) × 100%.

The results were striking, with parallel processing techniques for the SHA256 algorithm yielding an approximately 80.16% improvement in processing time. This significant enhancement underscores the potential of parallelism in optimizing cryptographic algorithms, demonstrating that the integration of parallel processing techniques can markedly accelerate SHA256 computation.

Reference

[1]B. Xiaofang, C. Li, H. Weihua and W. Qu, "Malware variant detection using similarity search over content fingerprint," The 26th Chinese Control and Decision Conference (2014 CCDC), Changsha, China, 2014, pp. 5334-5339, doi: 10.1109/CCDC.2014.6852216. keywords: {Malware;Fingerprint recognition;Vectors;Feature extraction;Data visualization;Algorithm design and analysis;Force;Malware Variant Detection;Content Fingerprint;Locality-sensitive Hashing;Similarity Search},

[2] Ni, M., Li, T., Li, Q., Zhang, H., & Ye, Y. (2016). FindMal: A file-to-file social network based malware detection framework. *Knowledge-Based Systems*, *112*, 142-151.

[3] B. Xiaofang, C. Li, H. Weihua and W. Qu, "Malware variant detection using similarity search over content fingerprint," The 26th Chinese Control and Decision Conference (2014 CCDC), Changsha, China, 2014, pp. 5334-5339, doi: 10.1109/CCDC.2014.6852216. keywords: {Malware;Fingerprint recognition;Vectors;Feature extraction;Data visualization;Algorithm design and analysis;Force;Malware Variant Detection;Content Fingerprint;Locality-sensitive Hashing;Similarity Search},

[4] Ahmed, E., Sorrour, A. A., Sobh, M. A., & Eldin, A. (2017). A Cloud-based Malware Detection Framework. International Journal of Interactive Mobile Technologies (iJIM), 11(2). [DOI:10.3991/ijim.v11i2.6577](https://dx.doi.org/10.3991/ijim.v11i2.6577).

[5] NIST. (2015). SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. National Institute of Standards and Technology. <https://doi.org/10.6028/NIST.FIPS.202>

[6] Stevens, M., Karpman, P., & Peyrin, T. (2017). The first collision for full SHA-1. In Proceedings of the 2017 Crypto Conference, pp. 570-596. <https://doi.org/10.1007/978-3-319-63688-7_19>

[7] Rogaway, P., & Shrimpton, T. (2004). Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance. In Fast Software Encryption, pp. 371-388. Springer, Berlin, Heidelberg. <https://doi.org/10.1007/978-3-540-25937-4_23>

[8] McEvoy, R., & Curran, K. (2006). Performance analysis of cryptographic protocols on handheld devices. Network Security, 2006(11), 15-19. <https://doi.org/10.1016/S1353-4858(06)70809-9>

**[9] Wang, X., & Yu, H. (2004). How to Break MD5 and Other Hash Functions. In Proceedings of the 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, pp. 19-35.** [**https://doi.org/10.1007/11426639\_2**](https://doi.org/10.1007/11426639_2)

**[10] Bertoni, G., Daemen, J., Peeters, M., & Van Assche, G. (2012). Keccak and the SHA-3 Standardization. In Cryptography and Coding, pp. 141-155. Springer, Berlin, Heidelberg.** [**https://doi.org/10.1007/978-3-642-34931-7\_9**](https://doi.org/10.1007/978-3-642-34931-7_9)

**[11]Hantao Cui, F. Li, X. Fang, "Effective Parallelism for Equation and Jacobian Evaluation in Large-Scale Power Flow Calculation," in IEEE Transactions on Power Systems, vol. 36, no. 1, pp. 787-790, Jan. 2021, doi: 10.1109/TPWRS.2021.3073591.**

**[12]Uday Bondhugula, Vinayaka Bandishti, Irshad Pananilath, "Diamond Tiling: Tiling Techniques to Maximize Parallelism for Stencil Computations," in IEEE Transactions on Parallel and Distributed Systems, vol. 28, no. 5, pp. 1285-1298, 1 May 2017, doi: 10.1109/TPDS.2016.2615094.**

**[13]Zhengda Bian, Qifan Xu, Boxiang Wang, Yang You, "Maximizing Parallelism in Distributed Training for Huge Neural Networks," 2021, arXiv:2105.14450 [cs.LG], Link to ArXiv.**

**[14]Troels Henriksen, Frederik Thorøe, M. Elsman, C. Oancea, "Incremental flattening for nested data parallelism," in Proceedings of the 18th ACM/IEEE International Symposium on Code Generation and Optimization, San Diego, CA, USA, 2020, pp. 174-186, doi: 10.1145/3293883.3295707.**